Libraries Guide

About This Manual

The Libraries Guide is part of the ISE documentation collection.

Manual Contents

This manual contains the following:

- Discussion of the Unified libraries
- Slice count information for FPGAs
- Selection guidelines of the various functional categories of design elements
- Architecture-specific chapters
- Individual chapters for each design element

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this Web site. You can also directly access these resources using the provided URLs.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging `http://support.xilinx.com/support/techsup/tutorials/index.htm`
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at `http://support.xilinx.com/support/searchtd.htm`
Application Notes	Descriptions of device-specific design techniques and approaches `http://support.xilinx.com/apps/appsweb.htm`
Data Book	Pages from *The Programmable Logic Data Book*, which contain device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging `http://support.xilinx.com/partinfo/databook.htm`
Xcell Journals	Quarterly journals for Xilinx programmable logic users `http://support.xilinx.com/xcell/xcell.htm`
Technical Tips	Latest news, design tips, and patch information for the Xilinx design environment `http://support.xilinx.com/support/techsup/journals/index.htm`

Conventions

This manual uses the following conventions. An example illustrates each convention.

Typographical

The following conventions are used for all documents.

- `Courier font` indicates messages, prompts, and program files that the system displays.

  ```
  speed grade: - 100
  ```

- **`Courier bold`** indicates literal commands that you enter in a syntactical statement. However, braces "{ }" in Courier bold are not literal and square brackets "[ ]" in Courier bold are literal only in the case of bus specifications, such as bus [7:0].

  `rpt_del_net=`

  `Courier bold` also indicates commands that you select from a menu.

  `File` → **`Open`**

- *Italic font* denotes the following items.

  - Variables in a syntax statement for which you must supply values

    `edif2ngd` *design_name*

  - References to other manuals

    See the *Development System Reference Guide* for more information.

  - Emphasis in text

    If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets "[ ]" indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

  `edif2ngd` *[option_name] design_name*

- Braces "{ }" enclose a list of items from which you must choose one or more.

  `lowpwr ={on|off}`

- A vertical bar " | " separates items in a list of choices.

  `lowpwr ={on|off}`

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'
IOB #2: Name = CLKIN'
.
.
.
```

- A horizontal ellipsis "...." indicates that an item can be repeated one or more times.

  `allow block` *block_name loc1 loc2locn ....;*

Online Documents

The following conventions are used for online documents.

- Red-underlined text indicates an interbook link, which is a cross-reference to another book. Click the red-underlined text to open the specified cross-reference.
- Blue-underlined text indicates an intrabook link, which is a cross-reference within a book. Click the blue-underlined text to open the specified cross-reference.

Contents

Design Elements

Xilinx Unified Libraries

This chapter describes the Unified Libraries and the applicable device architectures for each library. It also briefly discusses the contents of the other chapters, the general naming conventions, and performance issues.

This chapter consists of the following major sections.

Overview

Xilinx maintains software libraries with thousands of functional design elements (primitives and macros) for different device architectures. New functional elements are assembled with each release of development system software. The catalog of design elements is known as the Unified Libraries. Elements in these libraries are common to all Xilinx device architectures. This "unified" approach means that you can use your circuit design created with "unified" library elements across all current Xilinx device architectures that recognize the element you are using.

Elements that exist in multiple architectures look and function the same, but their implementations might differ to make them more efficient for a particular architecture. A separate library still exists for each architecture (or architectural group) and common symbols are duplicated in each one, which is necessary for simulation (especially board level) where timing depends on a particular architecture.

If you have active designs that were created with former Xilinx library primitives or macros, you may need to change references to the design elements that you were using to reflect the Unified Libraries elements.

The *Libraries Guide* describes the primitive and macro logic elements available in the Unified Libraries for the Xilinx FPGA and CPLD devices. Common logic functions can be implemented with these elements and more complex functions can be built by combining macros and primitives. Several hundred design elements (primitives and macros) are available across multiple device architectures, providing a common base for programmable logic designs.

This libraries guide provides a functional selection guide and describes the design elements.

Applicable Architectures

Design elements for the Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex -II, Virtex-II Pro, XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II libraries are included in the Xilinx Unified Libraries. Each library supports specific device architectures. For detailed information on the architectural families referenced below and the devices in each, refer to the current version of *The Programmable Logic Data Book* (an online version is available from the Xilinx web site, http://support.xilinx.com).

Selection Guidelines

Selection guidelines are provided in functional categories. These categories list the available elements in each category along with a brief description of each element and an applicability table identifying which libraries (Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex -II, Virtex-II Pro, XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II) contain the element.

Design Elements

Design elements are organized in alphanumeric order, with all numeric suffixes in ascending order. For example, FDR precedes FDRS, and ADD4 precedes ADD8, which precedes ADD16.

The following information is provided for each library element.

- Graphic symbol
- Applicability table (with primitive versus macro identification)
- Functional description
- Truth table (when applicable)
- Topology (when applicable)
- Schematic for macros
- VHDL and Verilog instantiation and inference code, if applicable
- Commonly used constraints, if applicable

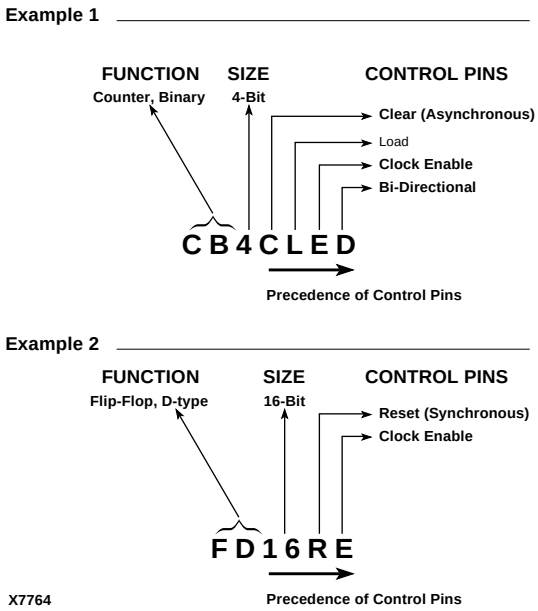
Schematic Examples

Schematics are included for each library if the implementation differs.

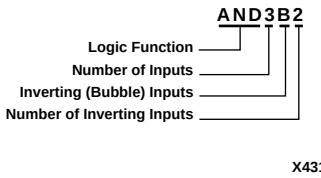
Design elements with bussed or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic — generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

Naming Conventions

Examples of the general naming conventions for the unified library elements are shown in the following figures.



Naming Conventions



Combinatorial Naming Conventions

Attributes and Constraints

Attributes and constraints are instructions placed on components or nets to indicate their placement, implementation, naming, directionality, and so forth. The *Constraints Guide* provides information on all attributes and constraints.

Carry Logic

The Spartan-II, Virtex, and Virtex-II architectures include dedicated carry logic components.

Spartan-II, Spartan-IIE, Virtex, and Virtex-E

Carry Logic for Spartan-II, Spartan-IIE, Virtex, and Virtex-E is a simple structure associated with each look-up table. The design entry library contains the following dedicated carry logic primitives: MULT_AND, MUXCY, MUXCY_D, MUXCY_L, XORCY, XORCY_D, and XORCY_L. The function performed is determined by their connectivity and the contents of the look-up table. For an example of how to use carry logic, see the "CC8CE, CC16CE" section.

For detailed information on Carry Logic in Virtex and Spartan-II, refer to *The Programmable Logic Data Book* available on the Xilinx web site, http://support.xilinx.com.

Virtex-II and Virtex-II Pro

The dedicated carry logic primitives for Virtex-II and Virtex-II Pro are MULT_AND, MUXCY, MUXCY_D, MUXCY_L, XORCY, XORCY_D, and XORCY_L.

ORCY can only be used exclusively with Virtex-II and Virtex-II Pro.

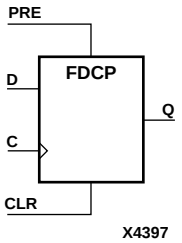
For detailed information on Carry Logic in Virtex-II and Virtex-II Pro, refer to *The Programmable Logic Data Book* available on the Xilinx web site, http://support.xilinx.com.

Flip-Flop, Counter, and Register Performance

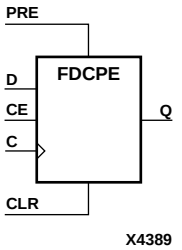
All counter, register, and storage functions derived from the flip-flops are available in the Configurable Logic Blocks (CLBs).

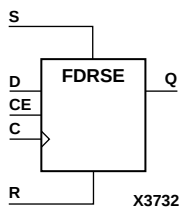
The D flip-flop is the basic building block for all architectures. Differences occur from the availability of asynchronous Clear (CLR) and Preset (PRE) inputs, and the source of the synchronous control signals, such as Clock Enable (CE), Clock (C), Load enable (L), synchronous Reset (R), and synchronous Set (S). The basic flip-flop configuration for each architecture follows.

The basic XC9000 flip-flops have both Clear and Preset inputs.



Virtex and Spartan-II have two basic flip-flop types. One has both Clear and Preset inputs and one has both asynchronous and synchronous control functions.





The asynchronous and synchronous control functions, when used, have a priority that is consistent across all devices and architectures. These inputs can be either active-High or active-Low as defined by the macro. The priority, from highest to lowest, is as follows.

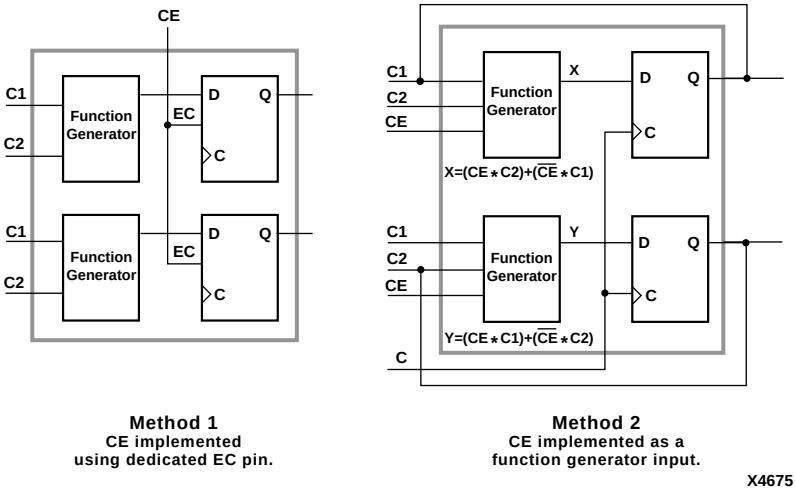
- Asynchronous Clear (CLR)
- Asynchronous Preset (PRE)
- Synchronous Set (S)
- Synchronous Reset (R)
- Clock Enable (CE)

Note The asynchronous CLR and PRE inputs, by definition, have priority over all the synchronous control and clock inputs.

For FPGA families, the Clock Enable (CE) function is implemented using two different methods in the Xilinx Unified Libraries; both are shown in the following figure.

- In method 1, CE is implemented by connecting the CE pin of the macro directly to the dedicated Enable Clock (EC) pin of the internal Configurable Logic Block (CLB) flip-flop. This allows one CE per CLB. CE takes precedence over the L, S, and R inputs. All flip-flops with asynchronous clear or preset use this method.
- In method 2, CE is implemented using function generator logic. This allows two CEs per CLB. CE has the same priority as the L, S, and R inputs. All flip-flops with synchronous set or reset use this method.

The method used in a particular macro is indicated by the inclusion of asynchronous clear, asynchronous preset, synchronous set, or synchronous reset in the macro's description.



Clock Enable Implementation Methods

Unconnected Pins

Xilinx recommends that you *always* connect input pins in your designs. This ensures that front end simulation functionally matches back end timing simulation. If an input pin is left unconnected, mapper errors may result.

If an output pin is left unconnected in your design, the corresponding function is trimmed. If the component has only one output, the entire component is trimmed. If the component has multiple outputs, the portion that drives the output is trimmed. As an example of the latter case, if the overflow pin (OFL) in an adder macro is unconnected, the logic that generates that term is trimmed, but the rest of the adder is retained (assuming all of the sum outputs are connected).

Slice Count

Configurable Logic Blocks (CLBs) implement most of the logic in an FPGA.

Each Virtex and Spartan-II CLB contains two slices. Each Virtex-II CLB contains four slices. In the following table, the numbers for Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro are the number of slices required to implement the component.

The following Slice Count table lists FPGA design elements in alphanumeric order with the number of CLBs or slices needed for their implementation in each applicable library.

Note This information is for reference only. The actual count could vary, depending upon the switch settings of the implementation tools; for example, the effort level in PAR (Place and Route) or usage of the components with other components.

The asterisk for the RAM16X1D and RAM16X1D_1 in the Virtex-II and Virtex-II Pro column indicates that these two design elements consume 1/2 of two slices.

The double asterisks for design elements indicate that these primitives cannot be used by themselves. However, there is only one available per slice.

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
ACC4	5	5	6
ACC8	9	9	10
ACC16	17	17	18
ADD4	3	3	3
ADD8	5	5	5
ADD16	9	9	9
ADSU4	3	3	3
ADSU8	5	5	5
ADSU16	9	9	9
AND2	1	1	1
AND3	1	1	1
AND4	1	1	1
AND5	1	1	1
AND6	1	1	1
AND7	1	1	1
AND8	2	2	2
AND9	2	2	2
AND12	2	2	2
AND16	2	2	2
BRLSHFT4	8	8	4
BRLSHFT8	12	12	12
BSCAN_SPARTAN2	-	-	-

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
BSCAN_VIRTEX	-	-	-
BSCAN_VIRTEX2	-	-	-
BUF	-	-	-
BUF4	-	-	-
BUF8	-	-	-
BUF16	-	-	-
BUFCF	-	-	-
BUFE	-	-	-
BUFE4	-	-	-
BUFE8	-	-	-
BUFE16	-	-	-
BUFG	-	-	-
BUFGCE	-	-	-
BUFGCE_1	-	-	-
BUFGDLL	-	-	-
BUFGMUX	-	-	-
BUFGMUX_1	-	-	-
BUFGP	-	-	-
BUFT	-	-	-
BUFT4	-	-	-
BUFT8	-	-	-
BUFT16	-	-	-
CAPTURE_SPARTAN2	-	-	-
CAPTURE_VIRTEX	-	-	-
CAPTURE_VIRTEX2	-	-	-
CB2CE	2	2	3
CB2CLE	3	3	3
CB2CLED	3	3	3
CB2RE	2	2	3
CB4CE	3	3	4
CB4CLE	5	5	5
CB4CLED	6	6	7
CB4RE	3	3	4
CB8CE	6	6	7
CB8CLE	9	9	10
CB8CLED	12	12	12
CB8RE	6	6	7
CB16CE	13	13	14
CB16CLE	18	18	19

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
CB16CLED	24	24	25
CB16RE	13	13	14
CC8CE	8	8	5
CC8CLE	9	9	9
CC8CLED	9	9	17
CC8RE	9	9	9
CC16CE	16	16	9
CC16CLE	17	17	17
CC16CLED	17	17	33
CC16RE	17	17	17
CD4CE	3	3	4
CD4CLE	5	5	5
CD4RE	3	3	4
CD4RLE	7	7	7
CJ4CE	2	2	4
CJ4RE	2	2	4
CJ5CE	3	3	5
CJ5RE	3	3	5
CJ8CE	4	4	4
CJ8RE	4	4	4
CLKDLL	-	-	-
CLKDLLE	-	-	-
CLKDLLHF	-	-	-
COMP2	1	1	1
COMP4	2	2	2
COMP8	3	3	4
COMP16	6	6	9
COMPM2	1	1	2
COMPM4	5	5	5
COMPM8	11	11	13
COMPM16	24	24	32
COMPMC8	8	8	8
COMPMC16	16	16	16
CR8CE	8	8	8
CR16CE	16	16	16
D2_4E	2	2	2
D3_8E	4	4	4
D4_16E	16	16	16
DCM	-	-	-

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
DEC_CC4	1	1	1
DEC_CC8	1	1	1
DEC_CC16	2	2	2
DECODE4	1	1	1
DECODE8	2	2	2
DECODE16	2	2	2
DECODE32	4	4	4
DECODE64	8	8	8
FD	1	1	1
FD_1	1	1	1
FD4CE	2	2	4
FD4RE	2	2	4
FD8CE	4	4	4
FD8RE	4	4	4
FD16CE	8	8	8
FD16RE	8	8	8
FDC	1	1	1
FDC_1	1	1	1
FDCE	1	1	1
FDCE_1	1	1	1
FDCP	1	1	1
FDCP_1	1	1	1
FDCPE	1	1	1
FDCPE_1	1	1	1
FDDRCPE	-	-	-
FDDRRSE	-	-	-
FDE	1	1	1
FDE_1	1	1	1
FDP	1	1	1
FDP_1	1	1	1
FDPE	1	1	1
FDPE_1	1	1	1
FDR	1	1	1
FDR_1	1	1	1
FDRE	1	1	1
FDRE_1	1	1	1
FDRS	1	1	1
FDRS_1	1	1	1
FDRSE	1	1	1

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
FDRSE_1	1	1	1
FDS	1	1	1
FDS_1	1	1	1
FDSE	1	1	1
FDSE_1	1	1	1
FJKC	1	1	1
FJKCE	1	1	1
FJKP	1	1	1
FJKPE	1	1	1
FJKRSE	1	1	1
FJKSRE	1	1	1
FMAP	-	-	-
FTC	1	1	1
FTCE	1	1	1
FTCLE	1	1	1
FTCLEX	1	1	1
FTP	1	1	1
FTPE	1	1	1
FTPLE	1	1	1
FTRSE	1	1	1
FTRSLE	2	2	1
FTSRE	1	1	1
FTSRLE	2	2	2
GND	-	-	-
GT_AURORA_n	-	-	-
GT_CUSTOM_n	-	-	-
GT_ETHERNET_n	-	-	-
GT_FIBRE_CHAN_n	-	-	-
GT_INFINIBAND_n	-	-	-
GT_XAUI_n	-	-	-
IBUF	-	-	-
IBUF4	-	-	-
IBUF8	-	-	-
IBUF16	-	-	-
IBUF_selectIO	-	-	-
IBUFDS	-	-	-
IBUFG	-	-	-
IBUFG_selectIO	-	-	-
IBUFGDS	-	-	-

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
ICAP_VIRTEX2	-	-	-
IFD	-	-	-
IFD_1	-	-	-
IFD4	-	-	-
IFD8	-	-	-
IFD16	-	-	-
IFDDRCPE	-	-	-
IFDDRRSE	-	-	-
IFDI	-	-	-
IFDI_1	-	-	-
IFDX	-	-	-
IFDX4	-	-	-
IFDX8	-	-	-
IFDX16	-	-	-
IFDX_1	-	-	-
IFDXI	-	-	-
IFDXI_1	-	-	-
ILD	1	1	1
ILD_1	1	1	1
ILD4	2	2	2
ILD8	4	4	4
ILD16	8	8	8
ILDI	-	-	-
ILDI_1	-	-	-
ILDX	-	-	-
ILDX4	-	-	-
ILDX8	-	-	-
ILDX16	-	-	-
ILDX_1	-	-	-
ILDXI	-	-	-
ILDXI_1	-	-	-
INV	1	1	1
INV4	1	1	1
INV8	1	1	1
INV16	1	1	1
IOBUF	-	-	-
IOBUF_selectIO	-	-	-
IOPAD	-	-	-
IOPAD4	-	-	-

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
IOPAD8	-	-	-
IOPAD16	-	-	-
IPAD	-	-	-
IPAD4	-	-	-
IPAD8	-	-	-
IPAD16	-	-	-
JTAGPPC	-	-	-
KEEPER	-	-	-
LD	1	1	1
LD_1	1	1	1
LD4	2	2	4
LD8	4	4	4
LD16	8	8	8
LD4CE	2	2	4
LD8CE	4	4	4
LD16CE	8	8	8
LDC	1	1	1
LDC_1	1	1	1
LDCE	1	1	1
LDCE_1	1	1	1
LDCP	1	1	1
LDCP_1	1	1	1
LDCPE	1	1	1
LDCPE_1	1	1	1
LDE	1	1	1
LDE_1	1	1	1
LDP	1	1	1
LDP_1	1	1	1
LDPE	1	1	1
LDPE_1	1	1	1
LUT1	1	1	1
LUT2	1	1	1
LUT3	1	1	1
LUT4	1	1	1
LUT1_D	1	1	1
LUT2_D	1	1	1
LUT3_D	1	1	1
LUT4_D	1	1	1
LUT1_L	1	1	1

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
LUT2_L	1	1	1
LUT3_L	1	1	1
LUT4_L	1	1	1
M2_1	1	1	1
M2_1B1	1	1	1
M2_1B2	1	1	1
M2_1E	1	1	1
M4_1E	1	1	1
M8_1E	2	2	2
M16_1E	5	5	5
MULT_AND **	-	-	-
MULT18X18	-	-	-
MULT18X18S	-	-	-
MUXCY **	-	-	-
MUXCY_D **	-	-	-
MUXCY_L **	-	-	-
MUXF5 **	-	-	-
MUXF5_D **	-	-	-
MUXF5_L **	-	-	-
MUXF6 **	-	-	-
MUXF6_D **	-	-	-
MUXF6_L **	-	-	-
MUXF7 **	-	-	-
MUXF7_D **	-	-	-
MUXF7_L **	-	-	-
MUXF8 **	-	-	-
MUXF8_D **	-	-	-
MUXF8_L **	-	-	-
NAND2	1	1	1
NAND3	1	1	1
NAND4	1	1	1
NAND5	1	1	1
NAND6	1	1	1
NAND7	1	1	1
NAND8	2	2	2
NAND9	2	2	2
NAND12	2	2	2
NAND16	2	2	2
NOR2	1	1	1

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
NOR3	1	1	1
NOR4	1	1	1
NOR5	1	1	1
NOR6	1	1	1
NOR7	1	1	1
NOR8	2	2	2
NOR9	2	2	2
NOR12	2	2	2
NOR16	2	2	2
OBUF	-	-	-
OBUF4	-	-	-
OBUF8	-	-	-
OBUF16	-	-	-
OBUF_selectIO	-	-	-
OBUFDS	-	-	-
OBUFE	-	-	-
OBUFE4	-	-	-
OBUFE8	-	-	-
OBUFE16	-	-	-
OBUFT	-	-	-
OBUFT4	-	-	-
OBUFT8	-	-	-
OBUFT16	-	-	-
OBUFT_selectIO	-	-	-
OBUFTDS	-	-	-
OFD	-	-	-
OFD_1	-	-	-
OFD4	-	-	-
OFD8	-	-	-
OFD16	-	-	-
OFDDRCPE	-	-	-
OFDDRRSE	-	-	-
OFDDRTCPE	-	-	-
OFDDRTRSE	-	-	-
OFDE	-	-	-
OFDE_1	-	-	-
OFDE4	-	-	-
OFDE8	-	-	-
OFDE16	-	-	-

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
OFDI	-	-	-
OFDI_I	-	-	-
OFDT	-	-	-
OFDT_1	-	-	-
OFDT4	-	-	-
OFDT8	-	-	-
OFDT16	-	-	-
OFDX	-	-	-
OFDX4	-	-	-
OFDX8	-	-	-
OFDX16	-	-	-
OFDX_1	-	-	-
OFDXI	-	-	-
OFDXI_I	-	-	-
OPAD	-	-	-
OPAD4	-	-	-
OPAD8	-	-	-
OPAD16	-	-	-
OR2	1	1	1
OR3	1	1	1
OR4	1	1	1
OR5	1	1	1
OR6	1	1	1
OR7	1	1	1
OR8	2	2	2
OR9	2	2	2
OR12	2	2	2
OR16	2	2	2
ORCY **	-	-	-
PPC405	-	-	-
PULLDOWN	-	-	-
PULLUP	-	-	-
RAM16X1D	1	1	2*
RAM16X1D_1	1	1	2*
RAM16X1S	1	1	1
RAM16X1S_1	1	1	1
RAM16X2D	2	2	4
RAM16X2S	2	2	2
RAM16X4D	4	4	8

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
RAM16X4S	4	4	3
RAM16X8D	8	8	16
RAM16X8S	8	8	5
RAM32X1D	-	-	2
RAM32X1D_1	-	-	2
RAM32X1S	1	1	1
RAM32X1S_1	1	1	1
RAM32X2S	2	2	2
RAM32X4S	8	8	3
RAM32X8S	-	-	6
RAM64X1D	-	-	4
RAM64X1D_1	-	-	4
RAM64X1S	-	-	2
RAM64X1S_1	-	-	2
RAM64X2S	-	-	4
RAM128X1S	-	-	4
RAM128X1S_1	-	-	4
RAMB4_Sn	-	-	-
RAMB4_Sm_Sn	-	-	-
RAMB16_Sn	-	-	-
RAMB16_Sm_Sn	-	-	-
ROM16X1	1	1	1
ROM32X1	1	1	1
ROM64X1	2	2	2
ROM128X1	-	-	4
ROM256X1	-	-	8
SOP3	1	1	1
SOP4	1	1	1
SR4CE	2	2	4
SR4CLE	3	3	3
SR4CLED	5	5	5
SR4RE	2	2	4
SR4RLE	3	3	3
SR4RLED	5	5	5
SR8CE	4	4	4
SR8CLE	5	5	5
SR8CLED	9	9	9
SR8RE	4	4	4
SR8RLE	5	5	5

Slice Count for FPGA Components

	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro
Name	**Number of Slices to Implement**		
SR8RLED	9	9	9
SR16CE	8	8	8
SR16CLE	9	9	9
SR16CLED	17	17	17
SR16RE	8	8	8
SR16RLE	9	9	9
SR16RLED	17	17	17
SRL16	1	1	1
SRL16_1	1	1	1
SRL16E	1	1	1
SRL16E_1	1	1	1
SRLC16	-	-	1
SRLC16_1	-	-	1
SRLC16E	-	-	1
SRLC16E_1	-	-	1
STARTUP_SPARTAN2	-	-	-
STARTUP_VIRTEX	-	-	-
STARTUP_VIRTEX2	-	-	-
UPAD	-	-	-
VCC	-	-	-
XNOR2	1	1	1
XNOR3	1	1	1
XNOR4	1	1	1
XNOR5	1	1	1
XNOR6	1	1	1
XNOR7	1	1	1
XNOR8	2	2	2
XNOR9	2	2	2
XOR2	1	1	1
XOR3	1	1	1
XOR4	1	1	1
XOR5	1	1	1
XOR6	1	1	1
XOR7	1	1	1
XOR8	2	2	2
XOR9	2	2	2
XORCY **	-	-	-
XORCY_D **	-	-	-
XORCY_L **	-	-	-

* The RAM16X1D and RAM16X1D_1 consume 1/2 of two slices.

** These primitives cannot be used by themselves. However, there is only one available per slice.

Architecture Specific Information

The following sections in the book list the design elements and constraints that can be used with supported architectures.

Spartan-II and Spartan-IIE

The two tables indicate the supported design elements and constraints for Spartan-II and Spartan-IIE. For a complete description, see the Product Data Sheets (http://www.xilinx.com/partinfo/databook.htm#spartan).

Spartan-II, Spartan-IIE Design Elements

ACC4, 8, 16	ADD4, 8, 16	ADSU4, 8, 16
AND2-9	AND12, 16	BRLSHFT4, 8
BSCAN_SPARTAN2	BUF	BUFCF
BUFE, 4, 8, 16	BUFG	BUFGDLL
BUFGP	BUFT, 4, 8, 16	CAPTURE_SPARTAN2
CB2CE, CB4CE, CB8CE, CB16CE	CB2CLE, CB4CLE, CB8CLE, CB16CLE	CB2CLED, CB4CLED, CB8CLED, CB16CLED
CB2RE, CB4RE, CB8RE, CB16RE	CC8CE, CC16CE	CC8CLE, CC16CLE
CC8CLED, CC16CLED	CC8RE, CC16RE	CD4CE
CD4CLE	CD4RE	CD4RLE
CJ4CE, CJ5CE, CJ8CE	CJ4RE, CJ5RE, CJ8RE	CLKDLL, CLKDLLE (Spartan-IIE only)
CLKDLLHF	COMP2, 4, 8, 16	COMPM2, 4, 8, 16
COMPMC8, 16	CR8CE, CR16CE	D2_4E
D3_8E	D4_16E	DEC_CC4, 8, 16
DECODE4, 8, 16	DECODE32, 64	FD
FD_1	FD4CE, FD8CE, FD16CE	FD4RE, FD8RE, FD16RE
FDC	FDC_1	FDCE
FDCE_1	FDCP	FDCP_1
FDCPE	FDCPE_1	FDE
FDE_1	FDP	FDP_1
FDPE	FDPE_1	FDR
FDR_1	FDRE	FDRE_1
FDRS	FDRS_1	FDRSE
FDRSE_1	FDS	FDS_1
FDSE	FDSE_1	FJKC
FJKCE	FJKP	FJKPE
FJKRSE	FJKSRE	FMAP
FTC	FTCE	FTCLE
FTCLEX	FTP	FTPE
FTPLE	FTRSE	FTRSLE
FTSRE	FTSRLE	GND
IBUF, 4, 8, 16	IBUF_selectIO	IBUFG, IBUFG_selectIO
IFD, 4, 8, 16	IFD_1	IFDI
IFDI_1	IFDX, 4, 8, 16	IFDX_1
IFDXI	IFDXI_1	ILD, 4, 8, 16

Spartan-II, Spartan-IIE Design Elements

ILD_1	ILDI	ILDI_1
ILDX, 4, 8, 16	ILDX_1	ILDXI
ILDXI_1	INV, 4, 8, 16	IOBUF, IOBUF_selectIO
IOPAD, 4, 8, 16	IPAD, 4, 8, 16	KEEPER
LD	LD_1	LD4, 8, 16
LDC	LDC_1	LDCE
LDCE_1	LD4CE, LD8CE, LD16CE	LDCP
LDCP_1	LDCPE	LDCPE_1
LDE	LDE_1	LDP
LDP_1	LDPE	LDPE_1
LUT1, 2, 3, 4	LUT1_D, LUT2_D, LUT3_D, LUT4_D	LUT1_L, LUT2_L, LUT3_L, LUT4_L
M2_1	M2_1B1	M2_1B2
M2_1E	M4_1E	M8_1E
M16_1E	MULT_AND	MUXCY
MUXCY_D	MUXCY_L	MUXF5
MUXF5_D	MUXF5_L	MUXF6
MUXF6_D	MUXF6_L	NAND2-9
NAND12, 16	NOR2-9	NOR12, 16
OBUF, 4, 8, 16	OBUF_selectIO	OBUFE, 4, 8, 16
OBUFT, 4, 8, 16	OBUFT_selectIO	OFD, 4, 8, 16
OFD_1	OFDE, 4, 8, 16	OFDE_1
OFDI	OFDI_1	OFDT, 4, 8, 16
OFDT_1	OFDX, 4, 8, 16	OFDX_1
OFDXI	OFDXI_1	OPAD, 4, 8, 16
OR2-9	OR12, 16	PULLDOWN
PULLUP	RAM16X1D	RAM16X1D_1
RAM16X1S	RAM16X1S_1	RAM16X2D
RAM16X2S	RAM16X4D	RAM16X4S
RAM16X8D	RAM16X8S	RAM32X1S
RAM32X1S_1	RAM32X2S	RAM32X4S
RAM32X8S	RAMB4_Sn	RAMB4_Sm_Sn
ROC	ROCBUF	ROM16X1
ROM32X1	SOP3-4	SR4CE, SR8CE, SR16CE
SR4CLE, SR8CLE, SR16CLE	SR4CLED, SR8CLED, SR16CLED	SR4RE, SR8RE, SR16RE
SR4RLE, SR8RLE, SR16RLE	SR4RLED, SR8RLED, SR16RLED	SRL16
SRL16_1	SRL16E	SRL16E_1
STARTBUF_architecture	STARTUP_SPARTAN2	TOC
TOCBUF	UPAD	VCC

XNOR2-9	XOR2-9	XORCY
XORCY_D	XORCY_L	

Spartan-II, Spartan-IIE Constraints

ALLCLOCKNETS	AREA_GROUP	ASYNC_REG
BEL	BLKNM	BOX_TYPE
BUFG (XST)	BUS_DELIMITER	CASE
CLK_FEEDBACK	CLKDV_DIVIDE	CLOCK_BUFFER
CLOCK_SIGNAL	COMPGRP	COMPLEX_CLKEN
CONFIG	CONFIG_MODE	CROSS_CLOCK_ANALYSIS
DECODER_EXTRACT	DISABLE	DOUBLE (Spartan-IIE supported but not Spartan-II)
DRIVE	DROP_SPEC	DUTY_CYCLE
DUTY_CYCLE_CORRECTION	ENABLE	ENUM_ENCODING
EQUIVALENT_REGISTER_REMOVAL	FAST	FILE
FROM-THRU-TO	FROM-TO	FSM_ENCODING
FSM_EXTRACT	FSM_FFTYPE	FULL_CASE
GLOB_OPT	HBLKNM	HIERARCHY_SEPARATOR
HIGH_FREQUENCY	HU_SET	INCREMENTAL_SYNTHESIS
INIT	INIT_xx	INPAD_TO_OUTPAD
IOB	IOBDELAY	IOBUF
IOSTANDARD	IUC	KEEP
KEEP_HIERARCHY	KEEPER	LOC
LOCATE	LOCK	LOCK_PINS
LUT_MAP	MAP	MAX_DELAY
MAXDELAY	MAX_FANOUT)	MAXSKEW
MEDDELAY (Spartan-IIE supported but not Spartan-II)	MOVE_FIRST_STAGE	MOVE_LAST_STAGE
MUX_EXTRACT	MUX_STYLE	NODELAY
OFFSET	OFFSET_IN_BEFORE	OFFSET_OUT_AFTER
ONESHOT	OPT_LEVEL	OPT_MODE
PARALLEL_CASE	PART	PERIOD
PIN (Modular Design Constraint)	PRIORITY	PRIORITY_EXTRACT
PROHIBIT	PULLDOWN	PULLUP
RAM_EXTRACT	RAM_STYLE	READ_CORES
REGISTER_BALANCING	REGISTER_DUPLICATION	RESERVED_SITES
RESOURCE_SHARING	RESYNTHESIZE	RLOC
RLOC_ORIGIN	RLOC_RANGE	ROM_EXTRACT
RTLVIEW	SAVE NET FLAG	SHIFT_EXTRACT
SHREG_EXTRACT	SLEW	SLEW
SLICE_PACKING	SLICE_UTILIZATION_RATIO	SLICE_UTILIZATION_RATIO_MAXMARGIN

Spartan-II, Spartan-IIE Constraints

STARTUP_WAIT	TEMPERATURE	TIG
TIMEGRP	TIMESPEC	TNM
TNM_NET	TPSYNC	TPTHRU
TRANSLATE_OFF and TRANSLATE_ON	TSidentifier	U_SET
UC	USE_RLOC	USELOWSKEWLINES
VERILOG2001	VLGCASE	VLGINCDIR
VLGPATH	VOLTAGE	WRITE_TIMING_CONSTRAINTS
XBLKNM	XILFILE	XOR_COLLAPSE
XSTHDPDIR	XSTHDPINI	

Virtex and Virtex-E

The two tables indicate the supported design elements and constraints for Virtex and Virtex-E. For a complete description, see the Product Data Sheets (http://www.xilinx.com/partinfo/databook.htm#virtex).

Virtex, Virtex-E Design Elements

ACC4, 8, 16	ADD4, 8, 16	ADSU4, 8, 16
AND2-9	AND12, 16	BRLSHFT4, 8
BSCAN_VIRTEX	BUF	BUFCF
BUFE, 4, 8, 16	BUFG	BUFGDLL
BUFGP	BUFT, 4, 8, 16	CAPTURE_VIRTEX
CB2CE, CB4CE, CB8CE, CB16CE	CB2CLE, CB4CLE, CB8CLE, CB16CLE	CB2CLED, CB4CLED, CB8CLED, CB16CLED
CB2RE, CB4RE, CB8RE, CB16RE	CC8CE, CC16CE	CC8CLE, CC16CLE
CC8CLED, CC16CLED	CC8RE, CC16RE	CD4CE
CD4CLE	CD4RE	CD4RLE
CJ4CE, CJ5CE, CJ8CE	CJ4RE, CJ5RE, CJ8RE	CLKDLL
CLKDLLE	CLKDLLHF	COMP2, 4, 8, 16
COMPM2, 4, 8, 16	COMPMC8, 16	CR8CE, CR16CE
D2_4E	D3_8E	D4_16E
DEC_CC4, 8, 16	DECODE4, 8, 16	DECODE32, 64
FD	FD_1	FD4CE, FD8CE, FD16CE
FD4RE, FD8RE, FD16RE	FDC	FDC_1
FDCE	FDCE_1	FDCP
FDCP_1	FDCPE	FDCPE_1
FDE	FDE_1	FDP
FDP_1	FDPE	FDPE_1
FDR	FDR_1	FDRE
FDRE_1	FDRS	FDRS_1
FDRSE	FDRSE_1	FDS
FDS_1	FDSE	FDSE_1
FJKC	FJKCE	FJKP
FJKPE	FJKRSE	FJKSRE
FMAP	FTC	FTCE
FTCLE	FTCLEX	FTP
FTPE	FTPLE	FTRSE
FTRSLE	FTSRE	FTSRLE
GND	IBUF, 4, 8, 16	IBUF_selectIO
IBUFG, IBUFG_selectIO	IFD, 4, 8, 16	IFD_1
IFDI	IFDI_1	IFDX, 4, 8, 16
IFDX_1	IFDXI	IFDXI_1

Virtex, Virtex-E Design Elements

ILD, 4, 8, 16	ILD_1	ILDI
ILDI_1	ILDX, 4, 8, 16	ILDX_1
ILDXI	ILDXI_1	INV, 4, 8, 16
IOBUF, IOBUF_selectIO	IOPAD, 4, 8, 16	IPAD, 4, 8, 16
KEEPER	LD	LD_1
LD4, 8, 16	LDC	LDC_1
LDCE	LDCE_1	LD4CE, LD8CE, LD16CE
LDCP	LDCP_1	LDCPE
LDCPE_1	LDE	LDE_1
LDP	LDP_1	LDPE
LDPE_1	LUT1, 2, 3, 4	LUT1_D, LUT2_D, LUT3_D, LUT4_D
LUT1_L, LUT2_L, LUT3_L, LUT4_L	M2_1	M2_1B1
M2_1B2	M2_1E	M4_1E
M8_1E	M16_1E	MULT_AND
MUXCY	MUXCY_D	MUXCY_L
MUXF5	MUXF5_D	MUXF5_L
MUXF6	MUXF6_D	MUXF6_L
NAND2-9	NAND12, 16	NOR2-9
NOR12, 16	OBUF, 4, 8, 16	OBUF_selectIO
OBUFE, 4, 8, 16	OBUFT, 4, 8, 16	OBUFT_selectIO
OFD, 4, 8, 16	OFD_1	OFDE, 4, 8, 16
OFDE_1	OFDI	OFDI_1
OFDT, 4, 8, 16	OFDT_1	OFDX, 4, 8, 16
OFDX_1	OFDXI	OFDXI_1
OPAD, 4, 8, 16	OR2-9	OR12, 16
PULLDOWN	PULLUP	RAM16X1D
RAM16X1D_1	RAM16X1S	RAM16X1S_1
RAM16X2D	RAM16X2S	RAM16X4D
RAM16X4S	RAM16X8D	RAM16X8S
RAM32X1S	RAM32X1S_1	RAM32X2S
RAM32X4S	RAM32X8S	RAMB4_Sn
RAMB4_Sm_Sn	ROC	ROCBUF
ROM16X1	ROM32X1	SOP3-4
SR4CE, SR8CE, SR16CE	SR4CLE, SR8CLE, SR16CLE	SR4CLED, SR8CLED, SR16CLED
SR4RE, SR8RE, SR16RE	SR4RLE, SR8RLE, SR16RLE	SR4RLED, SR8RLED, SR16RLED
SRL16	SRL16_1	SRL16E
SRL16E_1	STARTBUF_architecture	STARTUP_VIRTEX
TOC	TOCBUF	UPAD

VCC	XNOR2-9	XOR2-9
XORCY	XORCY_D	XORCY_L

Virtex, Virtex-E Constraints

ALLCLOCKNETS	AREA_GROUP	BEL
BLKNM	BOX_TYPE	BUFG (XST)
BUS_DELIMITER	CASE	CLK_FEEDBACK
CLKDV_DIVIDE	CLOCK_BUFFER	CLOCK_SIGNAL
COMPGRP	COMPLEX_CLKEN	CONFIG
CONFIG_MODE	CROSS_CLOCK_ANALYSIS	DECODER_EXTRACT
DISABLE	DRIVE	DROP_SPEC
DUTY_CYCLE	DUTY_CYCLE_CORRECTION	ENABLE
ENUM_ENCODING	EQUIVALENT_REGISTER_REMOVAL	FAST
FILE	FROM-THRU-TO	FROM-TO
FSM_ENCODING	FSM_EXTRACT	FSM_FFTYPE
FULL_CASE	GLOB_OPT	HBLKNM
HIERARCHY_SEPARATOR	HIGH_FREQUENCY	HU_SET (Virtex, but not Virtex-E)
INCREMENTAL_SYNTHESIS	INIT	INIT_xx
INPAD_TO_OUTPAD	IOB	IOBDELAY
IOBUF (Virtex, but not Virtex-E)	IOSTANDARD	IUC
KEEP	KEEP_HIERARCHY	KEEPER
LOC	LOCATE	LOCK
LOCK_PINS	LUT_MAP	MAP
MAX_DELAY	MAXDELAY	MAX_FANOUT
MAXSKEW	MOVE_FIRST_STAGE	MOVE_LAST_STAGE
MUX_EXTRACT	MUX_STYLE	NODELAY
OFFSET	OFFSET_IN_BEFORE	OFFSET_OUT_AFTER
ONESHOT	OPT_LEVEL	OPT_MODE
PARALLEL_CASE	PART	PERIOD
PIN (Modular Design Constraint)	PRIORITY	PRIORITY_EXTRACT
PROHIBIT	PULLDOWN	PULLUP
RAM_EXTRACT	RAM_STYLE	READ_CORES
RTLVIEW	REGISTER_BALANCING	REGISTER_DUPLICATION
RESERVED_SITES	RESOURCE_SHARING	RESYNTHESIZE
RLOC	RLOC_ORIGIN	RLOC_RANGE
ROM_EXTRACT	SAVE NET FLAG	SHIFT_EXTRACT
SHREG_EXTRACT	SLEW	SLEW
SLICE_PACKING	SLICE_UTILIZATION_RATIO	SLICE_UTILIZATION_RATIO_MAXMARGIN
STARTUP_WAIT	TEMPERATURE	TIG
TIMEGRP	TIMESPEC	TNM
TNM_NET	TPSYNC	TPTHRU

TRANSLATE_OFF and TRANSLATE_ON	TSidentifier	U_SET
UC	USE_RLOC	USELOWSKEWLINES
VERILOG2001	VLGCASE	VLGINCDIR
VLGPATH	VOLTAGE	WRITE_TIMING_CONSTRAINTS
XBLKNM	XILFILE	XOR_COLLAPSE
XSTHDPDIR	XSTHDPINI	

Virtex-II and Virtex-II Pro

The two tables indicate the supported design elements and constraints for Virtex-II and Virtex-II Pro. For a complete description of Virtex-II, see the Product Data Sheets (http://www.xilinx.com/partinfo/databook.htm#vtwo).

Virtex-II, Virtex-II Pro Design Elements

ACC4, 8, 16	ADD4, 8, 16	ADSU4, 8, 16
AND2-9	AND12, 16	BRLSHFT4, 8
BSCAN_VIRTEX2	BUF	BUFCF
BUFE, 4, 8, 16	BUFG	BUFGCE
BUFGCE_1	BUFGDLL	BUFGMUX
BUFGMUX_1	BUFGP	BUFT, 4, 8, 16
CAPTURE_VIRTEX2	CB2CE, CB4CE, CB8CE, CB16CE	CB2CLE, CB4CLE, CB8CLE, CB16CLE
CB2CLED, CB4CLED, CB8CLED, CB16CLED	CB2RE, CB4RE, CB8RE, CB16RE	CC8CE, CC16CE
CC8CLE, CC16CLE	CC8CLED, CC16CLED	CC8RE, CC16RE
CD4CE	CD4CLE	CD4RE
CD4RLE	CJ4CE, CJ5CE, CJ8CE	CJ4RE, CJ5RE, CJ8RE
COMP2, 4, 8, 16	COMPM2, 4, 8, 16	COMPMC8, 16
CR8CE, CR16CE	D2_4E	D3_8E
D4_16E	DCM	DEC_CC4, 8, 16
DECODE4, 8, 16	DECODE32, 64	FD
FD_1	FD4CE, FD8CE, FD16CE	FD4RE, FD8RE, FD16RE
FDC	FDC_1	FDCE
FDCE_1	FDCP	FDCP_1
FDCPE	FDCPE_1	FDDRCPE
FDDRRSE	FDE	FDE_1
FDP	FDP_1	FDPE
FDPE_1	FDR	FDR_1
FDRE	FDRE_1	FDRS
FDRS_1	FDRSE	FDRSE_1
FDS	FDS_1	FDSE
FDSE_1	FJKC	FJKCE
FJKP	FJKPE	FJKRSE
FJKSRE	FMAP	FTC
FTCE	FTCLE	FTCLEX
FTP	FTPE	FTPLE
FTRSE	FTRSLE	FTSRE
FTSRLE	GND	GT_AURORA_n (Virtex-II Pro only)
GT_CUSTOM (Virtex-II Pro only)	GT_ETHERNET_n (Virtex-II Pro only)	GT_FIBRE_CHAN_n (Virtex-II Pro only)

Virtex-II, Virtex-II Pro Design Elements

GT_INFINIBAND_n (Virtex-II Pro only)	GT_XAUI_n (Virtex-II Pro only)	IBUF, 4, 8, 16
IBUF_selectIO	IBUFDS	IBUFG, IBUFG_selectIO
IBUFGDS	ICAP_VIRTEX2	IFD, 4, 8, 16
IFD_1	IFDDRCPE	IFDDRRSE
IFDI	IFDI_1	IFDX, 4, 8, 16
IFDX_1	IFDXI	IFDXI_1
ILD, 4, 8, 16	ILD_1	ILDI
ILDI_1	ILDX, 4, 8, 16	ILDX_1
ILDXI	ILDXI_1	INV, 4, 8, 16
IOBUF, IOBUF_selectIO	IOBUFDS	IOPAD, 4, 8, 16
IPAD, 4, 8, 16	JTAGPPC (Virtex-II Pro only)	KEEPER
LD	LD_1	LD4, 8, 16
LDC	LDC_1	LDCE
LDCE_1	LD4CE, LD8CE, LD16CE	LDCP
LDCP_1	LDCPE	LDCPE_1
LDE	LDE_1	LDP
LDP_1	LDPE	LDPE_1
LUT1, 2, 3, 4	LUT1_D, LUT2_D, LUT3_D, LUT4_D	LUT1_L, LUT2_L, LUT3_L, LUT4_L
M2_1	M2_1B1	M2_1B2
M2_1E	M4_1E	M8_1E
M16_1E	MULT_AND	MULT18X18
MULT18X18S	MUXCY	MUXCY_D
MUXCY_L	MUXF5	MUXF5_D
MUXF5_L	MUXF6	MUXF6_D
MUXF6_L	MUXF7	MUXF7_D
MUXF7_L	MUXF8	MUXF8_D
MUXF8_L	NAND2-9	NAND12, 16
NOR2-9	NOR12, 16	OBUF, 4, 8, 16
OBUF_selectIO	OBUFDS	OBUFE, 4, 8, 16
OBUFT, 4, 8, 16	OBUFT_selectIO	OBUFTDS
OFD, 4, 8, 16	OFD_1	OFDDRCPE
OFDDRRSE	OFDDRTCPE	OFDDRTRSE
OFDE, 4, 8, 16	OFDE_1	OFDI
OFDI_1	OFDT, 4, 8, 16	OFDT_1
OFDX, 4, 8, 16	OFDX_1	OFDXI
OFDXI_1	OPAD, 4, 8, 16	OR2-9
OR12, 16	ORCY	PPC405 (Virtex-II Pro only)
PULLDOWN	PULLUP	RAM16X1D

RAM16X1D_1	RAM16X1S	RAM16X1S_1
RAM16X2D	RAM16X2S	RAM16X4D
RAM16X4S	RAM16X8D	RAM16X8S
RAM32X1D	RAM32X1D_1	RAM32X1S
RAM32X1S_1	RAM32X2S	RAM32X4S
RAM32X8S	RAM64X1D	RAM64X1D_1
RAM64X1S	RAM64X1S_1	RAM64X2S
RAM128X1S	RAM128X1S_1	RAMB16_Sn
RAMB16_Sm_Sn	ROC	ROCBUF
ROM16X1	ROM32X1	ROM64X1
ROM128X1	ROM256X1	SOP3-4
SR4CE, SR8CE, SR16CE	SR4CLE, SR8CLE, SR16CLE	SR4CLED, SR8CLED, SR16CLED
SR4RE, SR8RE, SR16RE	SR4RLE, SR8RLE, SR16RLE	SR4RLED, SR8RLED, SR16RLED
SRL16	SRL16_1	SRL16E
SRL16E_1	SRLC16	SRLC16_1
SRLC16E	SRLC16E_1	STARTBUF_architecture
STARTUP_VIRTEX2	TOC	TOCBUF
UPAD	VCC	XNOR2-9
XOR2-9	XORCY	XORCY_D
XORCY_L		

Virtex-II, Virtex-II Pro Constraints

ALLCLOCKNETS	AREA_GROUP	ASYNC_REG
BEL	BLKNM	BOX_TYPE
BUFG (XST)	BUFGCE	BUS_DELIMITER
CASE	CLK_FEEDBACK	CLK_FEEDBACK
CLKDV_DIVIDE	CLKFX_DIVIDE	CLKFX_MULTIPLY
CLKIN_DIVIDE_BY_2	CLKIN_PERIOD	CLKOUT_PHASE_SHIFT
CLOCK_BUFFER	CLOCK_SIGNAL	COMPGRP
COMPLEX_CLKEN	CONFIG	CONFIG_MODE
CROSS_CLOCK_ANALYSIS	DCI_VALUE	DECODER_EXTRACT
DESKEW_ADJUST	DFS_FREQUENCY_MODE	DISABLE
DLL_FREQUENCY_MODE	DRIVE	DROP_SPEC
DSS_MODE	DUTY_CYCLE	DUTY_CYCLE_CORRECTION
ENABLE	ENUM_ENCODING	EQUIVALENT_REGISTER_REMOVAL
FAST	FILE	FROM-THRU-TO
FROM-TO	FSM_ENCODING	FSM_EXTRACT
FSM_FFTYPE	FULL_CASE	GLOB_OPT
HBLKNM	HIERARCHY_SEPARATOR	HIGH_FREQUENCY
HU_SET	INCREMENTAL_SYNTHESIS	INIT
INIT_A	INIT_B	INIT_xx
INITP_xx	INPAD_TO_OUTPAD	IOB
IOBDELAY	IOBUF	IOSTANDARD
IUC	KEEP	KEEP_HIERARCHY
KEEPER	LOC	LOCATE
LOCK	LOCK_PINS	LUT_MAP
MAP	MAX_DELAY	MAXDELAY
MAX_FANOUT	MAXSKEW	MOVE_FIRST_STAGE
MOVE_LAST_STAGE	MULT_STYLE	MUX_EXTRACT
MUX_STYLE	NODELAY	OFFSET
OFFSET_IN_BEFORE	OFFSET_OUT_AFTER	ONESHOT
OPT_LEVEL	OPT_MODE	PARALLEL_CASE
PART	PERIOD	PHASE_SHIFT
PIN (Modular Design Constraint)	PRIORITY	PRIORITY_EXTRACT
PROHIBIT	PULLDOWN	PULLUP
RAM_EXTRACT	RAM_STYLE	READ_CORES
RTLVIEW	REGISTER_BALANCING	REGISTER_DUPLICATION
RESERVED_SITES	RESOURCE_SHARING	RESYNTHESIZE
RLOC	RLOC_ORIGIN	RLOC_RANGE

Virtex-II, Virtex-II Pro Constraints

ROM_EXTRACT	SAVE NET FLAG	SHIFT_EXTRACT
SHREG_EXTRACT	SLEW	SLEW
SLICE_PACKING	SLICE_UTILIZATION_RATIO	SLICE_UTILIZATION_RATIO_MAXMARGIN
SRVAL	SRVAL_A	SRVAL_B
STARTUP_WAIT	TEMPERATURE	TIG
TIMEGRP	TIMESPEC	TNM
TNM_NET	TPSYNC	TPTHRU
TRANSLATE_OFF and TRANSLATE_ON	TSidentifier	UC
U_SET	USE_RLOC	USELOWSKEWLINES
VERILOG2001	VLGCASE	VLGINCDIR
VLGPATH	VOLTAGE	WRITE_MODE
WRITE_MODE_A	WRITE_MODE_B	WRITE_TIMING_CONSTRAINTS
XBLKNM	XILFILE	XOR_COLLAPSE
XSTHDPDIR	XSTHDPINI	

XC9500/XV/XL

The two tables indicate the supported design elements and constraints for XC95000/XV/XL. For a complete description, see the Product Data Sheets (http://www.xilinx.com/partinfo/databook.htm#cpld).

XC9500/XV/XL Design Elements

ACC1	ACC4, 8, 16	ADD1
ADD4, 8, 16	ADSU1	ADSU4, 8, 16
AND2-9	BRLSHFT4, 8	BUF
BUF4, 8, 16	BUFE, 4, 8, 16	BUFG
BUFGSR	BUFGTS	BUFT, 4, 8, 16
CB2CE, CB4CE, CB8CE, CB16CE	CB2CLE, CB4CLE, CB8CLE, CB16CLE	CB2CLED, CB4CLED, CB8CLED, CB16CLED
CB2RE, CB4RE, CB8RE, CB16RE	CB2RLE, CB4RLE, CB8RLE, CB16RLE	CB2X1, CB4X1, CB8X1, CB16X1
CB2X2, CB4X2, CB8X2, CB16X2	CD4CE	CD4CLE
CD4RE	CD4RLE	CJ4CE, CJ5CE, CJ8CE
CJ4RE, CJ5RE, CJ8RE	COMP2, 4, 8, 16	COMPM2, 4, 8, 16
CR8CE, CR16CE	D2_4E	D3_8E
D4_16E	FD	FD4, 8, 16
FD4CE, FD8CE, FD16CE	FD4RE, FD8RE, FD16RE	FDC
FDCE	FDCP	FDCPE
FDP	FDPE	FDR
FDRE	FDRS	FDRSE
FDS	FDSE	FDSR
FDSRE	FJKC	FJKCE
FJKCP	FJKCPE	FJKP
FJKPE	FJKRSE	FJKSRE
FTC	FTCE	FTCLE
FTCP	FTCPE	FTCPLE
FTDCP	FTP	FTPE
FTPLE	FTRSE	FTRSLE
FTSRE	FTSRLE	GND
IBUF, 4, 8, 16	IOBUFE	INV, 4, 8, 16
IOPAD, 4, 8, 16	IPAD, 4, 8, 16	LD
LD4, 8, 16	LDC	LDCP
LDP	M2_1	M2_1B1
M2_1B2	M2_1E	M4_1E
M8_1E	M16_1E	NAND2-9
NOR2-9	OBUF, 4, 8, 16	OBUFE, 4, 8, 16
OBUFT, 4, 8, 16	OPAD, 4, 8, 16	OR2-9
SOP3-4	SR4CE, SR8CE, SR16CE	SR4CLE, SR8CLE, SR16CLE

XC9500/XV/XL Design Elements

SR4CLED, SR8CLED, SR16CLED	SR4RE, SR8RE, SR16RE	SR4RLE, SR8RLE, SR16RLE
SR4RLED, SR8RLED, SR16RLED	VCC	XNOR2-9
XOR2-9		

XC9500/XV/XL Constraints

BOX_TYPE	BUFG (CPLD)	BUS_DELIMITER
CASE	CLK_FEEDBACK	COLLAPSE
COMPLEX_CLKEN	CONFIG	DROP_SPEC
ENUM_ENCODING	EQUIVALENT_REGISTER_REMOVAL	FAST
FILE	FROM-TO	FSM_ENCODING
FSM_EXTRACT	FSM_FFTYPE	FULL_CASE
HBLKNM	HIERARCHY_SEPARATOR	HIGH_FREQUENCY
INIT	IOBUF	IOSTANDARD
IUC	KEEP	KEEP_HIERARCHY
LOC	MAP	MAXPT
MUX_EXTRACT	NOREDUCE	OFFSET
OPT_LEVEL	OPT_MODE	PARALLEL_CASE
PART	PERIOD	PLD_CE
PLD_MP	PLD_XP	PRIORITY
PROHIBIT	PWR_MODE	REG
REGISTER_POWERUP	RESOURCE_SHARING	RTLVIEW
SLEW	SLOW	TIMEGRP
TIMESPEC	TNM	TRANSLATE_OFF and TRANSLATE_ON
TSidentifier	UC	VERILOG2001
VLGCASE	VLGINCDIR	VLGPATH
WIREAND	XILFILE	XSTHDPDIR
XSTHDPINI		

CoolRunner XPLA3

The two tables indicate the supported design elements and constraints for CoolRunner XPLA3. For a complete description, see the Product Data Sheets (http://www.xilinx.com/partinfo/databook.htm#coolrunner).

CoolRunner XPLA3 Design Elements

ACC1	ACC4, 8, 16	ADD1
ADD4, 8, 16	ADSU1	ADSU4, 8, 16
AND2-9	BRLSHFT4, 8	BUF
BUF4, 8, 16	BUFE, 4, 8, 16	BUFG
BUFT, 4, 8, 16	CB2CE, CB4CE, CB8CE, CB16CE	CB2CLE, CB4CLE, CB8CLE, CB16CLE
CB2CLED, CB4CLED, CB8CLED, CB16CLED	CB2RE, CB4RE, CB8RE, CB16RE	CB2RLE, CB4RLE, CB8RLE, CB16RLE
CB2X1, CB4X1, CB8X1, CB16X1	CB2X2, CB4X2, CB8X2, CB16X2	CD4CE
CD4CLE	CD4RE	CD4RLE
CJ4CE, CJ5CE, CJ8CE	CJ4RE, CJ5RE, CJ8RE	COMP2, 4, 8, 16
COMPM2, 4, 8, 16	CR8CE, CR16CE	D2_4E
D3_8E	D4_16E	FD
FD4, 8, 16	FD4CE, FD8CE, FD16CE	FD4RE, FD8RE, FD16RE
FDC	FDCE	FDCP
FDCPE	FDP	FDPE
FDR	FDRE	FDRS
FDRSE	FDS	FDSE
FDSR	FDSRE	FJKC
FJKCE	FJKCP	FJKCPE
FJKP	FJKPE	FJKRSE
FJKSRE	FTC	FTCE
FTCLE	FTCP	FTCPE
FTCPLE	FTDCP	FTP
FTPE	FTPLE	FTRSE
FTRSLE	FTSRE	FTSRLE
GND	IBUF, 4, 8, 16	INV, 4, 8, 16
IOBUFE	IOPAD, 4, 8, 16	IPAD, 4, 8, 16
LD	LD4, 8, 16	LDC
LDCP	LDP	M2_1
M2_1B1	M2_1B2	M2_1E
M4_1E	M8_1E	M16_1E
NAND2-9	NOR2-9	OBUF, 4, 8, 16
OBUFE, 4, 8, 16	OBUFT, 4, 8, 16	OPAD, 4, 8, 16
OR2-9	SOP3-4	SR4CE, SR8CE, SR16CE
SR4CLE, SR8CLE, SR16CLE	SR4CLED, SR8CLED, SR16CLED	SR4RE, SR8RE, SR16RE

CoolRunner XPLA3 Design Elements

SR4RLE, SR8RLE, SR16RLE	SR4RLED, SR8RLED, SR16RLED	VCC
XNOR2-9	XOR2-9	

CoolRunner XPLA3 Constraints

BOX_TYPE	BUFG (CPLD)	BUS_DELIMITER
CASE	CLK_FEEDBACK	COLLAPSE
COMPLEX_CLKEN	CONFIG	DROP_SPEC
ENUM_ENCODING	EQUIVALENT_REGISTER_REMOVAL	FAST
FILE	FROM-TO	FSM_ENCODING
FSM_EXTRACT	FSM_FFTYPE	FULL_CASE
HBLKNM	HIERARCHY_SEPARATOR	HIGH_FREQUENCY
INREG	INIT	IOBUF
IOSTANDARD	IUC	KEEP
KEEP_HIERARCHY	LOC	MAXPT
MUX_EXTRACT	NOREDUCE	OFFSET
OPT_LEVEL	OPT_MODE	PARALLEL_CASE
PART	PERIOD	PLD_CE
PLD_MP	PLD_XP	PRIORITY
PROHIBIT	PULLUP	REG
REGISTER_POWERUP	RESOURCE_SHARING	RTLVIEW
SLEW	SLOW	TIMEGRP
TIMESPEC	TNM	TRANSLATE_OFF and TRANSLATE_ON
TSidentifier	UC	VERILOG2001
VLGCASE	VLGINCDIR	VLGPATH
XILFILE	XSTHDPDIR	XSTHDPINI

CoolRunner-II

The two tables indicate the supported design elements and constraints for CoolRunner-II. For a complete description of CoolRunner-II, see the Product Data Sheets (http:/ / www.xilinx.com / partinfo / databook.htm#cooltwo).

CoolRunner-II Design Elements

ACC1	ACC4, 8, 16	ADD1
ADD4, 8, 16	ADSU1	ADSU4, 8, 16
AND2-9	BRLSHFT4, 8	BUF
BUF4, 8, 16	BUFG	BUFGSR
BUFGTS	CB2CE, CB4CE, CB8CE, CB16CE	CB2CLE, CB4CLE, CB8CLE, CB16CLE
CB2CLED, CB4CLED, CB8CLED, CB16CLED	CB2RE, CB4RE, CB8RE, CB16RE	CB2RLE, CB4RLE, CB8RLE, CB16RLE
CB2X1, CB4X1, CB8X1, CB16X1	CB2X2, CB4X2, CB8X2, CB16X2	CBD2CE, CBD4CE, CBD8CE, CBD16CE
CBD2CLE, CBD4CLE, CBD8CLE, CBD16CLE	CBD2CLED, CBD4CLED, CBD8CLED, CBD16CLED	CBD2RE, CBD4RE, CBD8RE, CBD16RE
CBD2RLE, CBD4RLE, CBD8RLE, CBD16RLE	CBD2X1, CBD4X1, CBD8X1, CBD16X1	CBD2X2, CBD4X2, CBD8X2, CBD16X2
CD4CE	CD4CLE	CD4RE
CD4RLE	CDD4CE	CDD4CLE
CDD4RE	CDD4RLE	CJ4CE, CJ5CE, CJ8CE
CJ4RE, CJ5RE, CJ8RE	CJD4CE, CJD5CE, CJD8CE	CJD4RE, CJD5RE, CJD8RE
CLK_DIV2,4,6,8,10,12,14,16	CLK_DIV2,4,6,8,10,12,14,16R	CLK_DIV2,4,6,8,10,12,14,16RSD
CLK_DIV2,4,6,8,10,12,14,16SD	COMP2, 4, 8, 16	COMPM2, 4, 8, 16
CR8CE, CR16CE	CRD8CE, CRD16CE	D2_4E
D3_8E	D4_16E	FD
FD4, 8, 16	FD4CE, FD8CE, FD16CE	FD4RE, FD8RE, FD16RE
FDC	FDCE	FDCP
FDCPE	FDD	FDD4,8,16
FDD4CE, FDD8CE, FDD16CE	FDD4RE, FDD8RE, FDD16RE	FDDC
FDDCE	FDDCP	FDDCPE
FDDP	FDDPE	FDDR
FDDRE	FDDRS	FDDRSE
FDDS	FDDSE	FDDSR
FDDSRE	FDP	FDPE
FDR	FDRE	FDRS
FDRSE	FDS	FDSE
FDSR	FDSRE	FJKC
FJKCE	FJKCP	FJKCPE
FJKP	FJKPE	FJKRSE

CoolRunner-II Design Elements

FJKSRE	FTC	FTCE
FTCLE	FTCP	FTCPE
FTCPLE	FTDCE	FTDCLE
FTDCLEX	FTDCP	FTDRSE
FTDRSLE	FTP	FTPE
FTPLE	FTRSE	FTRSLE
FTSRE	FTSRLE	GND
IBUF, 4, 8, 16	INV, 4, 8, 16	IOBUFE
IOPAD, 4, 8, 16	IPAD, 4, 8, 16	KEEPER
LD	LD4, 8, 16	LDC
LDCP	LDG	LDG4, 8, 16
LDP	M2_1	M2_1B1
M2_1B2	M2_1E	M4_1E
M8_1E	M16_1E	NAND2-9
NOR2-9	OBUF, 4, 8, 16	OBUFE, 4, 8, 16
OBUFT, 4, 8, 16	OPAD, 4, 8, 16	OR2-9
PULLDOWN	PULLUP	SOP3-4
SR4CE, SR8CE, SR16CE	SR4CLE, SR8CLE, SR16CLE	SR4CLED, SR8CLED, SR16CLED
SR4RE, SR8RE, SR16RE	SR4RLE, SR8RLE, SR16RLE	SR4RLED, SR8RLED, SR16RLED
SRD4CE, SRD8CE, SRD16CE	SRD4CLE, SRD8CLE, SRD16CLE	SRD4CLED, SRD8CLED, SRD16CLED
SRD4RE, SRD8RE, SRD16RE	SRD4RLE, SRD8RLE, SRD16RLE	SRD4RLED, SRD8RLED, SRD16RLED
VCC	XNOR2-9	XOR2-9

CoolRunner-II Constraints

BUFG (CPLD)	BUS_DELIMITER	CASE
CLK_FEEDBACK	COLLAPSE	COMPLEX_CLKEN
CONFIG	COOL_CLK	DATA_GATE
DROP_SPEC	ENUM_ENCODING	EQUIVALENT_REGISTER_REMOVAL
FAST	FILE	FROM-TO
FSM_ENCODING	FSM_EXTRACT	FSM_FFTYPE
FULL_CASE	HBLKNM	HIERARCHY_SEPARATOR
HIGH_FREQUENCY	INREG	INIT
IOBUF	IOSTANDARD	IUC
KEEP	KEEP_HIERARCHY	KEEPER
LOC	MAXPT	MUX_EXTRACT
NOREDUCE	OFFSET	OPEN_DRAIN
OPT_LEVEL	OPT_MODE	PARALLEL_CASE
PART	PERIOD	PLD_CE
PLD_MP	PLD_XP	PRIORITY
PROHIBIT	PULLUP	REG
REGISTER_POWERUP	RESOURCE_SHARING	RTLVIEW
SCHMITT_TRIGGER	SLEW	SLOW
TIMEGRP	TIMESPEC	TNM
TRANSLATE_OFF and TRANSLATE_ON	TSidentifier	UC
VERILOG2001	VLGCASE	VLGINCDIR
VLGPATH	VREF	XILFILE
XSTHDPDIR	XSTHDPINI	

Functional Categories

This section categorizes, by function, the logic elements that are described in detail in the “Design Elements” sections. Each category is briefly described. Tables under each category identify all the available elements for the function and indicate which libraries include the element.

Elements are listed in alphanumeric order under each category.

Refer to the “Applicable Architectures” section of the “Xilinx Unified Libraries” chapter for information on the specific device families that use each library. "N/A" column means that the element does not apply.

"RPM" refers to Relationally Placed Macros. RPMs are “soft” macros that contain relative location constraint (RLOC) information. The Xilinx libraries contain three types of elements.

- Primitives are basic logical elements such as AND2 and OR2 gates.
- Soft macros are schematics made by combining primitives and sometimes other soft macros.
- Relationally placed macros (RPMs) are soft macros that contain relative location constraint (RLOC) information, carry logic symbols, and FMAP symbols, where appropriate.

The last item mentioned above, RPMs, applies only to FPGA families.

The relationally placed macro (RPM) library uses RLOC constraints to define the order and structure of the underlying design primitives. Because these macros are built upon standard schematic parts, they do not have to be translated before simulation. The components that are implemented as RPMs are listed in the “Slice Count” section.

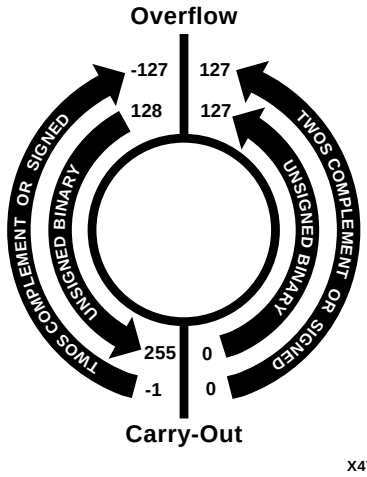
Designs created with RPMs can be functionally simulated. RPMs can, but need not, include all the following elements.

- FMAPs and CLB-grouping attributes to control mapping. FMAPs have pin-lock attributes, which allow better control over routing.
- Relative location (RLOC) constraints to provide placement structure. They allow positioning of elements relative to each other.
- Carry logic primitive symbols.

The RPM library offers the functionality and precision of the hard macro library with added flexibility. You can optimize RPMs and merge other logic within them. The elements in the RPM library allow you to access carry logic easily and to control mapping and block placement. Because RPMs are a superset of ordinary macros, you can design them in the normal design entry environment. They can include any primitive logic. The macro logic is fully visible to you and can be easily back-annotated with timing information.

Arithmetic Functions

There are three types of arithmetic functions: accumulators (ACC), adders (ADD), and adder / subtracters (ADSU). With an ADSU, either unsigned binary or twos-complement operations cause an overflow. If the result crosses the overflow boundary, an overflow is generated. Similarly, when the result crosses the carry-out boundary, a carry-out is generated. The following figure shows the ADSU carry-out and overflow boundaries.



ADSU Carry-Out and Overflow Boundaries

ACC1		1-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Macro	Macro	Macro

ACC4, 8, 16		4-, 8-, 16-Bit Loadable Cascadable Accumulators with Carry-In, Carry-Out, and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

ADD1		1-Bit Full Adder with Carry-In and Carry-Out			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Macro	Macro	Macro

ADD4, 8, 16		4-, 8-, 16-Bit Cascadable Full Adders with Carry-In, Carry-Out, and Overflow			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

ADSU1		1-Bit Cascadable Adder/Subtracter with Carry-In, Carry-Out			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	Macro	Macro	Macro

ADSU4, 8, 16		4-, 8-, 16-Bit Cascadable Adders/Subtracters with Carry-In, Carry-Out, and Overflow			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	N/A	Macro	Macro	Macro	Macro

MULT18X18		18 x 18 Signed Multiplier			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

MULT18X18S		18 x 18 Signed Multiplier			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

Buffers

The buffers in this section route high fanout signals, 3-state signals, and clocks inside a PLD device. The "Input/Output Functions" section covers off-chip interfaces.

BUF		General-Purpose Buffer			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

BUF4, 8, 16		4-, 8-, 16-Bit General-Purpose Buffers			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	Macro	Macro	Macro

BUFCF		Fast Connect Buffer			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

BUFE, 4, 8, 16		Internal 3-State Buffers with Active High Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500***	**CoolRunner XPLA3**	**CoolRunner-II**
BUFE -- Primitive	Primitive	Primitive	Primitive	Primitive	N/A
BUFE4, 8, 16 -- Macro	Macro	Macro	Macro	Macro	N/A

* not supported for XC9500XL and XC9500XV devices

BUFG		Global Clock Buffer			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

BUFGCE		Global Clock MUX with Clock Enable and Output State 0			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Macro	N/A	N/A	N/A

BUFGCE_1		Global Clock MUX Buffer with Clock Enable and Output State 1			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Macro	N/A	N/A	N/A

BUFGDLL		Clock Delay Locked Loop Buffer			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

BUFGMUX		Global Clock MUX Buffer with Output State 0			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

BUFGMUX_1		Global Clock MUX with Output State 1			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

BUFGP		Primary Global Buffer for Driving Clocks or Longlines (Four per PLD Device)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

BUFGSR		Global Set / Reset Input Buffer			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Primitive	Primitive	Primitive

BUFGTS		Global 3-State Input Buffer			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Primitive	Primitive	Primitive

BUFT, 4, 8, 16		Internal 3-State Buffers with Active-Low Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500**	**CoolRunner XPLA3**	**CoolRunner-II**
BUFT-- Primitive	Primitive	Primitive	Primitive*	Primitive	N / A
BUFT4, 8, 16 -- Macro	Macro	Macro	Macro*	Macro	N / A

*not supported for XC9500XL and XC9500XV devices

Comparators

Following is a list of comparators.

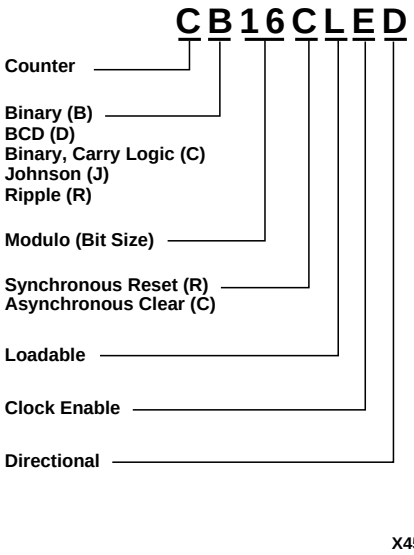
COMP2, 4, 8, 16		2-, 4-, 8-, 16-Bit Identity Comparators			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

COMPM2, 4, 8, 16		2-, 4-, 8-, 16-Bit Magnitude Comparators			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

COMPMC8, 16		8-, 16-Bit Magnitude Comparators			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

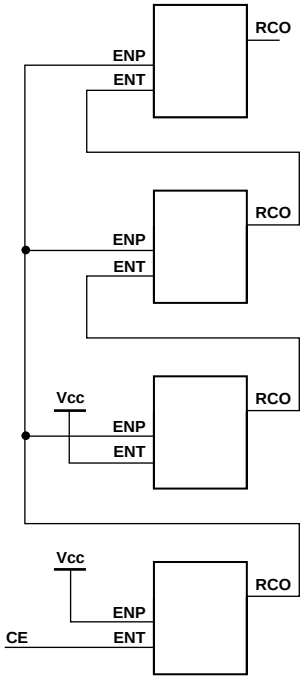
Counters

There are six types of counters with various synchronous and asynchronous inputs. The name of the counter defines the modulo or bit size, the counter type, and which control functions are included. The counter naming convention is shown in the following figure.



Counter Naming Convention

A carry-lookahead design accommodates large counters without extra gating. On TTL 7400-type counters with trickle clock enable (ENT), parallel clock enable (ENP), and ripple carry-out (RCO), both the ENT and ENP inputs must be High to count. ENT is propagated forward to enable RCO, which produces a High output with the approximate duration of the QA output. The following figure illustrates a carry-lookahead design.



Carry-Lookahead Design

The RCO output of the first stage of the ripple carry is connected to the ENP input of the second stage and all subsequent stages. The RCO output of the second stage and all subsequent stages is connected to the ENT input of the next stage. The ENT of the second stage is always enabled / tied to VCC. CE is always connected to the ENT input of the first stage. This cascading method allows the first stage of the ripple carry to be built as a prescaler. In other words, the first stage is built to count very fast.

CB2CE, CB4CE, CB8CE, CB16CE		2-, 4-, 8-, 16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CB2CLE, CB4CLE, CB8CLE, CB16CLE		2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CB2CLED, CB4CLED, CB8CLED, CB16CLED		2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CB2RE, CB4RE, CB8RE, CB16RE		2-, 4-, 8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CB2RLE, CB4RLE, CB8RLE, CB16RLE		2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	Macro	Macro	Macro

CB2X1, CB4X1, CB8X1, CB16X1		2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	Macro	Macro	Macro

CB2X2, CB4X2, CB8X2, CB16X2		2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	Macro	Macro	Macro

CBD2CE, CBD4CE, CBD8CE, CBD16CE		2-, 4-, 8-,16-Bit Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

CBD2CLE, CBD4CLE, CBD8CLE, CBD16CLE		2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

CBD2CLED, CBD4CLED, CBD8CLED, CBD16CLED		2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

CBD2RE, CBD4RE, CBD8RE, CBD16RE		2-, 4-, 8-, 16-Bit Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

CBD2RLE, CBD4RLE, CBD8RLE, CBD16RLE		2-, 4-, 8-, 16-Bit Loadable Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

CBD2X1, CBD4X1, CBD8X1, CBD16X1		2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

CBD2X2, CBD4X2, CBD8X2, CBD16X2		2-, 4-, 8-, and 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

CC8CE, CC16CE		8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

CC8CLE, CC16CLE		8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

CC8CLED, CC16CLED		8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

CC8RE, CC16RE		8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

CD4CE		4-Bit Cascadable BCD Counter with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CD4CLE		4-Bit Loadable Cascadable BCD Counter with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CD4RE		4-Bit Cascadable BCD Counter with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CD4RLE		4-Bit Loadable Cascadable BCD Counter with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CDD4CE		4-Bit Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

CDD4CLE		4-Bit Loadable Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

CDD4RE		4-Bit Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

CDD4RLE		4-Bit Loadable Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

CJ4CE, CJ5CE, CJ8CE		4-, 5-, 8-Bit Johnson Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CJ4RE, CJ5RE, CJ8RE		4-, 5-, 8-Bit Johnson Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CJD4CE, CJD5CE, CJD8CE		4-, 5-, 8-Bit Dual Edge Triggered Johnson Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

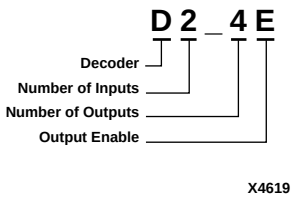
CJD4RE, CJD5RE, CJD8RE		4-, 5-, 8-Bit Dual Edge Triggered Johnson Counters with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

CR8CE, CR16CE		8-, 16-Bit Negative-Edge Binary Ripple Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

CRD8CE, CRD16CE		8-, 16-Bit Dual-Edge Triggered Binary Ripple Counters with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

Decoders

Decoder names, shown in the following figure, indicate the number of inputs and outputs and whether or not an enable is available. Decoders with an enable can be used as multiplexers.



Decoder Naming Convention

D2_4E		2- to 4-Line Decoder / Demultiplexer with Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

D3_8E		3- to 8-Line Decoder / Demultiplexer with Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

D4_16E		4- to 16-Line Decoder / Demultiplexer with Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

DEC_CC4, 8, 16		4-, 8-, 16-Bit Active Low Decoders			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

Edge Decoders

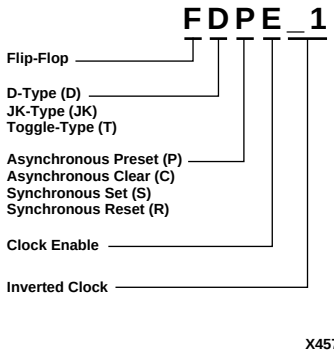
Edge decoders are open-drain wired AND gates that are available in different bit sizes.

DECODE4, 8, 16		4-, 8-, 16-Bit Active-Low Decoders			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

DECODE32, 64		32- and 64-Bit Active-Low Decoders			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

Flip-Flops

There are three types of flip-flops (D, J-K, toggle) with various synchronous and asynchronous inputs. Some are available with inverted clock inputs and / or the ability to set in response to global set / reset rather than reset. The naming convention shown in the following figure provides a description for each flip-flop. D-type flip-flops are available in multiples of up to 16 in one macro.



Flip-Flop Naming Convention

FD		D Flip-Flop			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FD_1		D Flip-Flop with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

FD4, 8, 16		Multiple D Flip-Flops			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Macro	Macro	Macro

FD4CE, FD8CE, FD16CE		4-, 8-, 16-Bit Data Registers with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FD4RE, FD8RE, FD16RE		4-, 8-, 16-Bit Data Registers with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FDC		D Flip-Flop with Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FDC_1		D Flip-Flop with Negative-Edge Clock and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

FDCE		D Flip-Flop with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

FDCE_1		D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

FDCP		D Flip-Flop with Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

FDCP_1		D Flip-Flop with Negative-Edge Clock and Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

FDCPE		D Flip-Flop with Clock Enable and Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Primitive

FDCPE_1		D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

FDD		Dual Edge Triggered D Flip-Flop			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDD4,8,16		Multiple Dual Edge Triggered D Flip-Flops			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDD4CE, FDD8CE, FDD16CE		4-, 8-, 16-Bit Dual Edge Triggered Data Registers with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDD4RE, FDD8RE, FDD16RE		4-, 8-, 16-Bit Dual Edge Triggered Data Registers with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDC		D Dual Edge Triggered Flip-Flop with Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDCE		Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

FDDCP		Dual Edge Triggered D Flip-Flop Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

FDDCPE		Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

FDDP		Dual Edge Triggered D Flip-Flop with Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

FDDPE		Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

FDDR		Dual Edge Triggered D Flip-Flop with Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDRCPE		Dual Data Rate D Flip-Flop with Clock Enable and Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

FDDRE		Dual Edge Triggered D Flip-Flop with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDRRSE		Dual Data Rate D Flip-Flop with Clock Enable and Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

FDDRS		Dual Edge Triggered D Flip-Flop with Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDRSE		Dual Edge Triggered D Flip-Flop with Synchronous Reset and Set and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDS		Dual Edge Triggered D Flip-Flop with Synchronous Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDSE		D Flip-Flop with Clock Enable and Synchronous Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDSR		Dual Edge Triggered D Flip-Flop with Synchronous Set and Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDDSRE		Dual Edge Triggered D Flip-Flop with Synchronous Set and Reset and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FDE		D Flip-Flop with Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

FDE_1		D Flip-Flop with Negative-Edge Clock and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

FDP		D Flip-Flop with Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FDP_1		D Flip-Flop with Negative-Edge Clock and Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

FDPE		D Flip-Flop with Clock Enable and Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

FDPE_1		D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

FDR		D Flip-Flop with Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FDR_1		D Flip-Flop with Negative-Edge Clock and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

FDRE		D Flip-Flop with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FDRE_1		D Flip-Flop with Negative-Clock Edge, Clock Enable, and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

FDRS		D Flip-Flop with Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FDRS_1		D Flip-Flop with Negative-Clock Edge and Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

FDRSE		D Flip-Flop with Synchronous Reset and Set and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FDRSE_1		D Flip-Flop with Negative-Clock Edge, Synchronous Reset and Set, and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

FDS		D Flip-Flop with Synchronous Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FDS_1		D Flip-Flop with Negative-Edge Clock and Synchronous Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

FDSE		D Flip-Flop with Clock Enable and Synchronous Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Macro	Macro

FDSE_1		D Flip-Flop with Negative-Edge Clock, Clock Enable, and Synchronous Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

FDSR		D Flip-Flop with Synchronous Set and Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Macro	Macro	Macro

FDSRE		D Flip-Flop with Synchronous Set and Reset and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Macro	Macro	Macro

FJKC		J-K Flip-Flop with Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FJKCE		J-K Flip-Flop with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FJKCP		J-K Flip-Flop with Asynchronous Clear and Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Macro	Macro	Macro

FJKCPE		J-K Flip-Flop with Asynchronous Clear and Preset and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	Macro	Macro	Macro

FJKP		J-K Flip-Flop with Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FJKPE		J-K Flip-Flop with Clock Enable and Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FJKRSE		J-K Flip-Flop with Clock Enable and Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FJKSRE		J-K Flip-Flop with Clock Enable and Synchronous Set and Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTC		Toggle Flip-Flop with Toggle Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTCE		Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTCLE		Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTCLEX		Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

FTCP		Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	Primitive	Primitive	Primitive

FTCPE		Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	Macro	Macro	Macro

FTCPLE		Loadable Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	Macro	Macro	Macro

FTDCE		Dual Edge Triggered Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

FTDCLE		Dual Edge Triggered Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

FTDCLEX		Dual Edge Triggered Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

FTDCP		Dual Edge Triggered Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

FTDRSLE		Dual Edge Triggered Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	N / A	N / A	N / A	Macro

FTP		Toggle Flip-Flop with Toggle Enable and Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTPE		Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTPLE		Toggle / Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTRSE		Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTRSLE		Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTSRE		Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

FTSRLE		Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

General

General elements include FPGA configuration functions, oscillators, boundary scan logic, and other functions not classified in other sections.

BSCAN_SPARTAN2		Spartan2 Boundary Scan Logic Control Circuit			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	N/A	N/A	N/A	N/A	N/A

BSCAN_VIRTEX		Virtex Boundary Scan Logic Control Circuit			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	Primitive	N/A	N/A	N/A	N/A

BSCAN_VIRTEX2		Virtex2 Boundary Scan Logic Control Circuit			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

CAPTURE_SPARTAN2		Spartan-II Register State Capture for Bitstream Readback			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	N/A	N/A	N/A	N/A	N/A

CAPTURE_VIRTEX		Virtex Register State Capture for Bitstream Readback			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	Primitive	N/A	N/A	N/A	N/A

CAPTURE_VIRTEX2		Virtex-II Register State Capture for Bitstream Readback			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

CLK_DIV2,4,6,8,10,12,14,16		Global Clock Divider			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

CLK_DIV2,4,6,8,10,12,14,16R		Global Clock Divider with Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

CLK_DIV2,4,6,8,10,12,14,16RSD		Global Clock Divider with Synchronous Reset and Start Delay			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

CLK_DIV2,4,6,8,10,12,14,16SD		Global Clock Divider with Start Delay			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Primitive

CLKDLL		Clock Delay Locked Loop			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive*	N/A	N/A	N/A	N/A

* Use CLKDLLE for Virtex-E Devices

CLKDLLE		Clock Delay Locked Loop with Expanded Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	Primitive*	N/A	N/A	N/A	N/A

* Use CLKDLLE for Virtex-E, Spartan-IIE Devices

CLKDLLHF		High Frequency Clock Delay Locked Loop			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive*	N/A	N/A	N/A	N/A

*Use CLKDLLE for Virtex-E devices

DCM		Digital Clock Manager			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

GND		Ground-Connection Signal Tag			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

ICAP_VIRTEX2		User Interface to Virtex2 Internal Configuration Access Port			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

JTAGPPC		JTAG Primitive for the Power PC			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive*	N/A	N/A	N/A

* Not supported for Virtex-II. Only supported for Virtex-II Pro

KEEPER		KEEPER Symbol			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	Primitive

LUT1, 2, 3, 4		1-, 2-, 3-, 4-Bit Look-Up-Table with General Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LUT1_D, LUT2_D, LUT3_D, LUT4_D		1-, 2-, 3-, 4-Bit Look-Up-Table with Dual Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LUT1_L, LUT2_L, LUT3_L, LUT4_L		1-, 2-, 3-, 4-Bit Look-Up-Table with Local Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

PPC405		Primitive for Power PC Core			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro***	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

* Not supported for Virtex-II. Only for Virtex-II Pro.

PULLDOWN		Resistor to GND for Input Pads			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	Primitive

PULLUP		Resistor to VCC for Input PADs, Open-Drain, and 3-State Outputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	Primitive

STARTUP_SPARTAN2		Spartan2 User Interface to Global Clock, Reset, and 3-State Controls			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	N / A	N / A	N / A	N / A	N / A

STARTUP_VIRTEX		Virtex User Interface to Global Clock, Reset, and 3-State Controls			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	Primitive	N / A	N / A	N / A	N / A

STARTUP_VIRTEX2		Virtex2 User Interface to Global Clock, Reset, and 3-State Controls			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

VCC		VCC-Connection Signal Tag			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

Input Latches

Single and multiple input latches can hold transient data entering a chip. Input latches use the same naming convention as I/O flip-flops.

ILD, 4, 8, 16		Transparent Input Data Latches			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

ILD_1		Transparent Input Data Latch with Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

ILDI		Transparent Input Data Latch (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

ILDI_1		Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

ILDX, 4, 8, 16		Transparent Input Data Latches			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

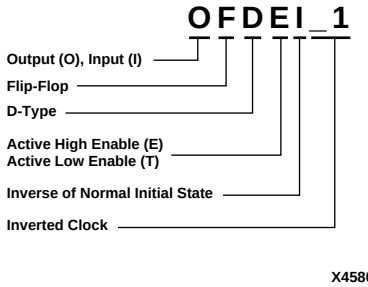
ILDX_1		Transparent Input Data Latch with Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

ILDXI		Transparent Input Data Latch (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

ILDXI_1		Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

Input/Output Flip-Flops

Input/Output flip-flops are configured in IOBs. They include flip-flops whose outputs are enabled by 3-state buffers, flip-flops that can be set upon global set/reset rather than reset, and flip-flops with inverted clock inputs. The naming convention specifies each flip-flop function and is illustrated in the following figure.



Input/Output Flip-Flop Naming Convention

IFD, 4, 8, 16		Single- and Multiple-Input D Flip-Flop			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

IFD_1		Input D Flip-Flop with Inverted Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

IFDDRCPE		Dual Data Rate Input D Flip-Flop with Clock Enable and Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Macro	N/A	N/A	N/A

IFDDRRSE		Dual Data Rate Input D Flip-Flop with Synchronous Reset and Set and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Macro	N/A	N/A	N/A

IFDI		Input D Flip-Flop (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

IFDI_1		Input D Flip-Flop with Inverted Clock (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

IFDX, 4, 8, 16		Single- and Multiple-Input D Flip-Flops with Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

IFDX_1		Input D Flip-Flop with Inverted Clock and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

IFDXI		Input D Flip-Flop with Clock Enable (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

IFDXI_1		Input D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

OFD, 4, 8, 16		Single- and Multiple-Output D Flip-Flops			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

OFD_1		Output D Flip-Flop with Inverted Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

OFDDRCPE		Dual Data Rate Output D Flip-Flop with Clock Enable and Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Macro	N/A	N/A	N/A

OFDDRRSE		Dual Data Rate Output D Flip-Flop with Synchronous Reset and Set and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Macro	N / A	N / A	N / A

OFDDRTCPE		Dual Data Rate D Flip-Flop with Active-Low 3--State Output Buffer, Clock Enable, and Asynchronous Preset and Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Macro	N / A	N / A	N / A

OFDDRTRSE		Dual Data Rate D Flip-Flop with Active -Low 3-State Output Buffer, Synchronous Reset and Set, and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Macro	N / A	N / A	N / A

OFDE, 4, 8, 16		D Flip-Flops with Active-High Enable Output Buffers			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

OFDE_1		D Flip-Flop with Active-High Enable Output Buffer and Inverted Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

OFDI		Output D Flip-Flop (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

OFDI_1		Output D Flip-Flop with Inverted Clock (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

OFDT, 4, 8, 16		Single and Multiple D Flip-Flops with Active-Low 3-State Output Buffers			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

OFDT_1		D Flip-Flop with Active-Low 3-State Output Buffer and Inverted Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

OFDX, 4, 8, 16		Single- and Multiple-Output D Flip-Flops with Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

OFDX_1		Output D Flip-Flop with Inverted Clock and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

OFDXI		Output D Flip-Flop with Clock Enable (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

OFDXI_1		Output D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

Input/Output Functions

Input/Output Block (IOB) resources are configured into various I/O primitives and macros for convenience, such as output buffers (OBUFs) and output buffers with an enable (OBUFEs). Pads used to connect the circuit to PLD device pins are also included.

Virtex, Virtex-E, Spartan-II, Spartan-IIE, Virtex-II, and Virtex-II Pro have multiple variants (primitives) to choose from for each selectI/O buffer. The I/O interface for each variant corresponds to a specific I/O standard.

GT_AURORA_n		Gigabit Transceiver for High-Speed I/O			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro***	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

*Not supported for Virtex-II. Only supported for Virtex-II Pro

GT_CUSTOM		Gigabit Transceiver for High-Speed I/O			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro***	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

*Not supported for Virtex-II. Only supported for Virtex-II Pro

GT_ETHERNET_n		Gigabit Transceiver for High-Speed I/O			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro***	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

*Not supported for Virtex-II. Only supported for Virtex-II Pro

GT_FIBRE_CHAN_n		Gigabit Transceiver for High-Speed I/O			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro***	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

*Not supported for Virtex-II. Only supported for Virtex-II Pro

GT_INFINIBAND_n		Gigabit Transceiver for High-Speed I/O			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro***	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

*Not supported for Virtex-II. Only supported for Virtex-II Pro

GT_XAUI_n		10-Gigabit Transceiver for High-Speed I/O			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro***	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

*Not supported for Virtex-II. Only supported for Virtex-II Pro

IBUF, 4, 8, 16		Single- and Multiple-Input Buffers			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
IBUF					
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IBUF4, 8, 16					
Macro	Macro	Macro	Macro	Macro	Macro

IBUF_selectIO		Single Input Buffer with Selectable I/O Interface (multiple primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

IBUFDS		Differential Signaling Input Buffer with Selectable I/O Interface			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

IBUFG, IBUFG_selectIO		Dedicated Input Buffer with Selectable I/O Interface (multiple primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

IBUFGDS		Dedicated Differential Signaling Input Buffer with Selectable I/O Interface			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

IOBUF, IOBUF_selectIO		Bi-Directional Buffer with Selectable I/O Interface (multiple primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

IOPAD, 4, 8, 16		Single- and Multiple-Input/Output Pads			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
IOPAD					
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IOPAD4, 8, 16					
Macro	Macro	Macro	Macro	Macro	Macro

IPAD, 4, 8, 16		Single- and Multiple-Input Pads			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
IPAD -- Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IPAD4, 8, 16 -- Macro	Macro	Macro	Macro	Macro	Macro

OBUF, 4, 8, 16		Single- and Multiple-Output Buffers			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
OBUF -- Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OBUF4, 8, 16 -- Macro	Macro	Macro	Macro	Macro	Macro

OBUF_selectIO		Single Output Buffer with Selectable I/O Interface (multiple primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

OBUFDS		Differential Signaling Output Buffer with Selectable I/O Interface			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

OBUFE, 4, 8, 16		3-State Output Buffers with Active-High Output Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
OBUFE -- Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUFE4, 8, 16 -- Macro	Macro	Macro	Macro	Macro	Macro

OBUFT, 4, 8, 16		Single and Multiple 3-State Output Buffers with Active Low Output Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
OBUFT -- Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OBUFT4, 8, 16 -- Macro	Macro	Macro	Macro	Macro	Macro

OBUFT_selectIO		Single 3-State Output Buffer with Active-Low Output Enable and Selectable I/O Interface (multiple primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

OBUFTDS		3-State Output Buffer with Differential Signaling, Active-Low Output Enable, and Selectable I/O Interface			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

OPAD, 4, 8, 16		Single- and Multiple-Output Pads			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
OPAD -- Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OPAD4, 8, 16 -- Macro	Macro	Macro	Macro	Macro	Macro

UPAD		Connects the I/O Node of an IOB to the Internal PLD Circuit			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

Latches

Latches (LD) are available for all architectures.

LD		Transparent Data Latch			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Primitive	Primitive

LD_1		Transparent Data Latch with Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LD4, 8, 16		Multiple Transparent Data Latches			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

LDC		Transparent Data Latch with Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Primitive	Primitive

LDC_1		Transparent Data Latch with Asynchronous Clear and Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDCE		Transparent Data Latch with Asynchronous Clear and Gate Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDCE_1		Transparent Data Latch with Asynchronous Clear, Gate Enable, and Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LD4CE, LD8CE, LD16CE		Transparent Data Latches with Asynchronous Clear and Gate Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

LDCP		Transparent Data Latch with Asynchronous Clear and Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Primitive	Primitive

LDCP_1		Transparent Data Latch with Asynchronous Clear and Preset and Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDCPE		Transparent Data Latch with Asynchronous Clear and Preset and Gate Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDCPE_1		Transparent Data Latch with Asynchronous Clear and Preset, Gate Enable, and Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDE		Transparent Data Latch with Gate Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDE_1		Transparent Data Latch with Gate Enable and Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDP		Transparent Data Latch with Asynchronous Preset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	Macro	Primitive	Primitive

LDP_1		Transparent Data Latch with Asynchronous Preset and Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDPE		Transparent Data Latch with Asynchronous Preset and Gate Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

LDPE_1		Transparent Data Latch with Asynchronous Preset, Gate Enable, and Inverted Gate			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

Logic Primitives

Combinatorial logic gates that implement the basic Boolean functions are available in all architectures with up to five inputs in all combinations of inverted and non-inverted inputs, and with six to nine inputs non-inverted.

AND2-9		2- to 9-Input AND Gates with Inverted and Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
AND2 through 5					
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND6 through 9					
Macro	Macro	Macro	Primitive	Primitive	Primitive

AND12, 16		12- and 16-Input AND Gates with Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

INV, 4, 8, 16		Single and Multiple Inverters			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
INV -- Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
INV4, 8, 16 -- Macro	Macro	Macro	Macro	Macro	Macro

MULT_AND		Fast Multiplier AND			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

MULT18X18		18 x 18 Signed Multiplier			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

MULT18X18S		18 x 18 Signed Multiplier -- Registered Version			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

NAND2-9		2- to 9-Input NAND Gates with Inverted and Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
NAND2 through 5					

Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND6 through 9					
Macro	Macro	Macro	Primitive	Primitive	Primitive

NAND12, 16		12- and 16-Input NAND Gates with Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

NOR2-9		2- to 9-Input NOR Gates with Inverted and Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
NOR2 through 5					
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR6 through 9					
Macro	Macro	Macro	Primitive	Primitive	Primitive

NOR12, 16		12 and 16-Input NOR Gates with Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

OR2-9		2- to 9-Input OR Gates with Inverted and Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
OR2 through 5					
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR6 through 9					
Macro	Macro	Macro	Primitive	Primitive	Primitive

OR12, 16		12- and 16-Input OR Gates with Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N/A	N/A	N/A

ORCY		OR with Carry Logic			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

SOP3-4		Sum of Products			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

XNOR2-9		2- to 9-Input XNOR Gates with Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
XNOR2 through 5					
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR6 through 9					
Macro	Macro	Macro	Primitive	Primitive	Primitive

XOR2-9		2- to 9-Input XOR Gates with Non-Inverted Inputs			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
XOR2 through 5					
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XOR6 through 9					
Macro	Macro	Macro	Primitive	Primitive	Primitive

XORCY		XOR for Carry Logic with General Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

XORCY_D		XOR for Carry Logic with Dual Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

XORCY_L		XOR for Carry Logic with Local Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

Map Elements

Map elements are used in conjunction with logic symbols to constrain the logic to particular CLBs or particular F or H function generators.

FMAP		F Function Generator Partitioning Control Symbol			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

Memory Elements

In the Virtex, Virtex-E, Spartan-II, and Spartan-IIE architectures, a number of static RAMs are defined as primitives. These 16- or 32-word RAMs are 1, 2, 4, and 8 bits wide.

The Virtex, Virtex-E, Spartan-II, and Spartan-IIE architectures have dedicated blocks of on-chip 4096-bit single-port and dual-port synchronous RAM. Each port is configured to a specific data width. There are five single-port block RAM primitives and 30 dual-port block RAM primitives.

RAM16X1D		16-Deep by 1-Wide Static Dual Port Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

RAM16X1D_1		16-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

RAM16X1S		16-Deep by 1-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

RAM16X1S_1		16-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

RAM16X2D		16-Deep by 2-Wide Static Dual Port Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

RAM16X2S		16-Deep by 2-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Primitive	N / A	N / A	N / A

RAM16X4D		16-Deep by 4-Wide Static Dual Port Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

RAM16X4S		16-Deep by 4-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Primitive	N / A	N / A	N / A

RAM16X8D		16-Deep by 8-Wide Static Dual Port Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	N / A	N / A	N / A

RAM16X8S		16-Deep by 8-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Primitive	N / A	N / A	N / A

RAM32X1D		32-Deep by 1-Wide Static Dual Port Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

RAM32X1D_1		32-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

RAM32X1S		32-Deep by 1-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

RAM32X1S_1		32-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N / A	N / A	N / A

RAM32X2S		32-Deep by 2-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Primitive	N / A	N / A	N / A

RAM32X4S		32-Deep by 4-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Primitive	N/A	N/A	N/A

RAM32X8S		32-Deep by 8-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Primitive	N/A	N/A	N/A

RAM64X1D		64-Deep by 1-Wide Static Dual Port Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

RAM64X1D_1		64-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

RAM64X1S		64-Deep by 1-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

RAM64X1S_1		64-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

RAM64X2S		64-Deep by 2-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

RAM128X1S		128-Deep by 1-Wide Static Synchronous RAM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

RAM128X1S_1		128-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

RAMB4_Sn		4096-Bit Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 8, or 16 Bits (5 primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	N/A	N/A	N/A	N/A

RAMB4_Sm_Sn		4096-Bit Dual-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 8, or 16 Bits (15 primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	N/A	N/A	N/A	N/A

RAMB16_Sn		16384-Bit Data Memory and 2048-Bit Parity Memory, Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 9, 18, or 36 Bits (6 primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

RAMB16_Sm_Sn		16384-Bit Data Memory and 2048-Bit Parity Memory, Dual-Port Synchronous Block RAM with Port Width (m or n) Configured to 1, 2, 4, 9, 18, or 36 Bits (21 primitives)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

ROM16X1		16-Deep by 1-Wide ROM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

ROM32X1		32-Deep by 1-Wide ROM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

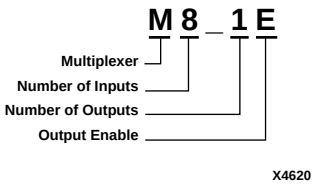
ROM64X1		64-Deep by 1-Wide ROM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

ROM128X1		128-Deep by 1-Wide ROM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

ROM256X1		256-Deep by 1-Wide ROM			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N / A	N / A	Primitive	N / A	N / A	N / A

Multiplexers

The multiplexer naming convention shown in the following figure indicates the number of inputs and outputs and whether or not an enable is available.



Multiplexer Naming Convention

M2_1		2-to-1 Multiplexer			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

M2_1B1		2-to-1 Multiplexer with D0 Inverted			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

M2_1B2		2-to-1 Multiplexer with D0 and D1 Inverted			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

M2_1E		2-to-1 Multiplexer with Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

M4_1E		4-to-1 Multiplexer with Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

M8_1E		8-to-1 Multiplexer with Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

M16_1E		16-to-1 Multiplexer with Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

MUXCY		2-to-1 Multiplexer for Carry Logic with General Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXCY_D		2-to-1 Multiplexer for Carry Logic with Dual Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXCY_L		2-to-1 Multiplexer for Carry Logic with Local Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXF5		2-to-1 Lookup Table Multiplexer with General Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXF5_D		2-to-1 Lookup Table Multiplexer with Dual Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXF5_L		2-to-1 Lookup Table Multiplexer with Local Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXF6		2-to-1 Lookup Table Multiplexer with General Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXF6_D		2-to-1 Lookup Table Multiplexer with Dual Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXF6_L		2-to-1 Lookup Table Multiplexer with Local Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

MUXF7		2-to-1 Lookup Table Multiplexer with General Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

MUXF7_D		2-to-1 Lookup Table Multiplexer with Dual Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

MUXF7_L		2-to-1 Lookup Table Multiplexer with Local Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

MUXF8		2-to-1 Lookup Table Multiplexer with General Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

MUXF8_D		2-to-1 Lookup Table Multiplexer with Dual Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

MUXF8_L		2-to-1 Lookup Table Multiplexer with Local Output			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

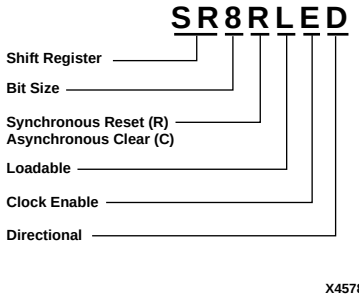
Shifters

Shifters are barrel shifters (BRLSHFT) of four and eight bits.

BRLSHFT4, 8		4-, 8-Bit Barrel Shifters			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

Shift Registers

Shift registers are available in a variety of sizes and capabilities. The naming convention shown in the following figure illustrates available features.



Shift Register Naming Convention

SR4CE, SR8CE, SR16CE		4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

SR4CLE, SR8CLE, SR16CLE		4-, 8-, 16-Bit Loadable Serial / Parallel-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

SR4CLED, SR8CLED, SR16CLED		4-, 8-, 16-Bit Shift Registers with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

SR4RE, SR8RE, SR16RE		4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

SR4RLE, SR8RLE, SR16RLE		4-, 8-, 16-Bit Loadable Serial / Parallel-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

SR4RLED, SR8RLED, SR16RLED		4-, 8-, 16-Bit Shift Registers with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Macro	Macro	Macro	Macro	Macro	Macro

SRD4CE, SRD8CE, SRD16CE		4-, 8-, 16-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

SRD4CLE, SRD8CLE, SRD16CLE		4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

SRD4CLED, SRD8CLED, SRD16CLED		4-, 8-, 16-Bit Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

SRD4RE, SRD8RE, SRD16RE		4-, 8-, 16-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

SRD4RLE, SRD8RLE, SRD16RLE		4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

SRD4RLED, SRD8RLED, SRD16RLED		4-, 8-, 16-Bit Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	N/A	N/A	N/A	Macro

SRL16		16-Bit Shift Register Look-Up-Table (LUT)			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

SRL16_1		16-Bit Shift Register Look-Up-Table (LUT) with Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

SRL16E		16-Bit Shift Register Look-Up-Table (LUT) with Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

SRLC16E_1		16-Bit Shift Register Look-Up-Table (LUT) with Negative-Edge Clock and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
Primitive	Primitive	Primitive	N/A	N/A	N/A

SRLC16		16-Bit Shift Register Look-Up-Table (LUT) with Carry			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

SRLC16_1		16-Bit Shift Register Look-Up-Table (LUT) with Carry and Negative-Edge Clock			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

SRLC16E		16-Bit Shift Register Look-Up-Table (LUT) with Carry and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

SRLC16E_1		16-Bit Shift Register Look-Up-Table (LUT) with Carry, Negative-Clock Edge, and Clock Enable			
Spartan-II, Spartan-IIE	**Virtex, Virtex-E**	**Virtex-II, Virtex-II Pro**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
N/A	N/A	Primitive	N/A	N/A	N/A

Design Elements

The remaining sections in the book describe each design element that can be used with the supported architectures.

Design elements are organized in alphanumeric order, with all numeric suffixes in ascending order. For example, FDR precedes FDRS, and ADD4 precedes ADD8, which precedes ADD16.

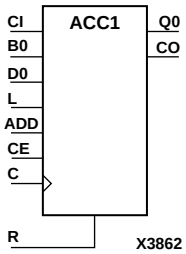
The following information is provided for each library element.

- Graphic symbol
- Applicability table (with primitive versus macro identification)
- Functional description
- Truth table (when applicable)
- Topology (when applicable)
- Schematic for macros
- VHDL and Verilog instantiation and inference code, if applicable
- Commonly used constraints, if applicable

ACC1

1-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	Macro	Macro	Macro



ACC1 can add or subtract a 1-bit unsigned-binary word to or from the contents of a 1-bit data register and store the results in the register. The register can be loaded with a 1-bit word. The synchronous reset (R) has priority over all other inputs and, when High, causes the output to go to logic level zero during the Low-to-High clock (C) transition. Clock (C) transitions are ignored when clock enable (CE) is Low. The accumulator is asynchronously cleared, outputs Low, when power is applied. For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Load

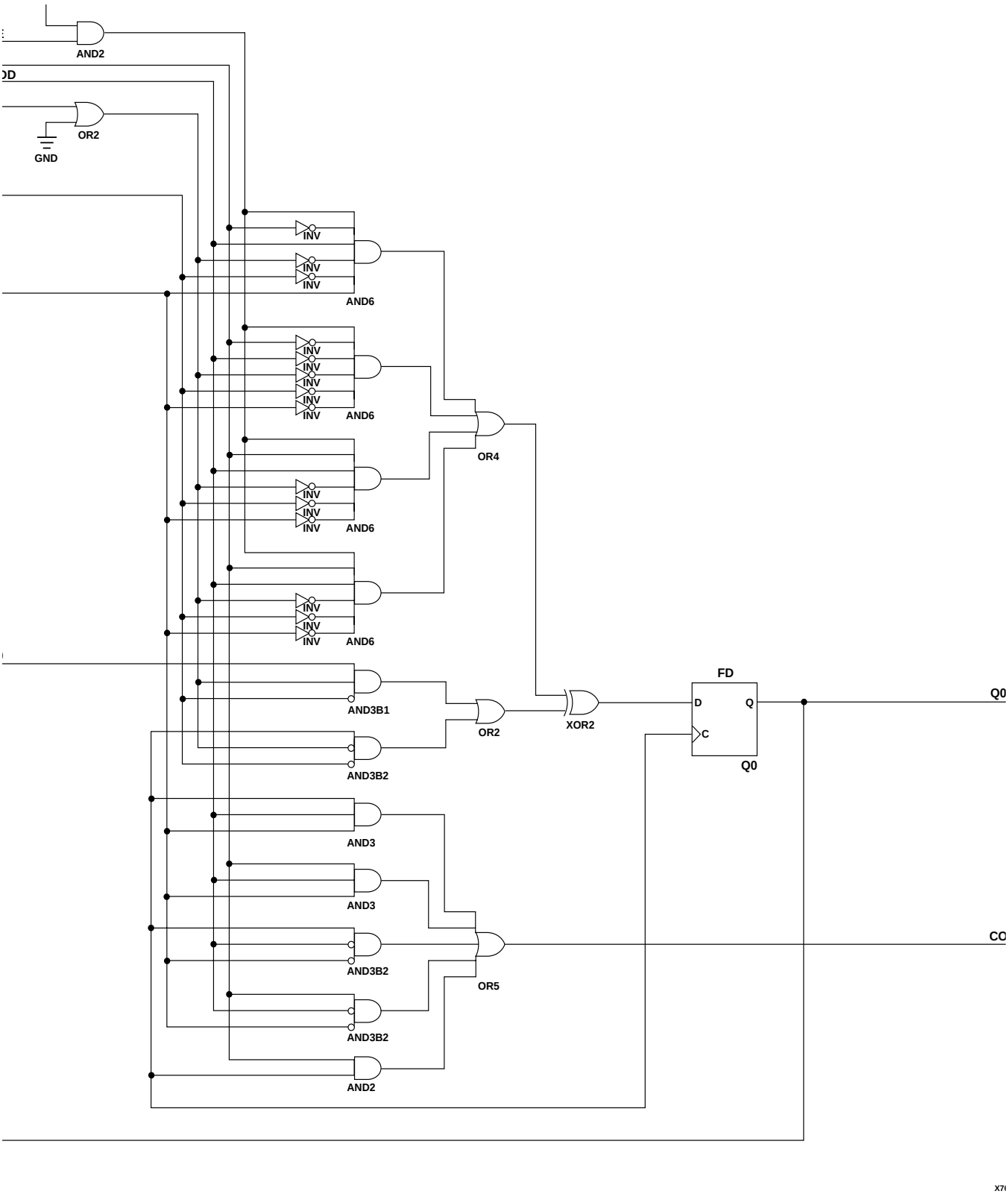
When the load input (L) is High, CE is ignored and the data on the input D0 is loaded into the 1-bit register during the Low-to-High clock (C) transition.

Add

When control inputs ADD and CE are both High, the accumulator adds a 1-bit word (B0) and carry-in (CI) to the contents of the 1-bit register. The result is stored in the register and appears on output Q0 during the Low-to-High clock transition. The carry-out (CO) is not registered synchronously with the data output. CO always reflects the accumulation of input B0 and the contents of the register, which allows cascading of ACC1s by connecting CO of one stage to CI of the next stage. In add mode, CO acts as a carry-out, and CO and CI are active-High.

Subtract

When ADD is Low and CE is High, the 1-bit word B0 and CI are subtracted from the contents of the register. The result is stored in the register and appears on output Q0 during the Low-to-High clock transition. The carry-out (CO) is not registered synchronously with the data output. CO always reflects the accumulation of input B0 and the contents of the register, which allows cascading of ACC1s by connecting CO of one stage to CI of the next stage. In subtract mode, CO acts as a borrow, and CO and CI are active-Low.



ACC1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

ACC is schematic and inference only-- not instantiated.

VHDL Inference Code

Following is some "basic" code for inference of the ACC modules.

```
architecture Behavioral of acc1 is
begin
 process(C, R)
 begin
  if (R = '1') then
   Q <= (others => '0');
  elsif (C'event and C = '1') then
   if (L = '1') then
    Q <= D;
   elsif (CE = '1') then
    if (ADD = '1') then
      Q <= Q + B;
    else
      Q <= Q - B;
    end if;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C)
begin
  if (R)
   Q <= 0;
  else if (L)
   Q <= D;
  else if (CE)
end

  if (ADD)
   Q <= Q + B;
  else
   Q <= Q - B;
 end
```

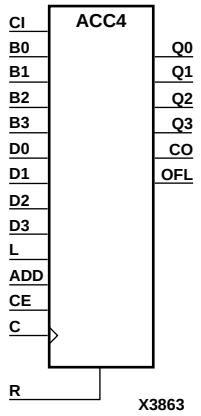
Commonly Used Constraints

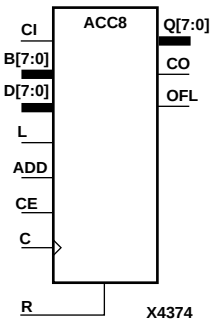
None

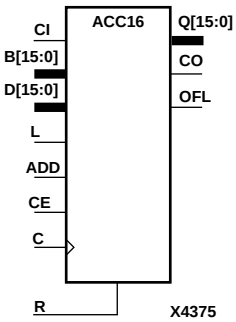
ACC4, 8, 16

4-, 8-, 16-Bit Loadable Cascadable Accumulators with Carry-In, Carry-Out, and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







ACC4, ACC8, ACC16 can add or subtract a 4-, 8-, 16-bit unsigned-binary, respectively or twos-complement word to or from the contents of a 4-, 8-, 16-bit data register and store the results in the register. The register can be loaded with the 4-, 8-, 16-bit word.

The synchronous reset (R) has priority over all other inputs, and when High, causes all outputs to go to logic level zero during the Low-to-High clock (C) transition. Clock (C) transitions are ignored when clock enable (CE) is Low.

The accumulator is asynchronously cleared, outputs Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Load

When the load input (L) is High, CE is ignored and the data on the D inputs is loaded into the register during the Low-to-High clock (C) transition. ACC4 loads the data on inputs D3 – D0 into the 4-bit register. ACC8 loads the data on D7 – D0 into the 8-bit register. ACC16 loads the data on inputs D15 – D0 into the 16-bit register.

Unsigned Binary Versus Twos Complement

ACC4, ACC8, ACC16 can operate, respectively, on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when “overflow” occurs. Unsigned binary uses CO, while twos complement uses OFL to determine when “overflow” occurs.

Unsigned Binary Operation

For unsigned binary operation, ACC4 can represent numbers between 0 and 15, inclusive; ACC8 between 0 and 255, inclusive; and ACC16 between 0 and 65535, inclusive. In add mode, CO is active (High) when the sum exceeds the bounds of the adder / subtracter. In subtract mode, CO is an active-Low borrow-out and goes Low when the difference exceeds the bounds. The carry-out (CO) is not registered synchronously with the data outputs. CO always reflects the accumulation of the B inputs (B3 – B0 for ACC4, B7 – B0 for ACC8, B15 – B0 for ACC16) and the contents of the register. This allows cascading of ACC4s, ACC8s, or ACC16s by connecting CO of one stage to CI of the next stage. An unsigned binary "overflow" that is always active-High can be generated by gating the ADD signal and CO as follows.

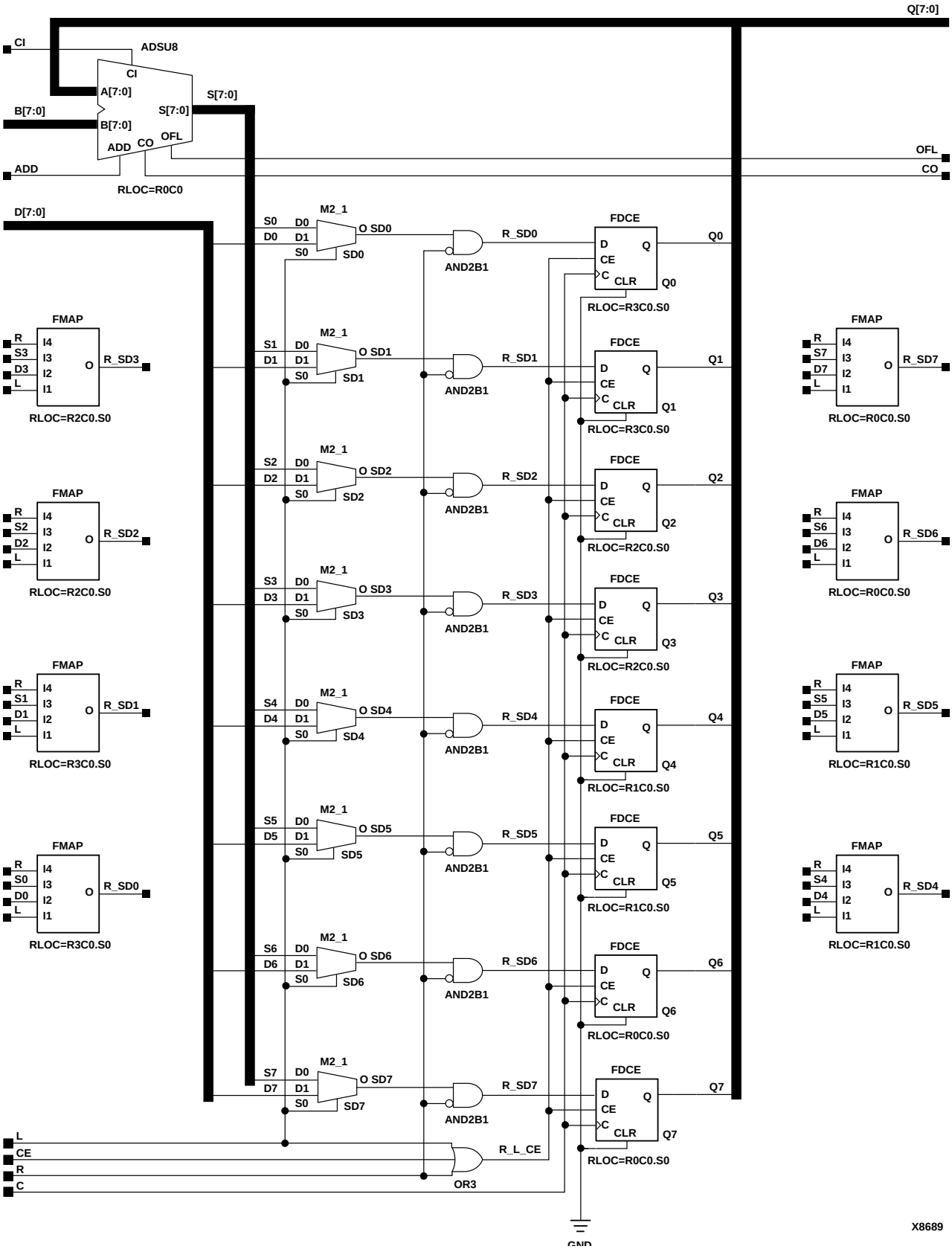
```
unsigned overflow = CO XOR ADD
```

Ignore OFL in unsigned binary operation.

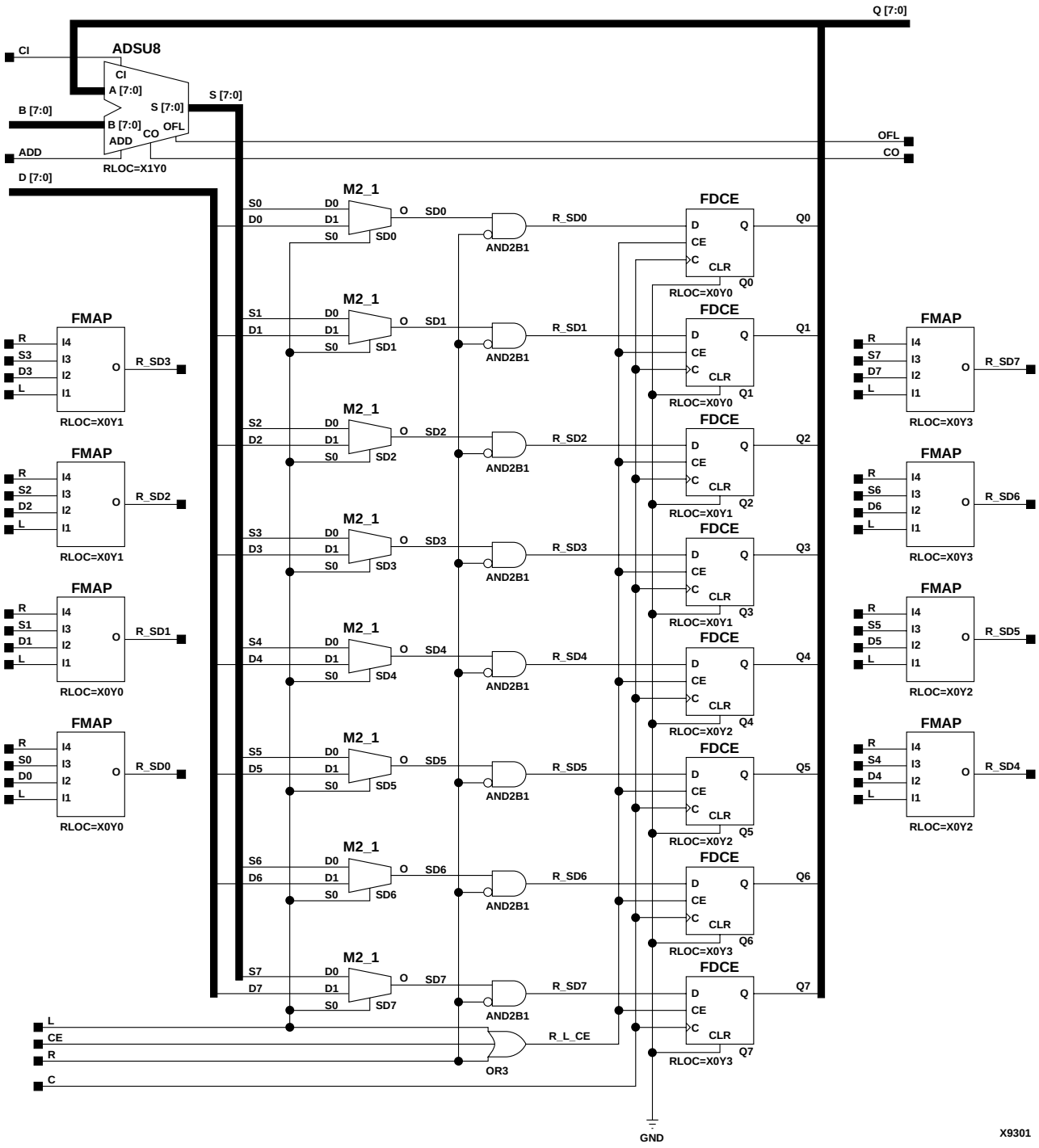
Twos-Complement Operation

For twos-complement operation, ACC4 can represent numbers between -8 and +7, inclusive; ACC8 between -128 and +127, inclusive; ACC16 between -32768 and +32767, inclusive. If an addition or subtraction operation result exceeds this range, the OFL output goes High. The overflow (OFL) is not registered synchronously with the data outputs. OFL always reflects the accumulation of the B inputs (B3 – B0 for ACC4, B7 – B0 for ACC8, B15 – B0 for ACC16) and the contents of the register, which allows cascading of ACC4s, ACC8s, or ACC16s by connecting OFL of one stage to CI of the next stage.

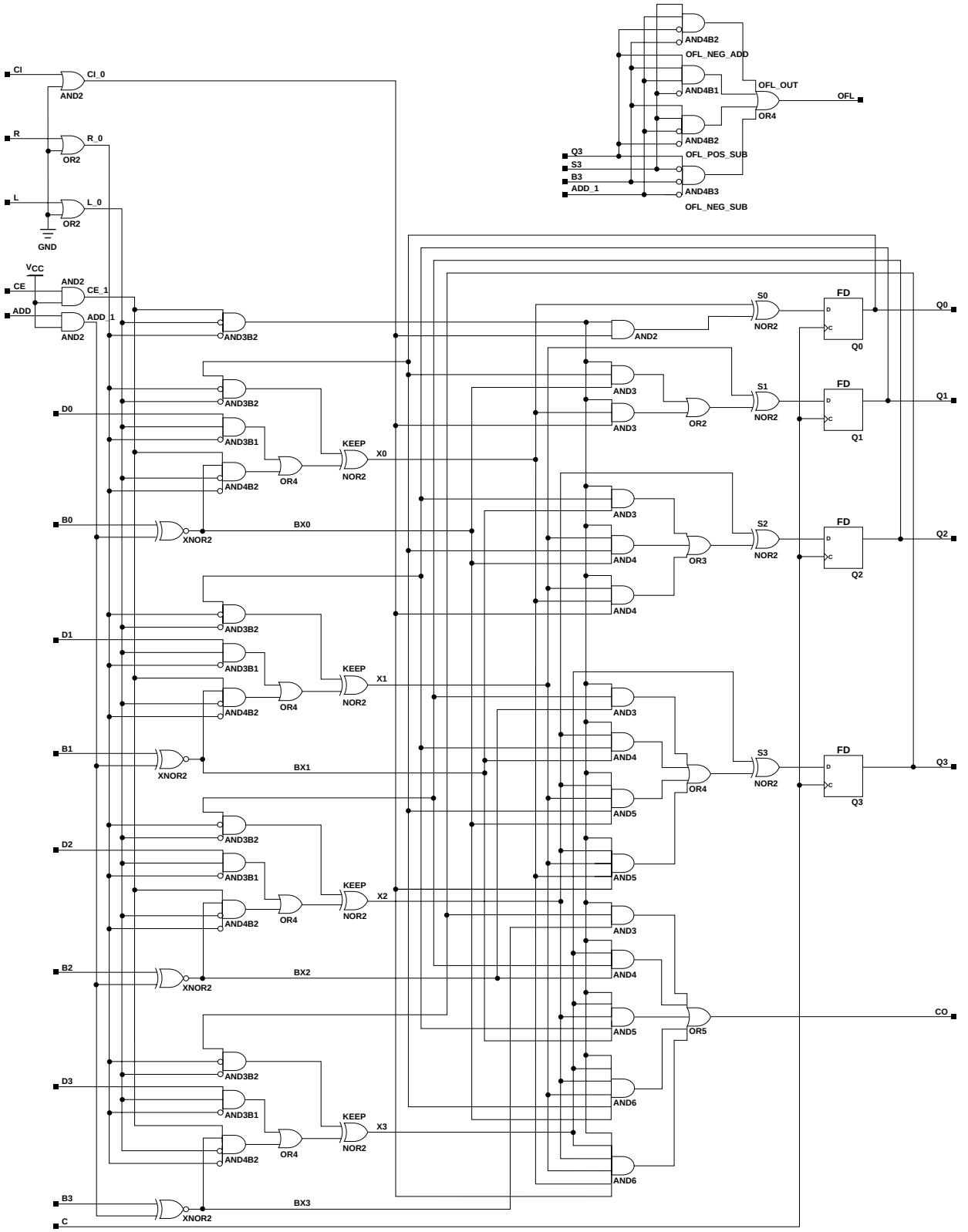
Ignore CO in twos-complement operation.



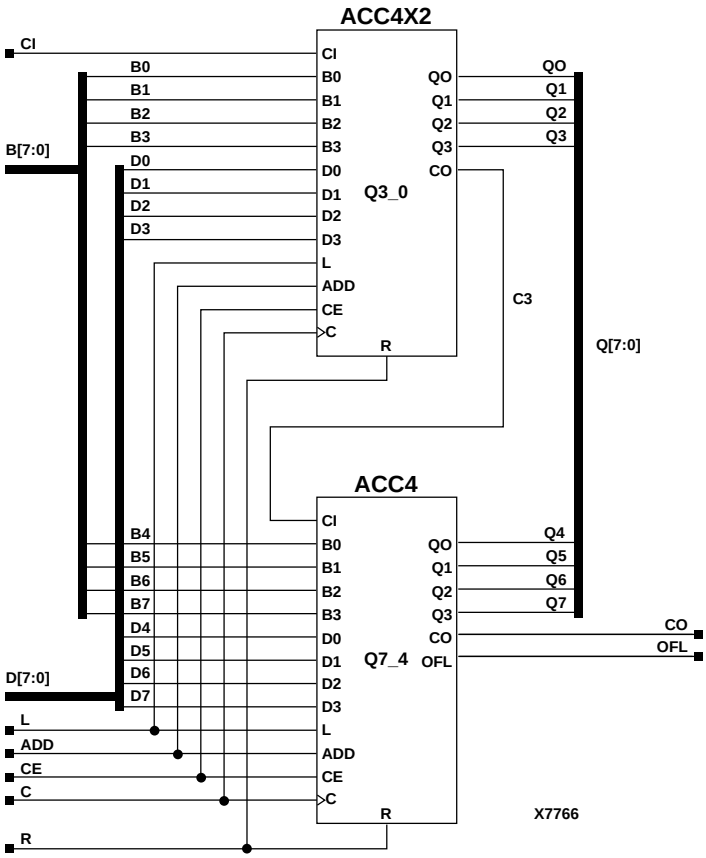
ACC8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ACC8 Implementation Virtex-II and Virtex-II Pro



ACC4 Implementation XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II



ACC8 Implementation XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II

Usage

ACC is schematic and inference only -- not instantiated.

VHDL Inference Code (ACC4)

Following is some "basic" code for inference of the ACC modules.

```
architecture Behavioral of acc4 is
begin
 process(C)
 begin
  if (R = '1') then
   Q <= (others => '0');
  elsif (C'event and C = '1') then
   if (L = '1') then
    Q <= D;
   elsif (CE = '1') then
    if (ADD = '1') then
      Q <= Q + B;
    else
      Q <= Q - B;
    end if;
   end if;
```

```
  end if;
 end process;
end Behavioral;
```

Note “Generic (WIDTH: Integer := 4)” at the top of the file. For ACC8 or ACC16, change the 4 to an 8 or 16.

Verilog Inference Code

```
always @ (posedge C)
 begin
  if (R)
   Q <= 0;
  else if (L)
   Q <= D;
  else if (CE)
 end
  if (ADD)
   Q <= Q + B;
  else
   Q <= Q - B;
 end
```

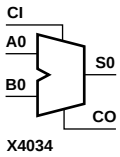
Commonly Used Constraints

None

ADD1

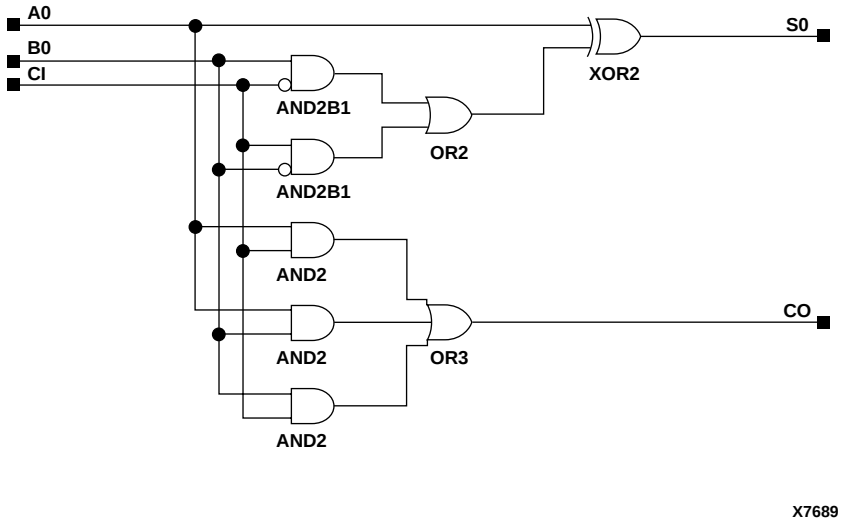
1-Bit Full Adder with Carry-In and Carry-Out

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



ADD1 is a cascadable 1-bit full adder with carry-in and carry-out. It adds two 1-bit words (A and B) and a carry-in (CI), producing a binary sum (S0) output and a carry-out (CO).

Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1



ADD1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is schematic or inference only -- no instantiation.

VHDL Inference Code

```
architecture Behavioral of ADD is

 signal sum: std_logic_vector(WIDTH downto 0);
 signal zeros: std_logic_vector(WIDTH-1 downto 0) := (others =>
   '0');

begin

process (CI, A, B, sum)
begin
 sum <= ('0' & A) + ('0' & B) + (zeros & CI);
 S <= sum(WIDTH-1 downto 0);
 CO <= sum(WIDTH);
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (A or B or CI)
begin
 {CO,S} <= A + B + CI;
end
```

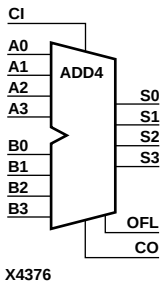
Commonly Used Constraints

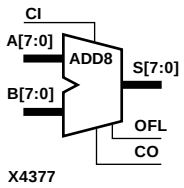
None

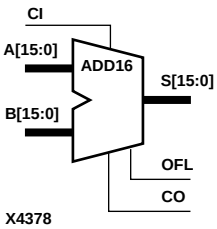
ADD4, 8, 16

4-, 8-, 16-Bit Cascadable Full Adders with Carry-In, Carry-Out, and Overflow

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







ADD4, ADD8, and ADD16 add two words and a carry-in (CI), producing a sum output and carry-out (CO) or overflow (OFL). ADD4 adds A3 – A0, B3 – B0, and CI producing the sum output S3 – S0 and CO (or OFL). ADD8 adds A7 – A0, B7 – B0, and CI, producing the sum output S7 – S0 and CO (or OFL). ADD16 adds A15 – A0, B15 – B0 and CI, producing the sum output S15 – S0 and CO (or OFL).

Unsigned Binary Versus Twos Complement

ADD4, ADD8, ADD16 can operate on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers, respectively. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when “overflow” occurs. Unsigned binary uses CO, while twos-complement uses OFL to determine when “overflow” occurs. To interpret the inputs as unsigned binary, follow the CO output. To interpret the inputs as twos complement, follow the OFL output.

Unsigned Binary Operation

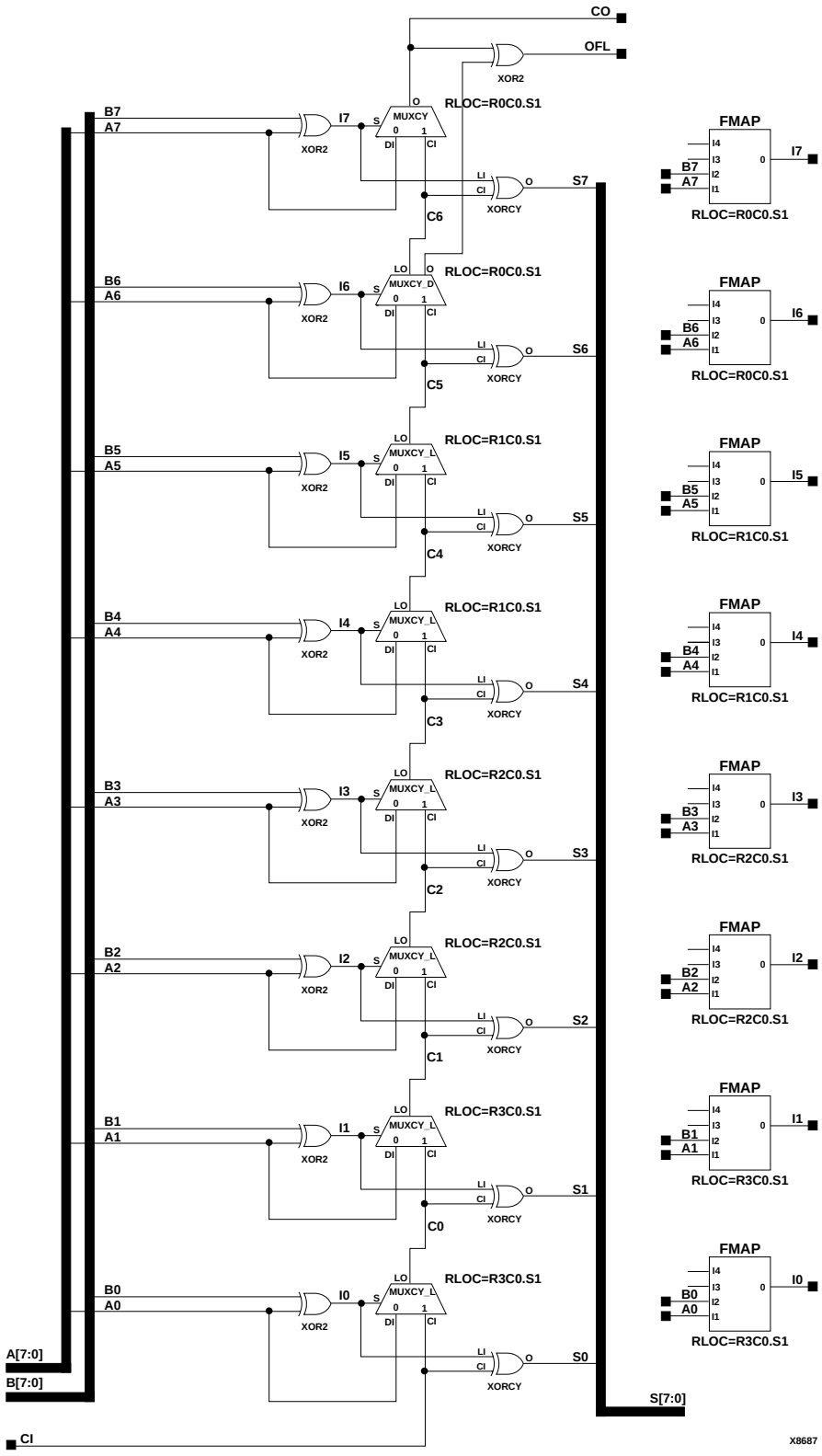
For unsigned binary operation, ADD4 can represent numbers between 0 and 15, inclusive; ADD8 between 0 and 255, inclusive; ADD16 between 0 and 65535, inclusive. CO is active (High) when the sum exceeds the bounds of the adder.

OFL is ignored in unsigned binary operation.

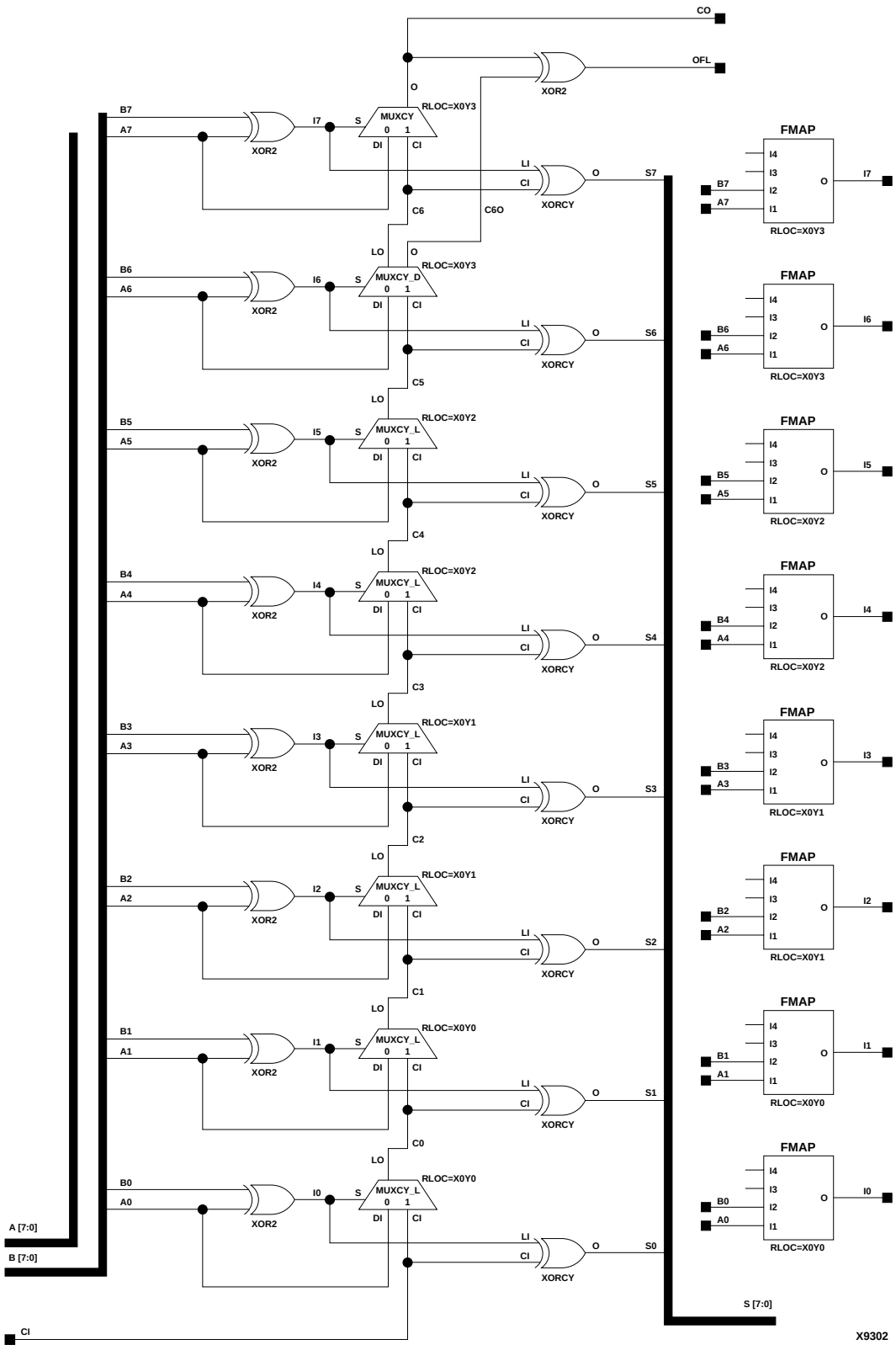
Twos-Complement Operation

For twos-complement operation, ADD4 can represent numbers between -8 and +7, inclusive; ADD8 between -128 and +127, inclusive; ADD16 between -32768 and +32767, inclusive. OFL is active (High) when the sum exceeds the bounds of the adder.

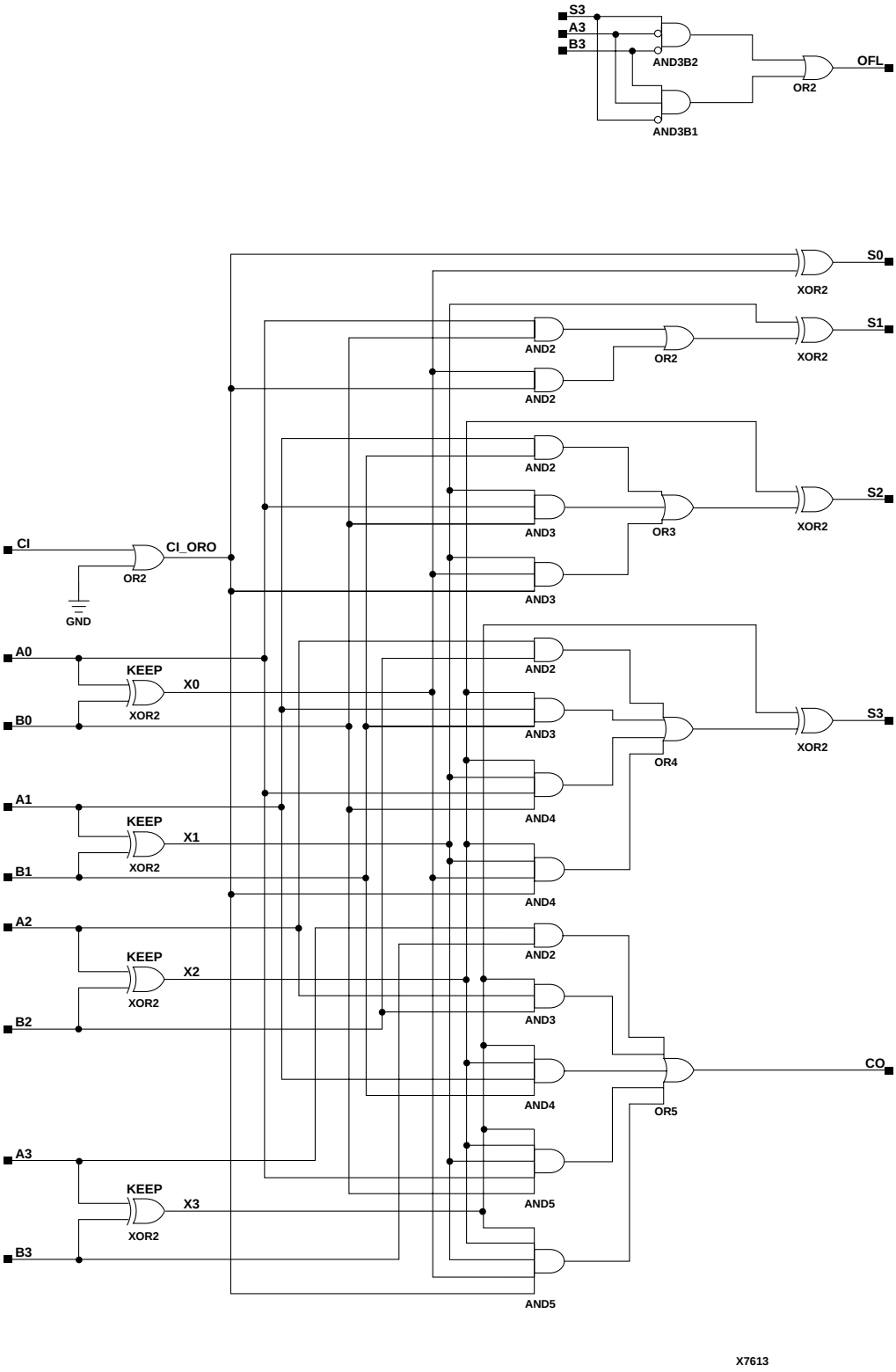
CO is ignored in twos-complement operation.



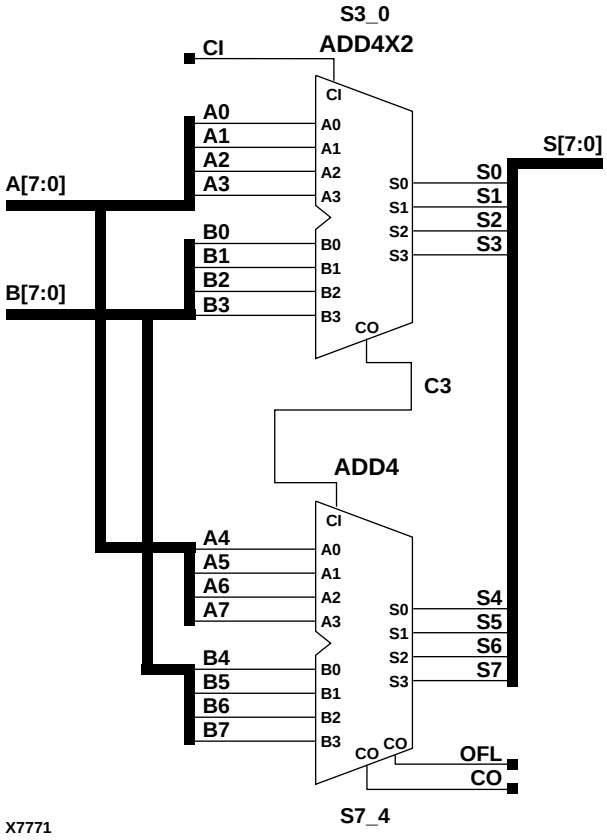
ADD8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ADD8 Implementation Virtex-II and Virtex-II Pro



ADD4 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



ADD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is schematic or inference only -- no instantiation.

VHDL Inference Code (ADD4)

```
architecture Behavioral of ADD is

 signal sum: std_logic_vector(WIDTH-1 downto 0);
 signal zeros: std_logic_vector(WIDTH-1 downto 0) := (others =>
   '0');

begin

process (CI, A, B, sum)
begin
 sum <= ('0' & A) + ('0' & B) + (zeros & CI);
 S <= sum(WIDTH-1 downto 0);
 CO <= sum(WIDTH);
end process;

end Behavioral;
```

Verilog Inference Code (ADD4)

```
always @ (A or B or CI)
begin
   {CO,sum} <= A + B + CI;
end
```

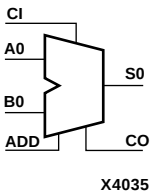
Commonly Used Constraints

None

ADSU1

1-Bit Cascadable Adder/Subtracter with Carry-In and Carry-Out

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



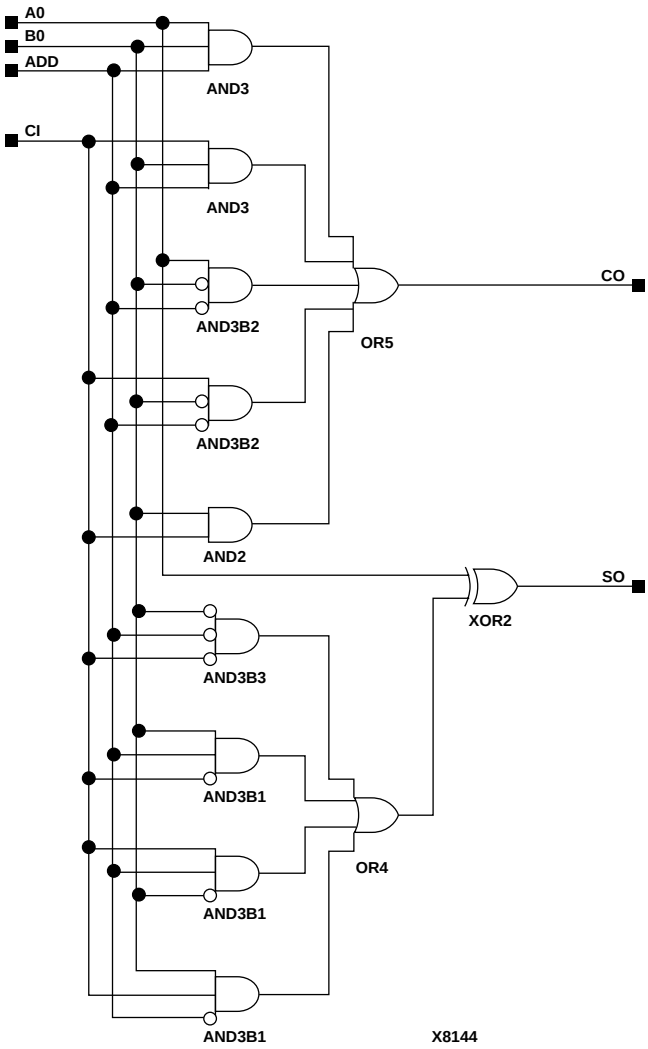
When the ADD input is High, two 1-bit words (A0 and B0) are added with a carry-in (CI), producing a 1-bit output (S0) and a carry-out (CO). When the ADD input is Low, B0 is subtracted from A0, producing a result (S0) and borrow (CO). In add mode, CO represents a carry-out, and CO and CI are active-High. In subtract mode, CO represents a borrow, and CO and CI are active-Low.

Add Function, ADD=1

Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Subtract Function, ADD=0

Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	1	0
0	1	0	0	0
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	1
1	1	1	0	1



ADSU1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of adsu4 is

begin
process (A0,ADD,B0)
begin
        if (ADD='1') then
                S0 <= A0 + B0;
        else
                S0 <= A0 - B0;
        end if;
end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (A0 or ADD or B0)
begin
        if (ADD)
                S0 <= A0 + B0;
        else
                S0 <= A0 - B0;
end
```

VHDL Instantiation Template

```
-- Component Declaration for ADSU1 should be placed
-- after architecture statement but before begin keyword

component ADSU1
   port (CO : out STD_ULOGIC;
         S0 : out STD_ULOGIC;
         A0 : in STD_ULOGIC;
         ADD: in STD_ULOGIC;
         B0 : in STD_ULOGIC;
         CI : in STD_ULOGIC);
end component;

-- Component Attribute specification for ADSU1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ADSU1 should be placed
-- in architecture after the begin keyword

ADSU1_INSTANCE_NAME : ADSU1
   -- synthesis translate_off
   generic (CDS_ACTION => "string_value");
   -- synthesis translate_on
      port map (CO  => user_CO,
                S0  => user_SO,
                A0  => user_A0,
                ADD => user_ADD,
                B0  => user_B0,
                CI  => user_CI);
```

Verilog Instantiation Template

```
ADSU1 instance_name(.CO (user_CO),
                    .S0 (user_SO),
                    .A0 (user_A0),
                    .ADD (user_ADD),
                    .B0 (user_B0),
                    .CI (user_CI));
```

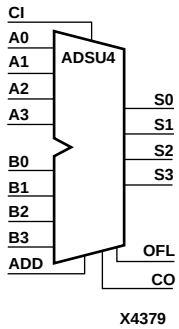
Commonly Used Constraints

None

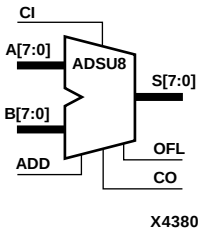
ADSU4, 8, 16

4-, 8-, 16-Bit Cascadable Adders/Subtracters with Carry-In, Carry-Out, and Overflow

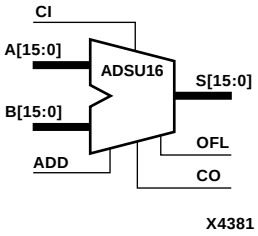
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



X4379



X4380



X4381

When the ADD input is High, ADSU4, ADSU8, and ADSU16 add two words and a carry-in (CI), producing a sum output and carry-out (CO) or overflow (OFL). ADSU4 adds two 4-bit words (A3 – A0 and B3 – B0) and a CI, producing a 4-bit sum output (S3 – S0) and CO or OFL. ADSU8 adds two 8-bit words (A7 – A0 and B7 – B0) and a CI producing, an 8-bit sum output (S7 – S0) and CO or OFL. ADSU16 adds two 16-bit words (A15 – A0 and B15 – B0) and a CI, producing a 16-bit sum output (S15 – S0) and CO or OFL.

When the ADD input is Low, ADSU4, ADSU8, and ADSU16 subtract Bz – B0 from Az– A0, producing a difference output and CO or OFL. ADSU4 subtracts B3 – B0 from A3 – A0, producing a 4-bit difference (S3 – S0) and CO or OFL. ADSU8 subtracts B7 – B0 from A7 – A0, producing an 8-bit difference (S7 – S0) and CO or OFL. ADSU16 subtracts B15 – B0 from A15 – A0, producing a 16-bit difference (S15 – S0) and CO or OFL.

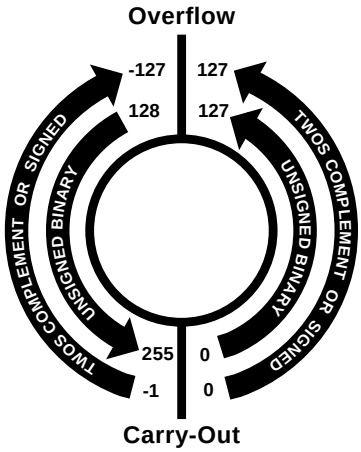
In add mode, CO and CI are active-High. In subtract mode, CO and CI are active-Low. OFL is active-High in add and subtract modes.

ADSU4, ADSU8, and ADSU16 CI and CO pins do not use the CPLD carry chain.

Unsigned Binary Versus Twos Complement

ADSU4, ADSU8, ADSU16 can operate, respectively, on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when “overflow” occurs. Unsigned binary uses CO, while twos complement uses OFL to determine when “overflow” occurs.

With adder / subtracters, either unsigned binary or twos-complement operations cause an overflow. If the result crosses the overflow boundary, an overflow is generated. Similarly, when the result crosses the carry-out boundary, a carry-out is generated. The following figure shows the ADSU carry-out and overflow boundaries.



ADSU Carry-Out and Overflow Boundaries

Unsigned Binary Operation

For unsigned binary operation, ADSU4 can represent numbers between 0 and 15, inclusive; ADSU8 between 0 and 255, inclusive; ADSU16 between 0 and 65535, inclusive. In add mode, CO is active (High) when the sum exceeds the bounds of the adder/subtracter. In subtract mode, CO is an active-Low borrow-out and goes Low when the difference exceeds the bounds.

An unsigned binary "overflow" that is always active-High can be generated by gating the ADD signal and CO as follows.

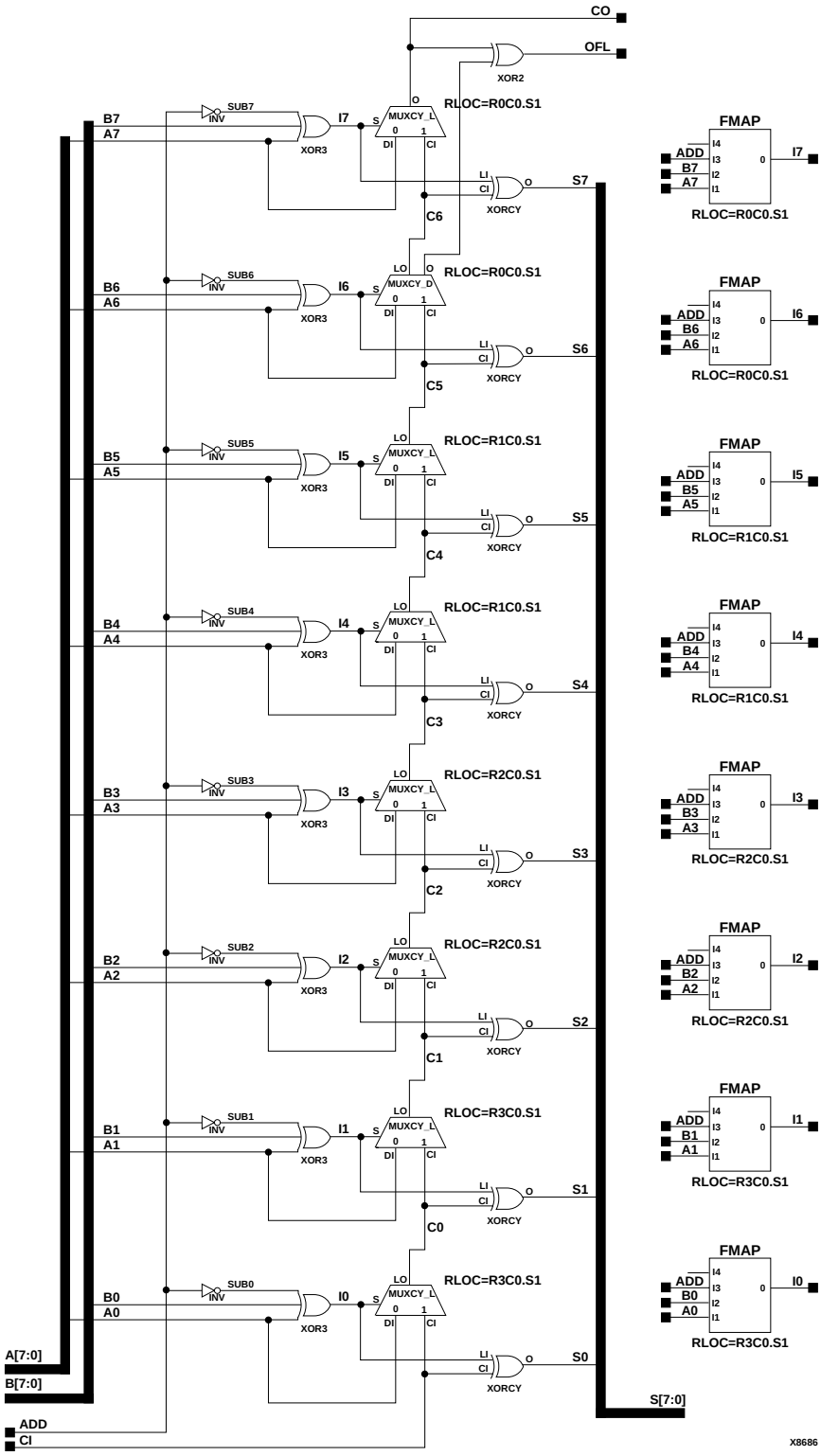
```
unsigned overflow = CO XOR ADD
```

OFL is ignored in unsigned binary operation.

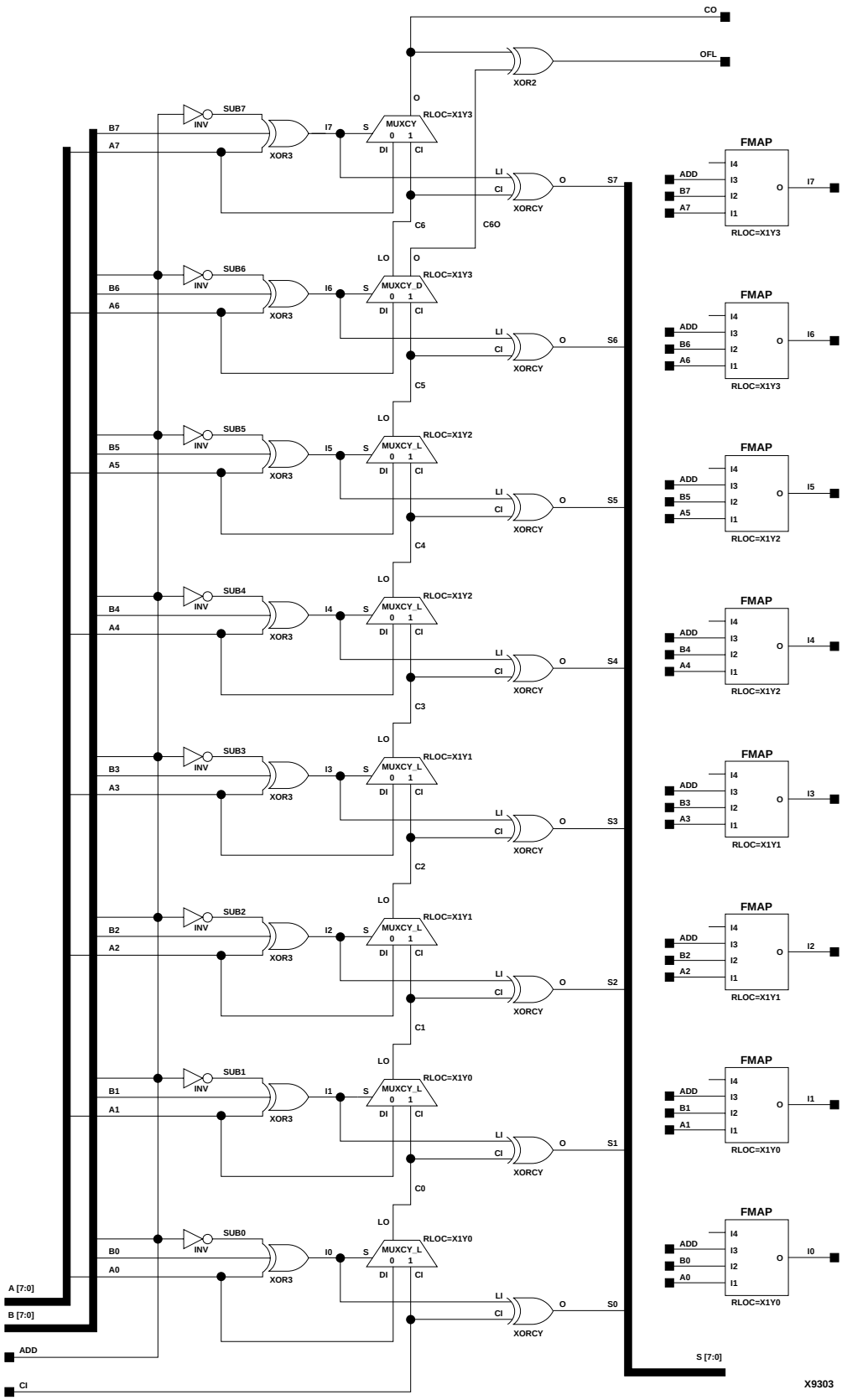
Twos-Complement Operation

For twos-complement operation, ADSU4 can represent numbers between -8 and +7, inclusive; ADSU8 between -128 and +127, inclusive; ADSU16 between -32768 and +32767, inclusive. If an addition or subtraction operation result exceeds this range, the OFL output goes High.

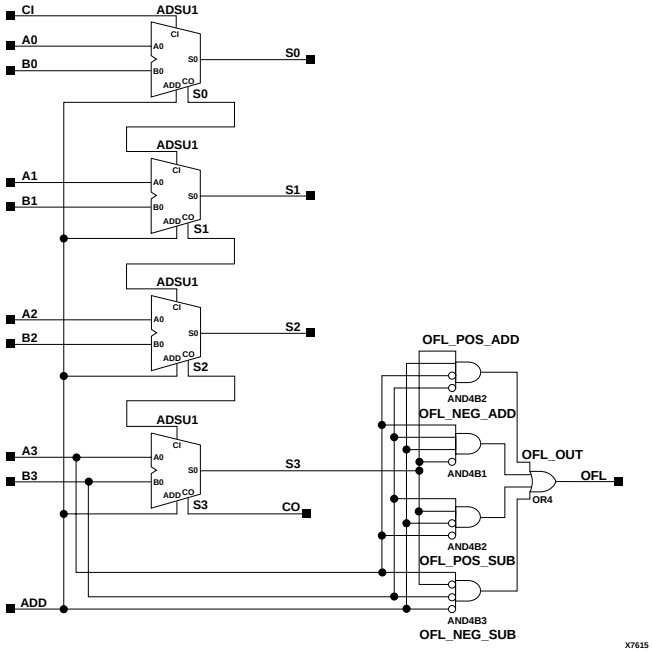
CO is ignored in twos-complement operation.



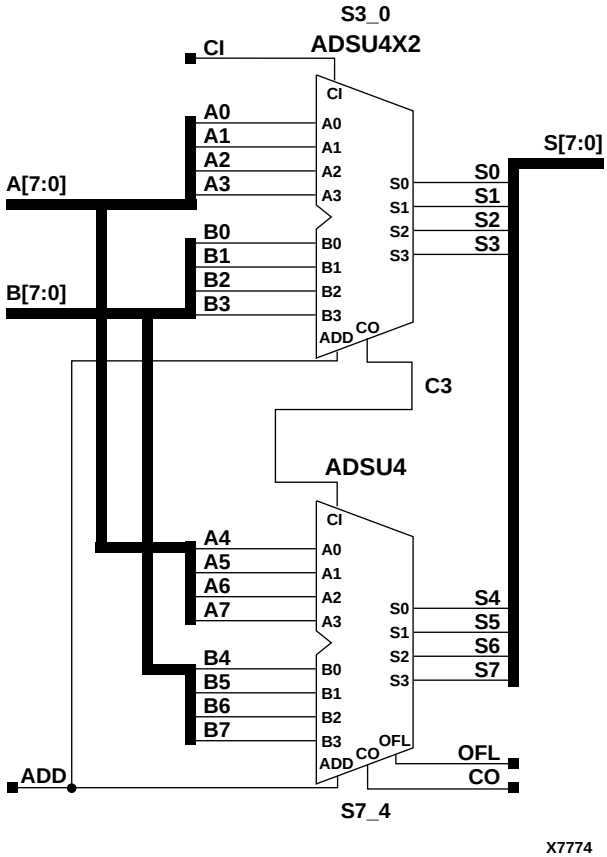
ADSU8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ADSU8 Implementation Virtex-II, Virtex-II Pro



ADSU4 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



ADSU8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usability

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of adsu4 is

begin

process (A,ADD,B)
begin
        if (ADD='1') then
                S <= A + B;
        else
                S <= A - B;
        end if;
end process;

end Behavioral;
```

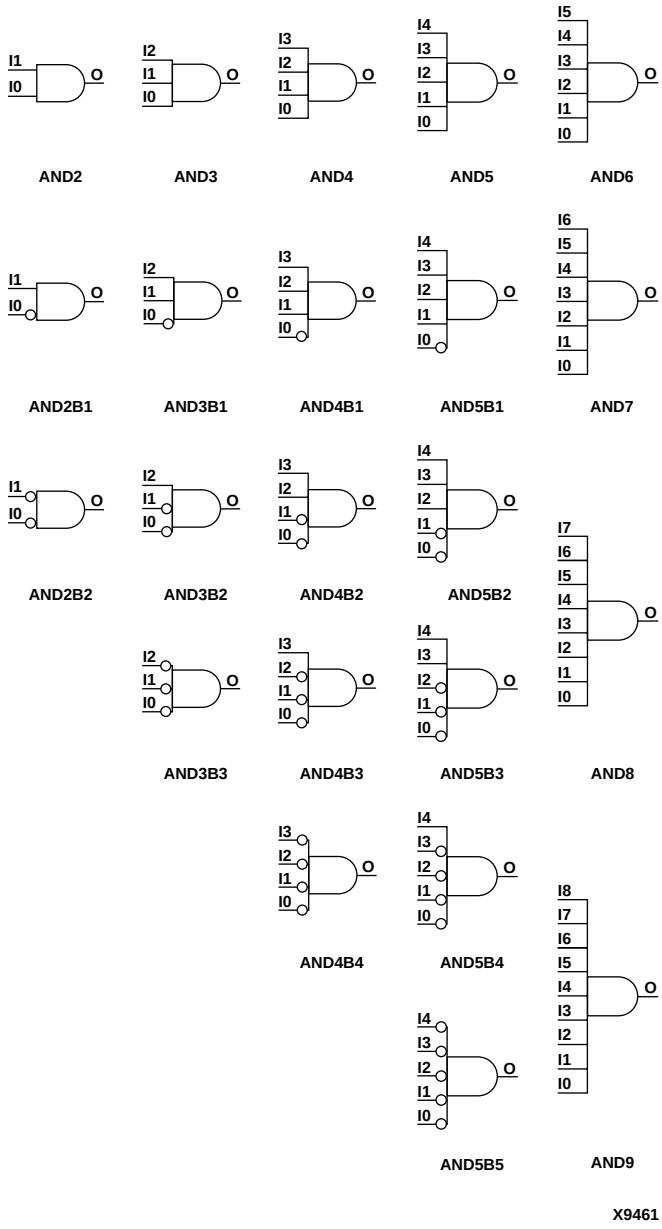
Verilog Inference Code

```
always @ (A or ADD or B)
begin
        if (ADD)
                S <= A + B;
        else
                S <= A - B;
end
```

AND2-9

2- to 9-Input AND Gates with Inverted and Non-Inverted Inputs

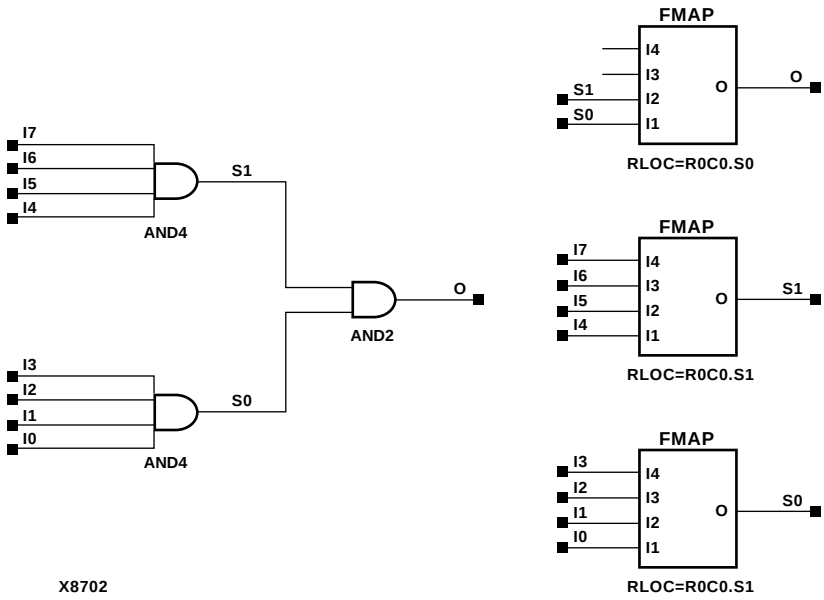
Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
AND2, AND2B1, AND2B2, AND3, AND3B1, AND3B2, AND3B3, AND4, AND4B1, AND4B2, AND4B3, AND4B4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND5, AND5B1, AND5B2, AND5B3, AND5B4, AND5B5	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND6, AND7, AND8, AND9	Macro	Macro	Macro	Primitive	Primitive	Primitive



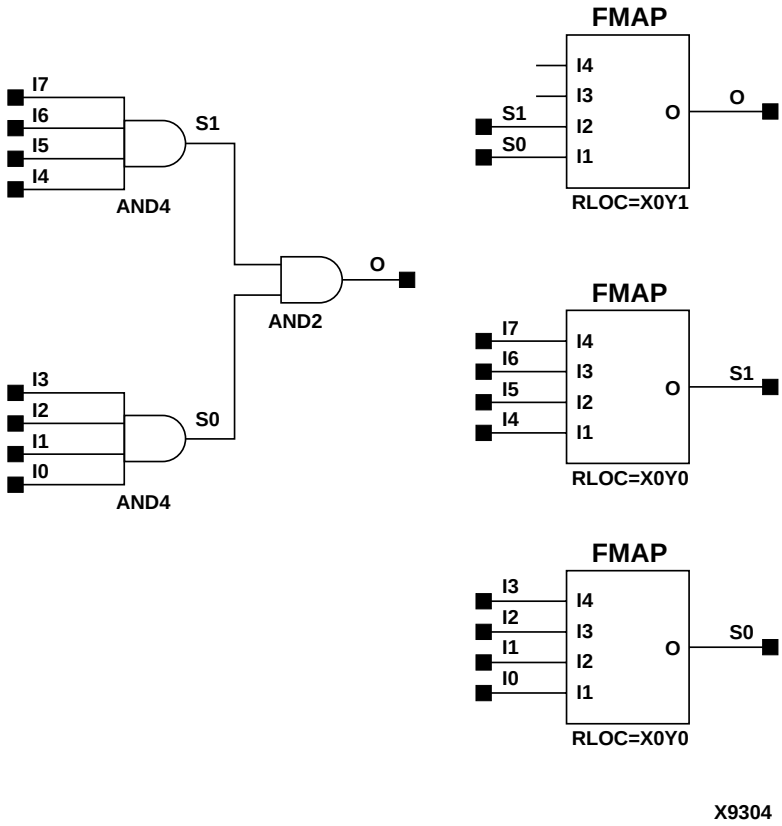
AND Gate Representations

AND functions of up to five inputs are available in any combination of inverting and non-inverting inputs. AND functions of six to nine inputs are available with only non-inverting inputs. To make some or all inputs inverting, use external inverters. Because each input uses a CLB resource in Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro replace functions with unused inputs with functions having the appropriate number of inputs.

See “AND12, 16” for information on additional AND functions for Virtex, Virtex-E, Virtex-II and Virtex-II Pro.



AND8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



AND8 Implementation Virtex-II, Virtex-II Pro

Usage

If possible, it is recommended that these design elements be inferred rather than instantiated.

VHDL Inference Code

AND2 example:

```
process (I0, I1)
begin
 O <= I0 and I1;
end process;
```

AND5B3 example:

```
process (I0, I1, I2, I3, I4)
begin
 O <= (not I0) and (not I1) and (not I2) and I3 and I4;
end process;
```

Verilog Inference Code

AND2:

```
always @ (I0 or I1)
begin
   O <= I0 && I1;
end
```

AND5B3:

```
always @ (I0 or I1 or I2 or I3 or I4)
begin
   O <= !I0 && !I1 && !I2 && I3 && I4;
end
```

VHDL Instantiation Template for AND2, AND2B1, or AND2B2

```
-- Component Declaration for AND2, AND2B1, or AND2B2 should
-- be placed after architecture statement but before begin
-- keyword

component {AND2|AND2B1|AND2B2}
   port (O    : out STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I0   : in STD_ULOGIC);
end component;

-- Component Attribute specification for AND2, AND2B1, or
-- AND2B2 should be placed after architecture declaration
-- but before the begin keyword

-- Enter attributes here


-- Component Instantiation for AND2, AND2B1, or AND2B2
```

```
-- should be placed in architecture after the begin
-- keyword

INSTANCE_NAME : {AND2|AND2B1|AND2B2}
   port map   (O => user_O,
               I0 => user_I0,
               I1 => user_I1);
```

Verilog Instantiation Template for AND2, AND2B1, or AND2B2

```
ANDn instance_name (.O (user_O),
                    .I0 (user_I0),
                    .I1 (user_I1));
```

VHDL Instantiation Template for AND3 Through AND3B3

```
-- Component Declaration for AND3 through AND3B3 should
-- be placed after architecture statement but before begin
-- keyword

component {AND3|AND3B1|AND3B2|AND3B3}
   port (O    : out STD_ULOGIC;
         I0   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC);
end component;

-- Component Attribute specification for AND3 through AND3B3
-- should be placed after architecture declaration
-- but before the begin keyword

-- Enter attributes here

-- Component Instantiation for AND3 through AND3B3
-- should be placed in architecture after the begin
-- keyword

INSTANCE_NAME : {AND3|AND3B1|AND3B2|AND3B3}
port map(O => user_O,
         I0 => user_I0,
         I1 => user_I1,
         I2 => user_I2);
```

Verilog Instantiation Template for AND3 Through AND3B3

```
ANDn instance_name (.O (user_O),
                    .I0 (user_I0),
                    .I1 (user_I1),
                    .I2 (user_I2));
```

VHDL Instantiation Template for AND4 Through AND4B4

```
-- Component Declaration for AND4 through AND4B4 should
-- be placed after architecture statement but before begin
-- keyword

component {AND4|AND4B1|AND4B2|AND4B3|AND4B4}
   port (O    : out STD_ULOGIC;
         I0   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC;
         I3   : in STD_ULOGIC);
end component;

-- Component Attribute specification for AND4 through AND4B4
-- should be placed after architecture declaration
-- but before the begin keyword

-- Enter attributes here

-- Component Instantiation for AND4 through AND4B4
-- should be placed in architecture after the begin
-- keyword

INSTANCE_NAME : AND4_thru_AND4B4
      port map  (O => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3);
```

Verilog Instantiation Template for AND4 through AND4B4

```
ANDn instance_name (.O (user_O),
                    .I0 (user_I0),
                    .I1 (user_I1),
                    .I2 (user_I2),
                    .I3 (user_I3));
```

VHDL Instantiation Template for AND5 Through AND5B5

```
-- Component Declaration for AND5 through AND5B5 should
-- be placed after architecture statement but before begin
-- keyword

component {AND5|AND5B1|AND5B2|AND5B3|AND5B4|AND5B5}
   port (O    : out STD_ULOGIC;
         I0   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC;
         I3   : in STD_ULOGIC;
         I4   : in STD_ULOGIC);
end component;

-- Component Attribute specification for AND5 through AND5B5
-- should be placed after architecture declaration
-- but before the begin keyword

-- Enter attributes here

-- Component Instantiation for AND5 through AND5B5
-- should be placed in architecture after the begin
-- keyword

INSTANCE_NAME : AND5_thru_AND5B5
      port map  (O => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3,
                 I4 => user_I4);
```

Verilog Instantiation Template for AND5 through AND5B5

```
ANDn instance_name (.O (user_O),
                    .I0 (user_I0),
                    .I1 (user_I1),
                    .I2 (user_I2),
                    .I3 (user_I3),
                    .I4 (user_I4));
```

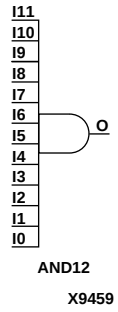
Commonly Used Constraints

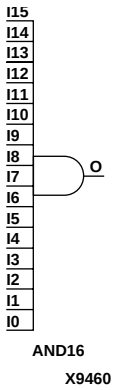
None

AND12, 16

12- and 16-Input AND Gates with Non-Inverted Inputs

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A

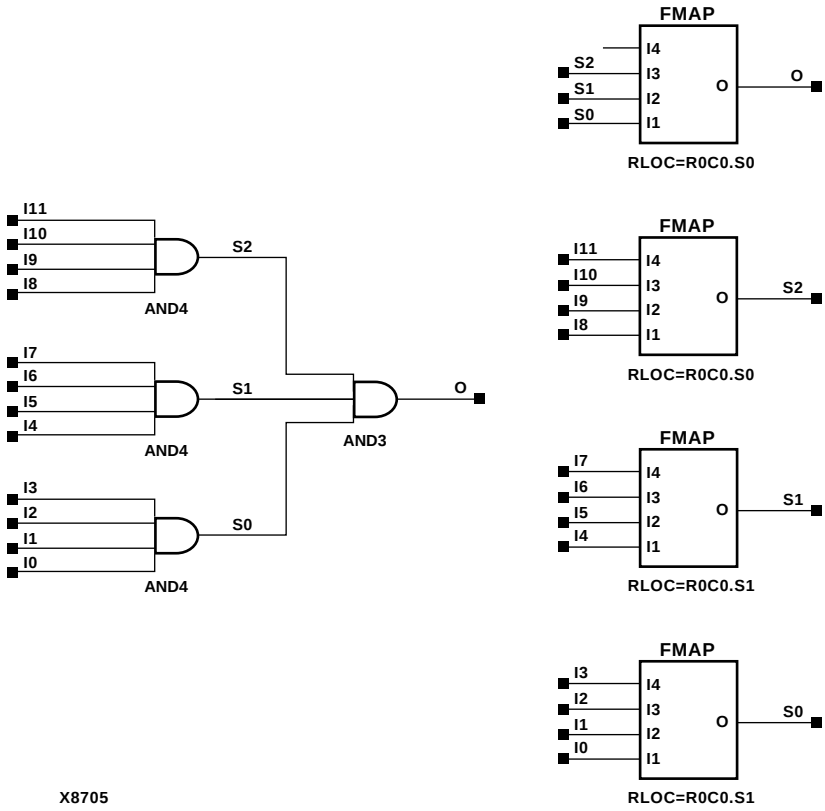




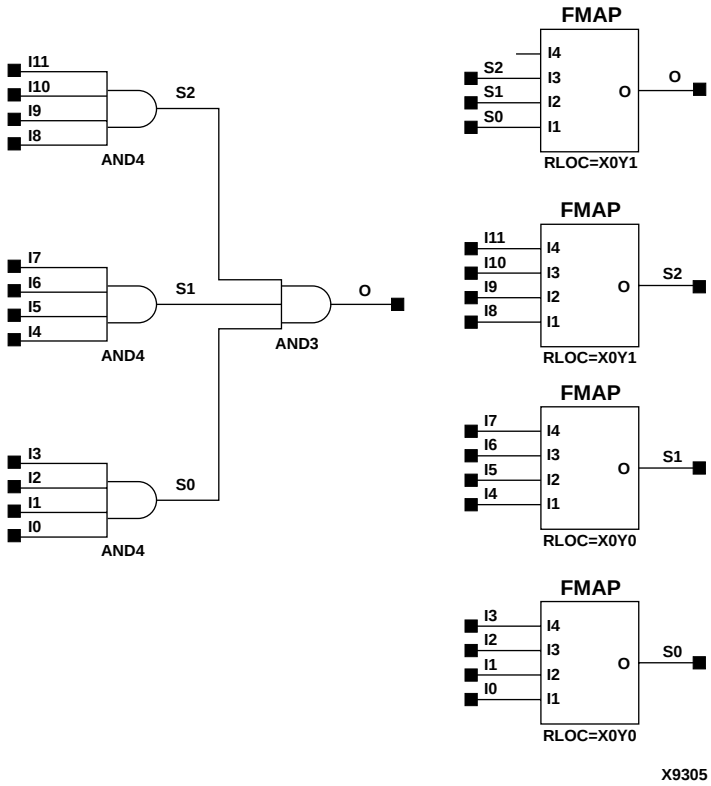
AND12 and AND16 functions are performed in the Configurable Logic Block (CLB) function generator.

The 12- and 16-input AND functions are available only with non-inverting inputs. To invert all of some inputs, use external inverters.

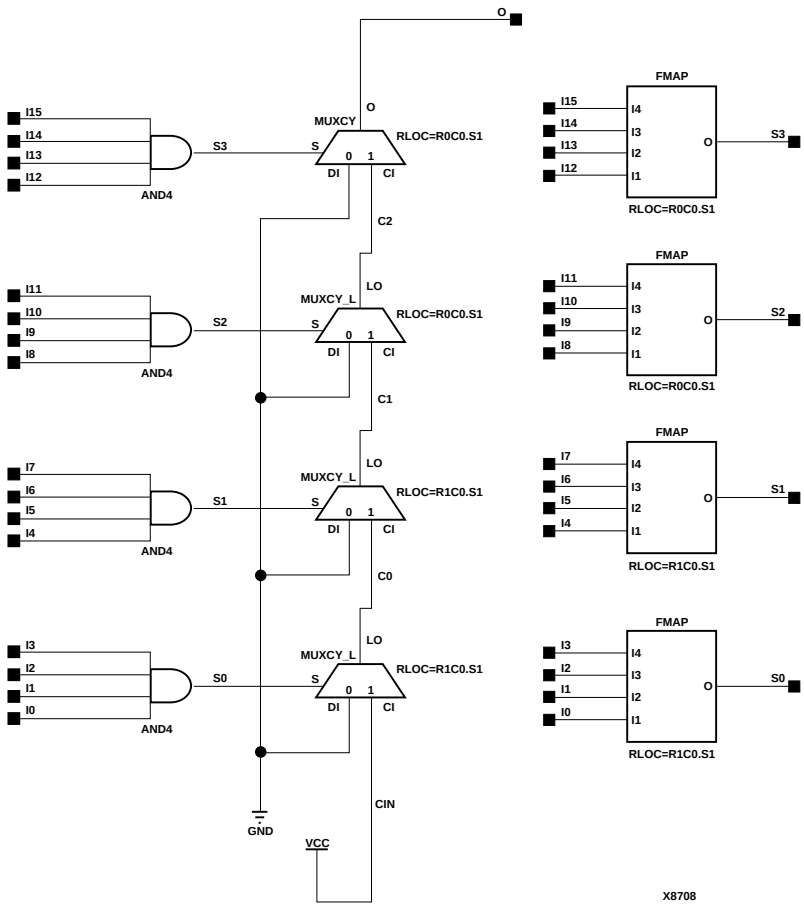
See "AND2-9" for information on more AND functions.



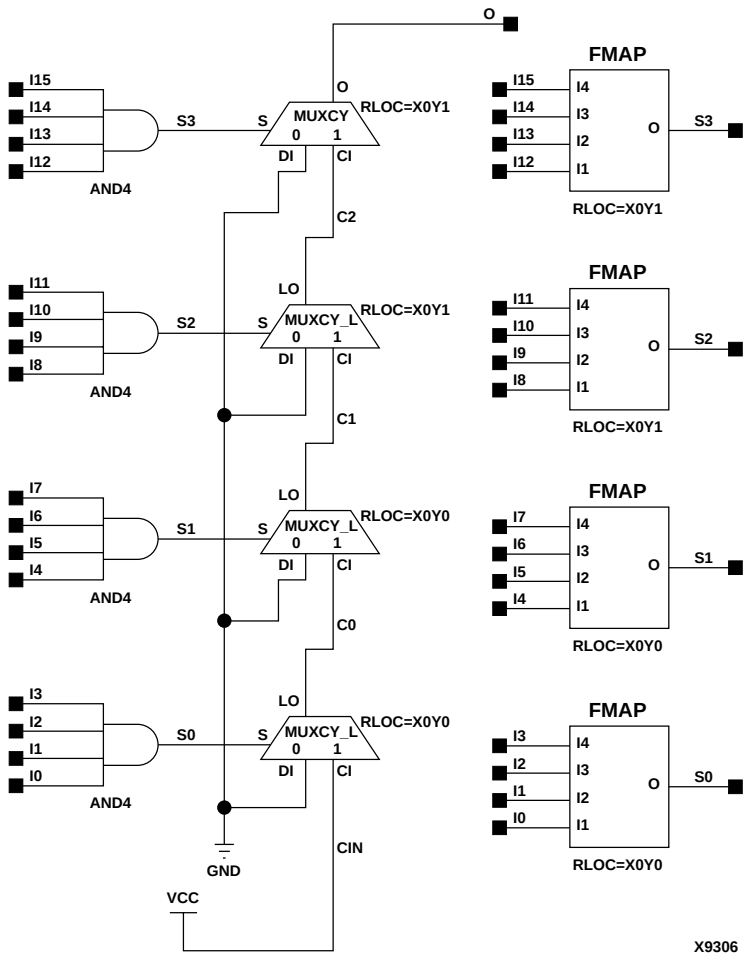
AND12 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



AND12 Implementation Virtex-II, Virtex-II Pro



AND16 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



AND16 Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, it is recommended that these design elements be inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of and12 is

begin

process (I0, I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11)
begin
 O <= I0 and I1 and I2 and I3 and I4 and I5 and I6 and I7
   and I8 and I9 and I10 and I11;
end process;


end Behavioral;
```

Verilog Inference Code

```
always @ (I0 or I1 or I2 or I3 or I4 or I5
   or I6 or I7 or I8 or I9 or I10 or I11)
begin
  O <= I0 && I1 && I2 && I3 && I4 && I5
   && I6 && I7 && I8 && I9 && I10 && I11;
end
```

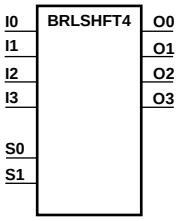
Commonly Used Constraints

None

BRLSHFT4, 8

4-, 8-Bit Barrel Shifters

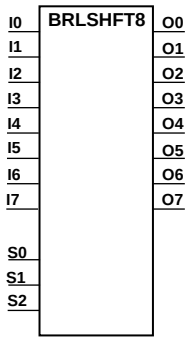
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



X3856

BRLSHFT4, a 4-bit barrel shifter, can rotate four inputs (I3 – I0) up to four places. The control inputs (S1 and S0) determine the number of positions, from one to four, that the data is rotated. The four outputs (O3 – O0) reflect the shifted data inputs.

BRLSHFT8, an 8-bit barrel shifter, can rotate the eight inputs (I7 – I0) up to eight places. The control inputs (S2 – S0) determine the number of positions, from one to eight, that the data is rotated. The eight outputs (O7 – O0) reflect the shifted data inputs.



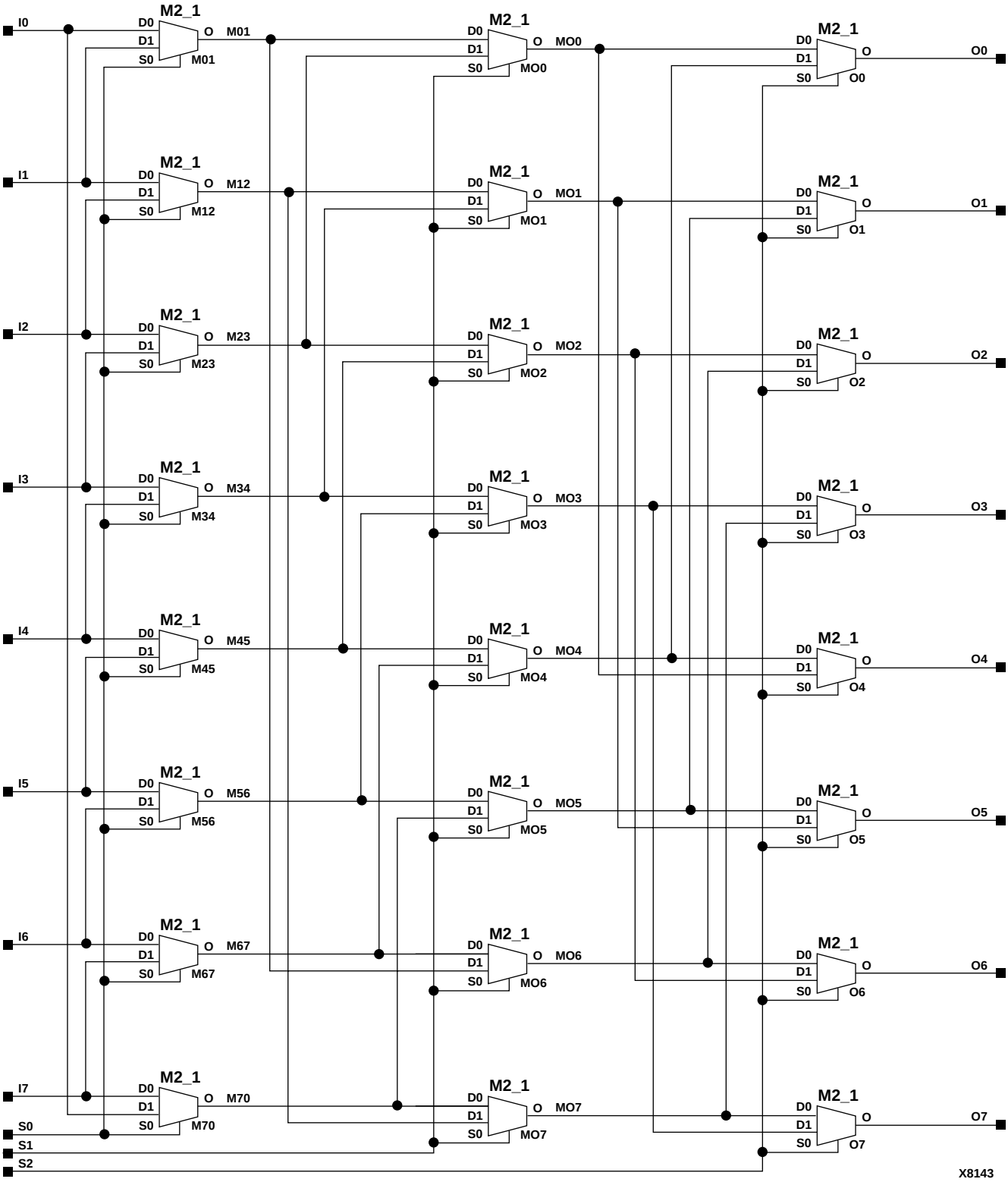
X3857

BRLSHFT4 Truth Table

Inputs						Outputs			
S1	S0	I0	I1	I2	I3	O0	O1	O2	O3
0	0	a	b	c	d	a	b	c	d
0	1	a	b	c	d	b	c	d	a
1	0	a	b	c	d	c	d	a	b
1	1	a	b	c	d	d	a	b	c

BRLSHFT8 Truth Table

Inputs											Outputs							
S2	S1	S0	I0	I1	I2	I3	I4	I5	I6	I7	O0	O1	O2	O3	O4	O5	O6	O7
0	0	0	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h
0	0	1	a	b	c	d	e	f	g	h	b	c	d	e	f	g	h	a
0	1	0	a	b	c	d	e	f	g	h	c	d	e	f	g	h	a	b
0	1	1	a	b	c	d	e	f	g	h	d	e	f	g	h	a	b	c
1	0	0	a	b	c	d	e	f	g	h	e	f	g	h	a	b	c	d
1	0	1	a	b	c	d	e	f	g	h	f	g	h	a	b	c	d	e
1	1	0	a	b	c	d	e	f	g	h	g	h	a	b	c	d	e	f
1	1	1	a	b	c	d	e	f	g	h	h	a	b	c	d	e	f	g



BRLSHFT8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of brlshft4 is

begin

process (I,S)
begin
        case S is
                when "00" =>
                        O <= I;
                when "01" =>
                        O(3) <= I(0);
                        O(2 downto 0) <= I(3 downto 1);
                when "10" =>
                        O(3 downto 2) <= I(1 downto 0);
                        O(1 downto 0) <= I(3 downto 2);
                when "11" =>
                        O(3 downto 1) <= I(2 downto 0);
                        O(0) <= I(3);
                when others =>
                        O <= I;
        end case;
end process;

end Behavioral;
```

Verilog Inference Code

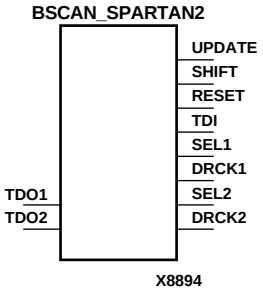
```
always @(I or S)
begin
        case (S)
                2'b00 : O <= I;
                2'b01 : O <= {I[0],I[3:1]};
                2'b10 : O <= {I[1:0],I[3:2]};
                2'b11 : O <= {I[2:0],I[3]};
        endcase
end
```

BSCAN_SPARTAN2

Spartan-II Boundary Scan Logic Control Circuit

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive*	N / A	N / A	N / A	N / A	N / A

* Supported for Spartan-II, but not for Spartan-IIE, which is supported by BSCAN_VIRTEX.



The BSCAN_SPARTAN2 symbol creates internal boundary scan chains in a Spartan-II device. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Spartan-II. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_SPARTAN2 symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active.The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, and SHIFT pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Note For specific information on boundary scan for an architecture, see *The Programmable Logic Data Book.*

Usage

This design element is supported for instantiation and schematics but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BSCAN_SPARTAN2 should be placed
-- after architecture statement but before begin keyword

component BSCAN_SPARTAN2
   port (DRCK1  : out STD_ULOGIC;
         DRCK2  : out STD_ULOGIC;
         RESET  : out STD_ULOGIC;
         SEL1   : out STD_ULOGIC;
         SEL2   : out STD_ULOGIC;
         SHIFT  : out STD_ULOGIC;
         TDI    : out STD_ULOGIC;
         UPDATE : out STD_ULOGIC;
         TD01   : in STD_ULOGIC;
         TD02   : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for BSCAN_SPARTAN2
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BSCAN_SPARTAN2 should be
-- placed in architecture after the begin keyword

BSCAN_SPARTAN2_INSTANCE_NAME : BSCAN_SPARTAN2
      port map  (DRCK1  => user_DRCK1,
                 DRCK2  => user_DRCK2,
                 RESET  => user_RESET,
                 SEL1   => user_SEL1,
                 SEL2   => user_SEL2,
                 SHIFT  => user_SHIFT,
                 TDI    => user_TDI,
                 UPDATE => user_UPDATE,
                 TD01   => user_TD01,
                 TD02   => user_TD02);
```

Verilog Instantiation Template

```
BSCAN_SPARTAN2 instance_name (.DRCK1(user_DRCK1),
                              .DRCK2(user_DRCK2),
                              .RESET(user_RESET),
                              .SEL1 (user_SEL1),
                              .SEL2 (user_SEL2),
                              .SHIFT(user_SHIFT),
                              .TDI  (user_TDI),
                              .UPDATE(user_UPDATE),
                              .TD01(user_TD01),
                              .TD02(user_TD02));
```

Commonly Used Constraints

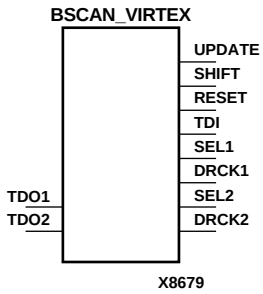
None

BSCAN_VIRTEX

Virtex Boundary Scan Logic Control Circuit

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive*	Primitive	N/A	N/A	N/A	N/A

* Supported for Spartan-IIE, but not for Spartan-II, which is supported by BSCAN_SPARTAN2.



The BSCAN_VIRTEX symbol is used to create internal boundary scan chains in a Virtex or Virtex- E device. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Virtex and Virtex-E. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_VIRTEX symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

Note For Virtex-II and Virtex-II Pro, see BSCAN_VIRTEX2.

A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active.The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, and SHIFT pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Note For specific information on boundary scan for an architecture, see *The Programmable Logic Data Book.*

Usage

This design element is supported for instantiation and schematics but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BSCAN_VIRTEX should be placed
-- after architecture statement but before begin keyword

component BSCAN_VIRTEX
   port (DRCK1  : out STD_ULOGIC;
         DRCK2  : out STD_ULOGIC;
         RESET  : out STD_ULOGIC;
         SEL1   : out STD_ULOGIC;
         SEL2   : out STD_ULOGIC;
         SHIFT  : out STD_ULOGIC;
         TDI    : out STD_ULOGIC;
         UPDATE : out STD_ULOGIC;
         TD01   : in STD_ULOGIC;
         TD02   : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for BSCAN_VIRTEX
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BSCAN_VIRTEX should be
-- placed in architecture after the begin keyword

BSCAN_VIRTEX_INSTANCE_NAME : BSCAN_VIRTEX
      port map  (DRCK1  => user_DRCK1,
                 DRCK2  => user_DRCK2,
                 RESET  => user_RESET,
                 SEL1   => user_SEL1,
                 SEL2   => user_SEL2,
                 SHIFT  => user_SHIFT,
                 TDI    => user_TDI,
                 UPDATE => user_UPDATE,
                 TD01   => user_TD01,
                 TD02   => user_TD02);
```

Verilog Instantiation Template

```
BSCAN_VIRTEX instance_name (.DRCK1(user_DRCK1),
                            .DRCK2(user_DRCK2),
                            .RESET(user_RESET),
                            .SEL1 (user_SEL1),
                            .SEL2 (user_SEL2),
                            .SHIFT(user_SHIFT),
                            .TDI  (user_TDI),
                            .UPDATE(user_UPDATE),
                            .TD01(user_TD01),
                            .TD02(user_TD02));
```

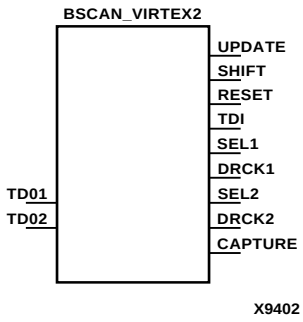
Commonly Used Constraints

None

BSCAN_VIRTEX2

Virtex-II Boundary Scan Logic Control Circuit

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



BSCAN_VIRTEX2 provides access to the BSCAN sites on a Virtex-II or Virtex-II Pro device. It is used to create internal boundary scan chains. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Virtex-II and Virtex-II Pro. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_VIRTEX2 symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

Note For Virtex and Virtex-E, see BSCAN_VIRTEX.

A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active.The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, SHIFT, and CAPTURE pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Note For specific information on boundary scan for an architecture, see *The Programmable Logic Data Book.*

Usage

This design element is supported for instantiation and schematics but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BSCAN_VIRTEX2 should be placed
-- after architecture statement but before begin keyword

component BSCAN_VIRTEX2
   port (CAPTURE: out STD_ULOGIC;
         DRCK1  : out STD_ULOGIC;
         DRCK2  : out STD_ULOGIC;
         RESET  : out STD_ULOGIC;
         SEL1   : out STD_ULOGIC;
         SEL2   : out STD_ULOGIC;
         SHIFT  : out STD_ULOGIC;
         TDI    : out STD_ULOGIC;
         UPDATE : out STD_ULOGIC;
         TD01   : in STD_ULOGIC;
         TD02   : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for BSCAN_VIRTEX2
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BSCAN_VIRTEX2 should be
-- placed in architecture after the begin keyword

BSCAN_VIRTEX2_INSTANCE_NAME : BSCAN_VIRTEX2
      port map  (CAPTURE=> user_CAPTURE,
                 DRCK1  => user_DRCK1,
                 DRCK2  => user_DRCK2,
                 RESET  => user_RESET,
                 SEL1   => user_SEL1,
                 SEL2   => user_SEL2,
                 SHIFT  => user_SHIFT,
                 TDI    => user_TDI,
                 UPDATE => user_UPDATE,
                 TD01   => user_TD01,
                 TD02   => user_TD02);
```

Verilog Instantiation Template

```
BSCAN_VIRTEX2 instance_name  (.CAPTURE (user_CAPTURE),
                              .DRCK1 (user_DRCK1),
                              .DRCK2 (user_DRCK2),
                              .RESET (user_RESET),
                              .SEL1 (user_SEL1),
                              .SEL2 (user_SEL2),
                              .SHIFT(user_SHIFT),
                              .TDI  (user_TDI),
                              .UPDATE (user_UPDATE),
                              .TD01 (user_TD01),
                              .TD02 (user_TD02));
```

Commonly Used Constraints

None

BUF

General-Purpose Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

X9444

BUF is a general purpose, non-inverting buffer.

In Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, BUF is usually not necessary and is removed by the partitioning software (MAP).

In XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, BUF is usually removed, unless you inhibit optimization by applying the OPT=OFF attribute to the BUF symbol or by using the LOGIC_OPT=OFF global attribute.

Usage

This design is supported in schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUF should be placed
-- after architecture statement but before begin keyword

component BUF
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUF
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUF should be placed
-- in architecture after the begin keyword

BUF_INSTANCE_NAME : BUF
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
BUF instance_name (.O (user_O),
                   .I (user_I));
```

Commonly Used Constraints

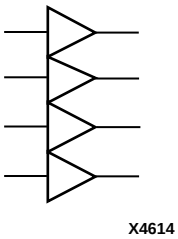
None

BUF4, 8, 16

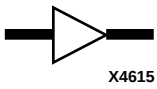
General-Purpose Buffers

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro

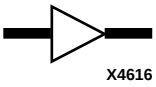
BUF4



BUF8

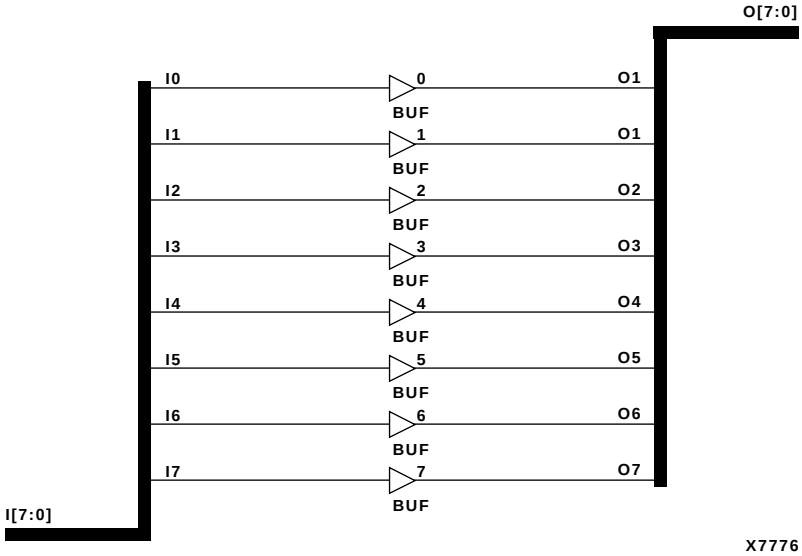


BUF16



BUF4, 8, 16 are general purpose, non-inverting buffers.

In XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, BUF4, BUF8, and BUF16 are usually removed, unless you inhibit optimization by applying the OPT=OFF attribute to the BUF4, BUF8, or BUF16 symbol or by using the LOGIC_OPT=OFF global attribute.



BUF8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

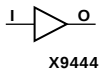
Usage

These design elements are schematic only.

BUFCF

Fast Connect Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



X9444

BUFCF is a single fast connect buffer used to connect the outputs of the LUTs and some dedicated logic directly to the input of another LUT. Using this buffer implies CLB packing. No more than four LUTs may be connected together as a group.

Usage

This design element is supported for schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFCF should be placed
-- after architecture statement but before begin keyword

component BUFCF
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFCF
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFCF should be placed
-- in architecture after the begin keyword

BUFCF_INSTANCE_NAME : BUFCF
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
BUFCF instance_name (.O (user_O),
                     .I (user_I));
```

Commonly Used Constraints

None

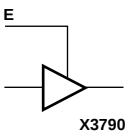
BUFE, 4, 8, 16

Internal 3-State Buffers with Active High Enable

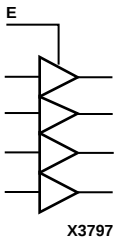
Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
BUFE	Primitive	Primitive	Primitive	Primitive*	Primitive	N/A
BUFE4, BUFE8, BUFE16	Macro	Macro	Macro	Macro*	Macro	N/A

* not supported for XC9500XL and XC9500XV devices

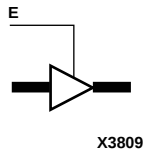
BUFE



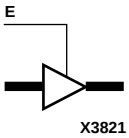
BUFE4



BUFE8



BUFE16



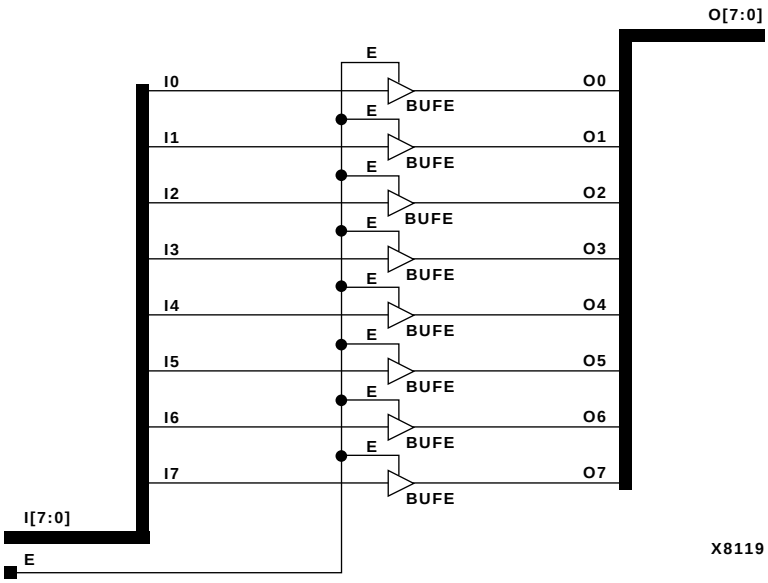
BUFE, BUFE4, BUFE8, and BUFE16 are single or multiple 3-state buffers with inputs I, I3 – I0, I7 – I0, and I15 – I0, respectively; outputs O, O3 – O0, O7 – O0, and O15 – O0, respectively; and active-High output enable (E). When E is High, data on the inputs of the buffers is transferred to the corresponding outputs. When E is Low, the output is high impedance (Z state or Off). The outputs of the buffers are connected to horizontal longlines in FPGA architectures.

The outputs of separate BUFE symbols can be tied together to form a bus or a multiplexer. Make sure that only one E is High at any one time. If none of the E inputs is active-High, a "weak-keeper" circuit (Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro) keeps the output bus from floating but does not guarantee that the bus remains at the last value driven onto it.

For XC9500/XVdevices, BUFE output nets assume the High logic level when all connected BUFE/BUFT buffers are disabled. On-chip 3-state multiplexing is not available in XC9500XL devices.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, BUFE elements need a PULLUP element connected to their output. NGDBuild inserts a PULLUP element if one is not connected.

Inputs		Outputs
E	I	O
0	X	Z
1	1	1
1	0	0



BUFE8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

These design elements are supported for schematic, inference, and instantiation.

VHDL Inference Code

```
architecture Behavioral of bufe is
begin
process (I, E)
begin
 if (E = '1') then
  O <= I;
 else
  O <= 'Z';
 end if;
end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (I or E)
 begin
  if (E)
   O <= I;
  else
   O <= 1'bZ;
 end
```

VHDL Instantiation Template

```
-- Component Declaration for BUFE should be placed
-- after architecture statement but before begin keyword

component BUFE
   port (O  : out STD_ULOGIC;
         E  : in STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFE should be placed
-- in architecture after the begin keyword

BUFE_INSTANCE_NAME : BUFE
      port map  (O => user_O,
                 E => user_E,
                 I => user_I);
```

Verilog Instantiation Template

```
BUFE instance_name (.O (user_O),
                    .E (user_E),
                    .I (user_I));
```

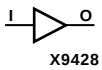
Commonly Used Constraints

None

BUFG

Global Clock Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive



BUFG, an architecture-independent global buffer, distributes high fan-out clock signals throughout a PLD device. The Xilinx implementation software converts each BUFG to an appropriate type of global buffer for the target PLD device. To use a specific type of buffer, instantiate it manually.

To use a BUFG in a schematic, connect the input of the BUFG symbol to the clock source. Depending on the target PLD family, the clock source can be an external PAD symbol, an IBUF symbol, or internal logic. For a negative-edge clock input, insert an INV (inverter) symbol between the BUFG output and the clock input. The inversion is implemented at the Configurable Logic Block (CLB) or Input Output Block (IOB) clock pin.

XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II

Consult the device data sheet for the number of available global pins. For these architectures BUFG is always implemented using an IOB. Connect the input of BUFG to an IPAD or an IOPAD that represents an external signal source. Each BUFG can drive any number of register clocks in a design. The output of a BUFG may also be used as an ordinary input signal to other logic elsewhere in the design.

Virtex, Virtex-E, Spartan-II, Spartan-IIE

In Virtex, Virtex-E, Spartan-II, and Spartan-IIE, the BUFG cannot be driven directly from a pad. It can be driven from an IBUFG to indicate to use the dedicated pin (GCLKIOB pin) or from an internal driver to create an internal clock. BUFG can also be driven with an IBUF to represent an externally driven clock that does not use the dedicated pin.

Virtex-II, Virtex-II Pro

Virtex-II and Virtex-II Pro clock buffers are multiplexed clock buffers. In Virtex-II and Virtex-II Pro, a BUFG is implemented using a BUFGMUX with the S_B input tied high, basically meaning the S input is tied low. I1 is unused. I0 is used.

Usage

This design element is supported for schematic and instantiation. Synthesis tools usually infer a BUFGP on any clock net. If there are more clock nets than BUFGPs, the synthesis tool usually instantiates BUFGPs for the clocks that are most utilized. The BUFGP contains both a BUFG and an IBUFG.

VHDL Instantiation Template

```
-- Component Declaration for BUFG should be placed
-- after architecture statement but before begin keyword

component BUFG
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFG
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFG should be placed
-- in architecture after the begin keyword

BUFG_INSTANCE_NAME : BUFG
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
BUFG instance_name (.O (user_O),
                    .I (user_I));
```

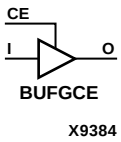
Commonly Used Constraints

LOC

BUFGCE

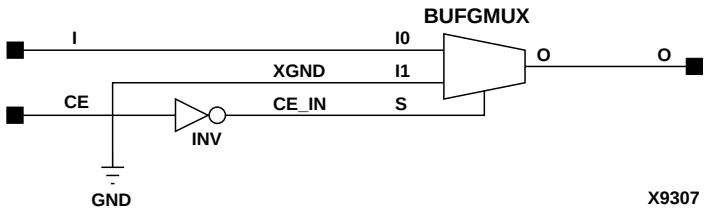
Global Clock MUX Buffer with Clock Enable and Output State 0

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



BUFGCE is a multiplexed global clock buffer with a single gated input. Its O output is "0" when clock enable (CE) is Low (inactive). When clock enable (CE) is High, the I input is transferred to the O output.

Inputs		Outputs
I	CE	O
X	0	0
I	1	I



BUFGCE Implementation Virtex-II, Virtex-II Pro

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGCE should be placed
-- after architecture statement but before begin keyword

component BUFGCE
   port (O  : out STD_ULOGIC;
         CE : in STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGCE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here
```

```
-- Component Instantiation for BUFGCE should be placed
-- in architecture after the begin keyword

BUFGCE_INSTANCE_NAME : BUFGCE
      port map  (O => user_O,
                 CE => user_CE,
                 I => user_I);
```

Verilog Instantiation Template

```
BUFGCE instance_name (.O (user_O),
                      .CE (user_CE),
                      .I (user_I));
```

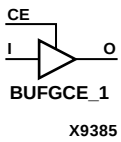
Commonly Used Constraints

LOC

BUFGCE_1

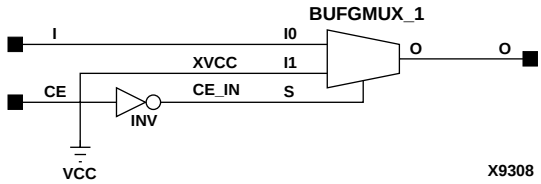
Global Clock MUX Buffer with Clock Enable and Output State 1

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



BUFGCE_1 is a multiplexed global clock buffer with a single gated input. Its O output is High (1) when clock enable (CE) is Low (inactive). When clock enable (CE) is High, the I input is transferred to the O output.

Inputs		Outputs
I	CE	O
X	0	1
I	1	I



BUFGCE_1 Implementation Virtex-II, Virtex-II Pro

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGCE_1 should be placed
-- after architecture statement but before begin keyword

component BUFGCE_1
   port (O  : out STD_ULOGIC;
         CE : in STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGCE_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here
```

```
-- Component Instantiation for BUFGCE_1 should be placed
-- in architecture after the begin keyword

BUFGCE_1_INSTANCE_NAME : BUFGCE_1
      port map  (O => user_O,
                 CE => user_CE,
                 I => user_I);
```

Verilog Instantiation Template

```
BUGCE_1 instance_name (.O (user_O),
                       .CE (user_CE),
                       .I (user_I));
```

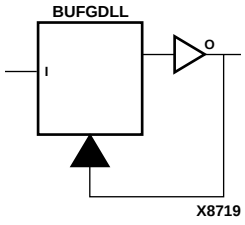
Commonly Used Constraints

LOC

BUFGDLL

Clock Delay Locked Loop Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



BUFGDLL is a special purpose clock delay locked loop buffer for clock skew management. It is provided as a user convenience for the most frequently used configuration of elements for clock skew management. Internally, it consists of an IBUFG driving the CLKIN pin of a CLKDLL followed by a BUFG that is driven by the CLK0 pin of the CLKDLL. Because BUFGDLL already contains an input buffer (IBUFG), it can only be driven by a top-level port (IPAD).

Any DUTY_CYCLE_CORRECTION attribute on a BUFGDLL applies to the underlying CLKDLL symbol.

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGDLL should be placed
-- after architecture statement but before begin keyword

component BUFGDLL
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGDLL
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFGDLL should be placed
-- in architecture after the begin keyword

BUFGDLL_INSTANCE_NAME : BUFGDLL

      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
BUFGDLL instance_name (.O (user_O),
                       .I (user_I));
```

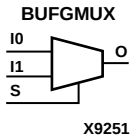
Commonly Used Constraints

STARTUP_WAIT

BUFGMUX

Global Clock MUX Buffer with Output State 0

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A



BUFGMUX is a multiplexed global clock buffer that can select between two input clocks I0 and I1. When the select input (S) is Low, the signal on I0 is selected for output (O). When the select input (S) is High, the signal on I1 is selected for output.

BUFGMUX and BUFGMUX_1 are distinguished by which state the output assumes when it switches between clocks in response to a change in its select input. BUGFMUX0 assumes output state 0 and BUFGMUX_1 assumes output state 1.

Using a BUFGMUX element in your design may cause inaccurate simulation if all the following conditions occur: both clock inputs (I0 and I1) are used, GSR is activated during simulation (after simulation time ‘0’), and the secondary clock input (I1) is selected before or while GSR is active. In this case, the primary clock input (I0) is incorrectly selected. This occurs because there is a cross-coupled register pair that ensures the BUFGMUX output does not inadvertently generate a clock edge. When GSR is asserted, these registers initialize to the default state of I0. To select the secondary clock, you must send a clock pulse to both the primary and secondary clock inputs while GSR is inactive.

Note BUFGMUX guarantees that when S is toggled, the state of the output will remain in the inactive state until the next active clock edge (either I0 or I1) occurs.

Inputs			Outputs
I0	I1	S	O
I0	X	0	I0
X	I1	1	I1
X	X	↑	0
X	X	↓	0

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGMUX should be placed
-- after architecture statement but before begin keyword

component BUFGMUX
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGMUX
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFGMUX should be placed
-- in architecture after the begin keyword

BUFGMUX_INSTANCE_NAME : BUFGMUX
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

Verilog Instantiation Template

```
BUFGMUX instance_name (.O (user_O),
                       .I0 (user_I0),
                       .I1 (user_I1),
                       .S  (user_S));
```

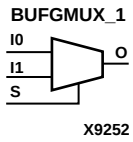
Commonly Used Constraints

LOC

BUFGMUX_1

Global Clock MUX Buffer with Output State 1

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A



BUFGMUX_1 is a multiplexed global clock buffer that can select between two input clocks I0 and I1. When the select input (S) is Low, the signal on I0 is selected for output (O). When the select input (S) is High, the signal on I1 is selected for output.

BUFGMUX and BUFGMUX_1 are distinguished by which state the output assumes when it switches between clocks in response to a change in its select input. BUFGMUX assumes output state 0 and BUFGMUX_1 assumes output state 1.

Using a BUFGMUX_1 element in your design may cause inaccurate simulation if all the following conditions occur: both clock inputs (I0 and I1) are used, GSR is activated during simulation (after simulation time ‘0’), and the secondary clock input (I1) is selected before or while GSR is active. In this case, the primary clock input (I0) is incorrectly selected. This occurs because there is a cross-coupled register pair that ensures the BUFGMUX_1 output does not inadvertently generate a clock edge. When GSR is asserted, these registers initialize to the default state of I0. To select the secondary clock, you must send a clock pulse to both the primary and secondary clock inputs while GSR is inactive.

Inputs			Outputs
I0	I1	S	O
I0	X	0	I0
X	I1	1	I1
X	X	↑	1
X	X	↓	1

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGMUX_1 should be placed
-- after architecture statement but before begin keyword

component BUFGMUX_1

   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGMUX_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFGMUX should be placed
-- in architecture after the begin keyword

BUFGMUX_1_INSTANCE_NAME : BUFGMUX_1

      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

Verilog Instantiation Template

```
BUFGMUX_1 instance_name (.O (user_O),
                         .I0 (user_I0),
                         .I1 (user_I1),
                         .S  (user_S));
```

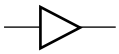
Commonly Used Constraints

LOC

BUFGP

Primary Global Buffer for Driving Clocks or Longlines (Four per PLD Device)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



BUFGP, a primary global buffer, is used to distribute high fan-out clock or control signals throughout PLD devices.

In Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, BUFGP is equivalent to an IBUFG driving a BUFG.

In XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, BUFGP is treated like BUFG.

A BUFGP provides direct access to Configurable Logic Block (CLB) and Input Output Block (IOB) clock pins and limited access to other CLB inputs. The input to a BUFGP comes only from a dedicated IOB.

Because of its structure, a BUFGP can always access a clock pin directly. However, it can access only one of the F3, G1, C3, or C1 pins, depending on the corner in which the BUFGP is placed. When the required pin cannot be accessed directly from the vertical line, PAR feeds the signal through another CLB and uses general purpose routing to access the load pin.

To use a BUFGP in a schematic, connect the input of the BUFGP element directly to the PAD symbol. Do not use any IBUFs, because the signal comes directly from a dedicated IOB. The output of the BUFGP is then used throughout the schematic. For a negative-edge clock, insert an INV (inverter) element between the output of the BUFGP and the clock input. This inversion is performed inside each CLB or IOB.

A Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, or Virtex-II Pro BUFGP must be sourced by an external signal. Other BUFGPs can be sourced by an internal signal, but PAR must use the dedicated IOB to drive the BUFGP, which means that the IOB is not available for use by other signals. If possible, use a BUFGS instead, because it can be sourced internally without using an IOB.

Usage

This design element is supported for schematic and instantiation. Synthesis tools usually infer a BUFGP on any clock net. If there are more clock nets than BUFGPs, the synthesis tool usually instantiates BUFGPs for the clocks that are most utilized.

VHDL Instantiation Template

```
-- Component Declaration for BUFGP should be placed
-- after architecture statement but before begin keyword

component BUFGP
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGP
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFGP should be placed
-- in architecture after the begin keyword

BUFGP_INSTANCE_NAME : BUFGP

      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
BUFGP instance_name (.O (user_O),
                     .I (user_I));
```

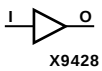
Commonly Used Constraints

HBLKNM

BUFGSR

Global Set/Reset Input Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	Primitive	Primitive	Primitive



BUFGSR distributes global set/reset signals throughout selected flip-flops of an XC9500/XV/XL, CoolRunner XPLA3, or CoolRunner-II device. Global Set/Reset (GSR) control pins are available on these CPLD devices. Consult device data sheets for availability.

BUFGSR always acts as an input buffer. To use it in a schematic, connect the input of the BUFGSR symbol to an IPAD or an IOPAD representing the GSR signal source. GSR signals generated on-chip must be passed through an OBUF-type buffer before they are connected to BUFGSR.

For global set/reset control, the output of BUFGSR normally connects to the CLR or PRE input of a flip-flop symbol, like FDCP, or any registered symbol with asynchronous clear or preset. The global set/reset control signal may pass through an inverter to perform an active-low set/reset. The output of BUFGSR may also be used as an ordinary input signal to other logic elsewhere in the design. Each BUFGSR can control any number of flip-flops in a design.

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGSR should be placed
-- after architecture statement but before begin keyword

component BUFGSR
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGSR
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFGSR should be placed
-- in architecture after the begin keyword

BUFGSR_INSTANCE_NAME : BUFGSR
      port map  (O => user_O,
                 I => user_I);
```

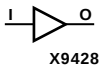
Verilog Instantiation Template

```
BUFGSR instance_name(.O (user_O),
                     .I (user_I));
```

BUFGTS

Global 3-State Input Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	Primitive	Primitive	Primitive



BUFGTS distributes global output-enable signals throughout the output pad drivers of an XC9500/XV/XL, CoolRunner XPLA3, or CoolRunner-II device. Global Three-State (GTS) control pins are available on these CPLD devices. Consult device data sheets for availability.

BUFGTS always acts as an input buffer. To use it in a schematic, connect the input of the BUFGTS symbol to an IPAD or an IOPAD representing the GTS signal source. GTS signals generated on-chip must be passed through an OBUF-type buffer before they are connected to BUFGTS.

For global 3-state control, the output of BUFGTS normally connects to the E input of a 3-state output buffer symbol, OBUFE. The global 3-state control signal may pass through an inverter or control an OBUFT symbol to perform an active-low output-enable. The same 3-state control signal may even be used both inverted and non-inverted to enable alternate groups of device outputs. The output of BUFGTS may also be used as an ordinary input signal to other logic elsewhere in the design. Each BUFGTS can control any number of output buffers in a design.

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGTS should be placed
-- after architecture statement but before begin keyword

component BUFGTS
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGTS
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BUFGTS should be placed
-- in architecture after the begin keyword

BUFGTS_INSTANCE_NAME : BUFGTS
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
BUFGTS instance_name(.O (user_O),
                     .I (user_I));
```

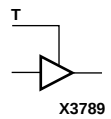
BUFT, 4, 8, 16

Internal 3-State Buffers with Active-Low Enable

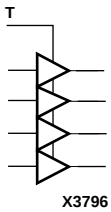
Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
BUFT	Primitive	Primitive	Primitive	Primitive*	Primitive	N/A
BUFT4, BUFT8, BUFT16	Macro	Macro	Macro	Macro*	Macro	N/A

* not supported for XC9500XL and XC9500XV devices

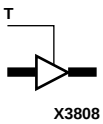
BUFT



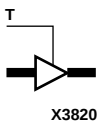
BUFT4



BUFT8



BUFT16



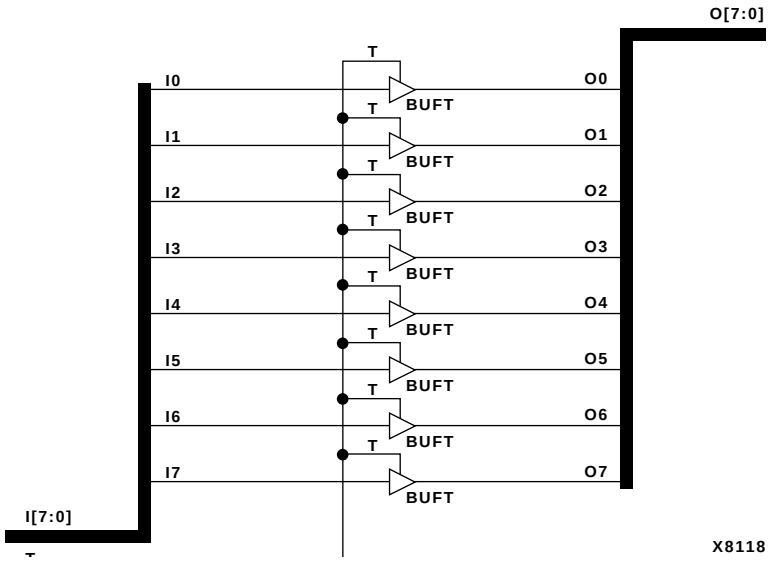
BUFT, BUFT4, BUFT8, and BUFT16 are single or multiple 3-state buffers with inputs I, I3 – I0, I7 – I0, and I15 – 10, respectively; outputs O, O3 – O0, O7 – O0, and O15 – O0, respectively; and active-Low output enable (T). When T is Low, data on the inputs of the buffers is transferred to the corresponding outputs. When T is High, the output is high impedance (Z state or off). The outputs of the buffers are connected to horizontal longlines in FPGA architectures.

The outputs of separate BUFT symbols can be tied together to form a bus or a multiplexer. Make sure that only one T is Low at one time. If none of the T inputs is active (Low), a "weak-keeper" circuit (Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro) prevents the output bus from floating but does not guarantee that the bus remains at the last value driven onto it.

For XC9500/XV/XL devices, BUFT output nets assume the High logic level when all connected BUFE/BUFT buffers are disabled. On-chip 3-state multiplexing is not available in XC9500XL devices.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, when all BUFTs on a net are disabled, the net is High. For correct simulation of this effect, a PULLUP element must be connected to the net. NGDBuild inserts a PULLUP element if one is not connected so that back-annotation simulation reflects the true state of the device.

Inputs		Outputs
T	I	O
1	X	Z
0	1	1
0	0	0



BUFT8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

These design elements are supported for schematics, instantiations, or inferences.

VHDL Inference Code

```
architecture Behavioral of buft is
begin
process (I, T)
begin
 if (T = '0') then
  O <= I;
 else
  O <= 'Z';
 end if;
end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (I or T)
 begin
  if (!T)
   O <= I;
  else
   O <= 1'bZ;
 end
```

VHDL Instantiation Template

```
-- Component Declaration for BUFT should be placed
-- after architecture statement but before begin keyword

component BUFT
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFT
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for BUFT should be placed
-- in architecture after the begin keyword

BUFT_INSTANCE_NAME : BUFT
      port map  (O => user_O,
                 I => user_I,
                 T => user_T);
```

Verilog Instantiation Template

```
BUFT instance_name (.O (user_O),
                    .I (user_I),
                    .T (user_T));
```

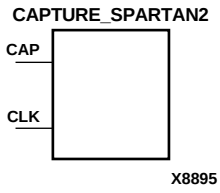
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, LOC, U_SET

CAPTURE_SPARTAN2

Spartan-II Register State Capture for Bitstream Readback

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	N/A	N/A	N/A	N/A	N/A



CAPTURE_SPARTAN2 provides user control over when to capture register (flip-flop and latch) information for readback. Spartan-II and Spartan-IIE devices provide the readback function through dedicated configuration port instructions, instead of with a READBACK component as in other FPGA architectures. The CAPTURE_SPARTAN2 symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Note Spartan-II and Spartan-IIE devices only allow for capturing register (flip-flop and latch) states. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured.

An asserted High CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition. The Low-to-High clock transition triggers the capture clock (CLK) which clocks out the readback data.

By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, add the ONESHOT attribute to CAPTURE_SPARTAN2. See the *Constraints Guide* for information on the ONESHOT attribute.

Note For details on the Spartan-II and Spartan-IIE readback functions, see *The Programmable Logic Data Book*.

Usage

This design element is supported for schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for CAPTURE_SPARTAN2 should be
-- placed after architecture statement but before
-- begin keyword

component CAPTURE_SPARTAN2
   port (CAP: in STD_ULOGIC;
         CLK: in STD_ULOGIC);
end component;

-- Component Attribute specification for CAPTURE_SPARTAN2
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for CAPTURE_SPARTAN2 should be
-- placed in architecture after the begin keyword

CAPTURE_SPARTAN2_INSTANCE_NAME : CAPTURE_SPARTAN2
      port map  (CAP => user_CAP,
                 CLK => user_CLK);
```

Verilog Instantiation Template

```
CAPTURE_SPARTAN2 instance_name (.CAP (user_CAP),
                                .CLK (user_CLK));
```

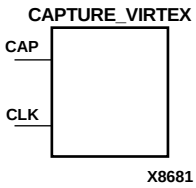
Commonly Used Constraints

ONESHOT

CAPTURE_VIRTEX

Virtex Register State Capture for Bitstream Readback

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	Primitive	N/A	N/A	N/A	N/A



CAPTURE_VIRTEX provides user control over when to capture register (flip-flop and latch) information for readback. Virtex and Virtex-E devices provide the readback function through dedicated configuration port instructions, instead of with a READBACK component as in other FPGA architectures.

Note For Virtex-II and Virtex-II Pro, see CAPTURE_VIRTEX2.

The CAPTURE_VIRTEX symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Note Virtex and Virtex-E allow for capturing register (flip-flop and latch) states only. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured.

An asserted High CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition. The Low-to-High clock transition triggers the capture clock (CLK) which clocks out the readback data.

By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, add the ONESHOT attribute to CAPTURE_VIRTEX. See the *Constraints Guide* for information on the ONESHOT attribute.

For details on the Virtex and Virtex-E readback functions, see the Virtex datasheets on the Xilinx web site, http://support.xilinx.com.

Usage

This design element is supported for schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for CAPTURE_VIRTEX should be
-- placed after architecture statement but before
-- begin keyword

component CAPTURE_VIRTEX
   port (CAP: in STD_ULOGIC;
         CLK: in STD_ULOGIC);
end component;

-- Component Attribute specification for CAPTURE_VIRTEX
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here
```

```
-- Component Instantiation for CAPTURE_VIRTEX should be
-- placed in architecture after the begin keyword

CAPTURE_VIRTEX_INSTANCE_NAME : CAPTURE_VIRTEX
      port map  (CAP => user_CAP,
                 CLK => user_CLK);
```

Verilog Instantiation Template

```
CAPTURE_VIRTEX instance_name (.CAP (user_CAP),
                              .CLK (user_CLK));
```

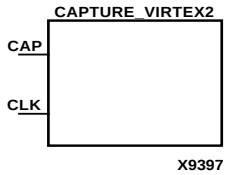
Commonly Used Constraints

ONESHOT

CAPTURE_VIRTEX2

Virtex-II Register State Capture for Bitstream Readback

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



CAPTURE_VIRTEX2 provides user control over when to capture register (flip-flop and latch) information for readback. Virtex-II and Virtex-II Pro devices provide the readback function through dedicated configuration port instructions, instead of with a READBACK component as in other FPGA architectures.

Note For Virtex and Virtex-E, see CAPTURE_VIRTEX.

The CAPTURE_VIRTEX2 symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Virtex-II and Virtex-II Pro allow for capturing register (flip-flop and latch) states only. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured.

An asserted high CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition. The Low-to-High clock transition triggers the capture clock (CLK) which clocks out the readback data.

By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, add the ONESHOT attribute to CAPTURE_VIRTEX2. See the *Constraints Guide* for information on the ONESHOT attribute.

For details on the Virtex-II and Virtex-II Pro readback functions, see *The Programmable Logic Data Book*.

Usage

This design element is supported for schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for CAPTURE_VIRTEX2 should be
-- placed after architecture statement but before
-- begin keyword

component CAPTURE_VIRTEX2
   port (CAP: in STD_ULOGIC;
         CLK: in STD_ULOGIC);
end component;

-- Component Attribute specification for CAPTURE_VIRTEX2
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here
```

```
-- Component Instantiation for CAPTURE_VIRTEX2 should be
-- placed in architecture after the begin keyword

CAPTURE_VIRTEX2_INSTANCE_NAME : CAPTURE_VIRTEX2
      port map  (CAP => user_CAP,
                 CLK => user_CLK);
```

Verilog Instantiation Template

```
CAPTURE_VIRTEX2 instance_name (.CAP (user_CAP),
                               .CLK (user_CLK));
```

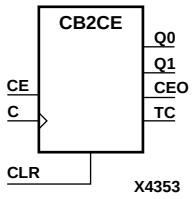
Commonly Used Constraints

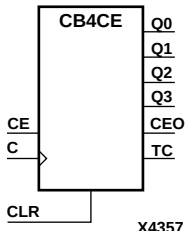
None

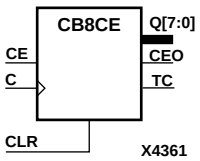
CB2CE, CB4CE, CB8CE, CB16CE

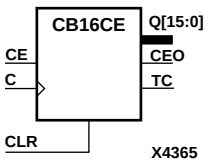
2-, 4-, 8-,16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro









CB2CE, CB4CE, CB8CE, and CB16CE are, respectively, 2-, 4-, 8-, and 16-bit (stage), asynchronous, clearable, cascadable binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, outputs Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

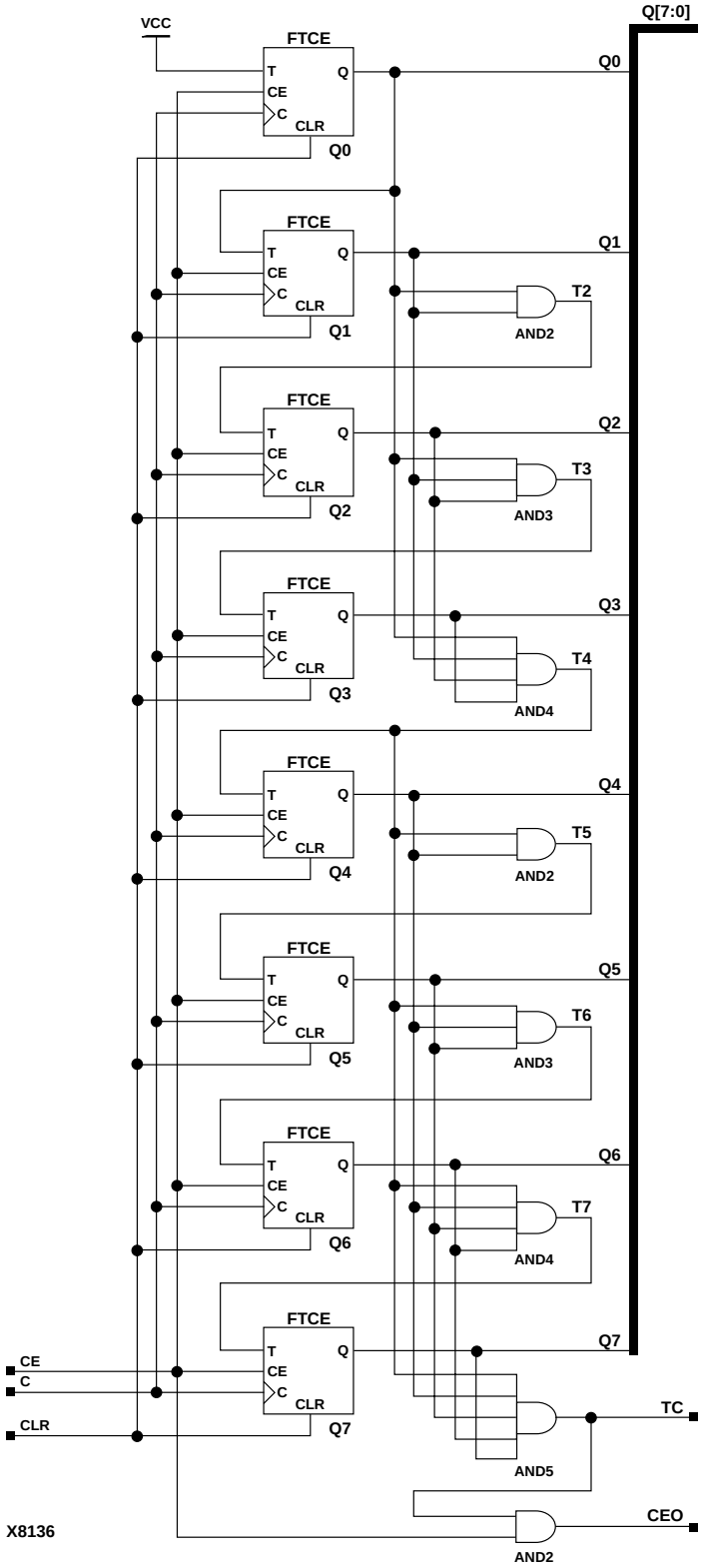
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs		
CLR	CE	C	Qz-Q0	TC	CEO
1	X	X	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

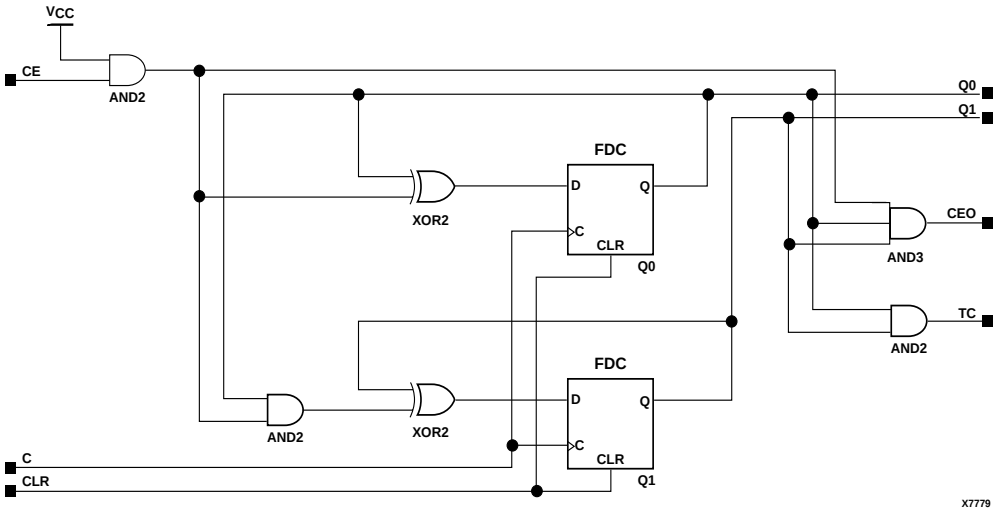
z= 1 for CB2CE; z = 3 for CB4CE; z = 7 for CB8CE; z = 15 for CB16CE

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0

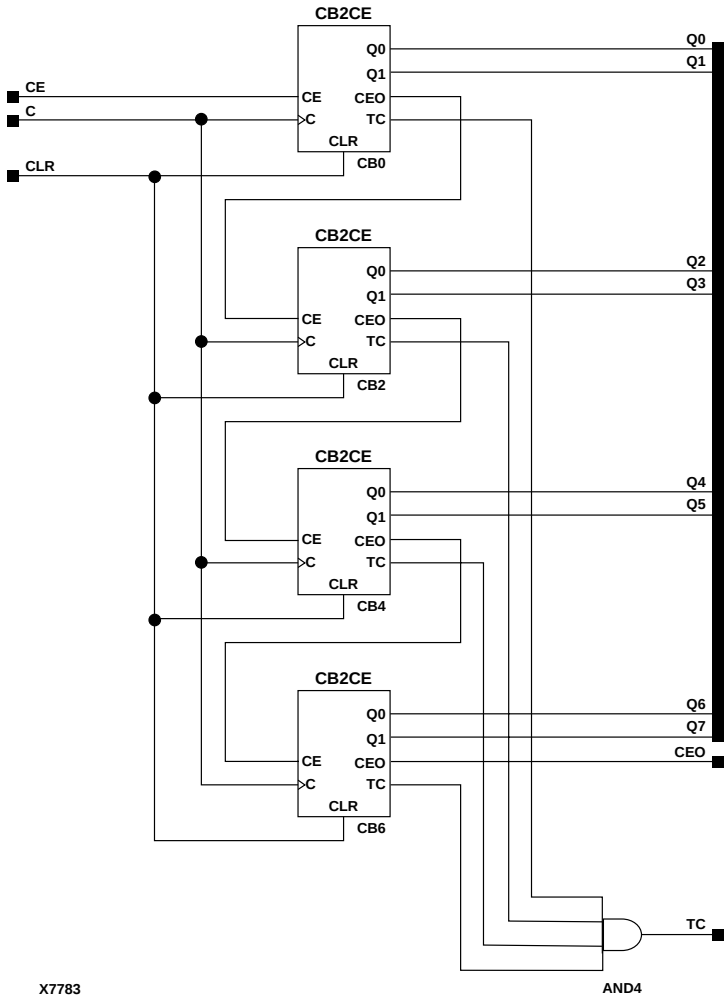
CEO = TC • CE



CB8CE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CB2CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



CB8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cb2ce is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C, CLR)
begin
 if (CLR='1') then
   Q <= (others => '0');
 elsif (C'event and C='1') then
  if (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process (Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

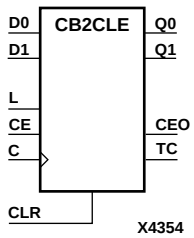
Verilog Inference Code

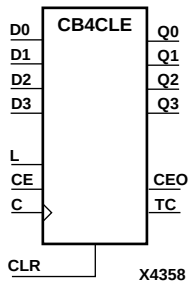
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
   Q <= 0;
 else if (CE)
   Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (CE or TC)
begin
   CE0 <= TC && CE;
end
```

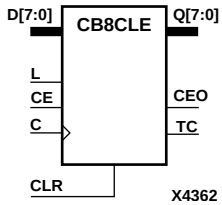
CB2CLE, CB4CLE, CB8CLE, CB16CLE

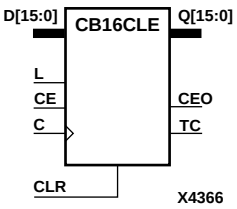
2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro









CB2CLE, CB4CLE, CB8CLE, and CB16CLE are, respectively, 2-, 4-, 8-, and 16-bit (stage) synchronously loadable, asynchronously clearable, cascadable binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock transition, independent of the state of clock enable (CE). The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C, L, and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

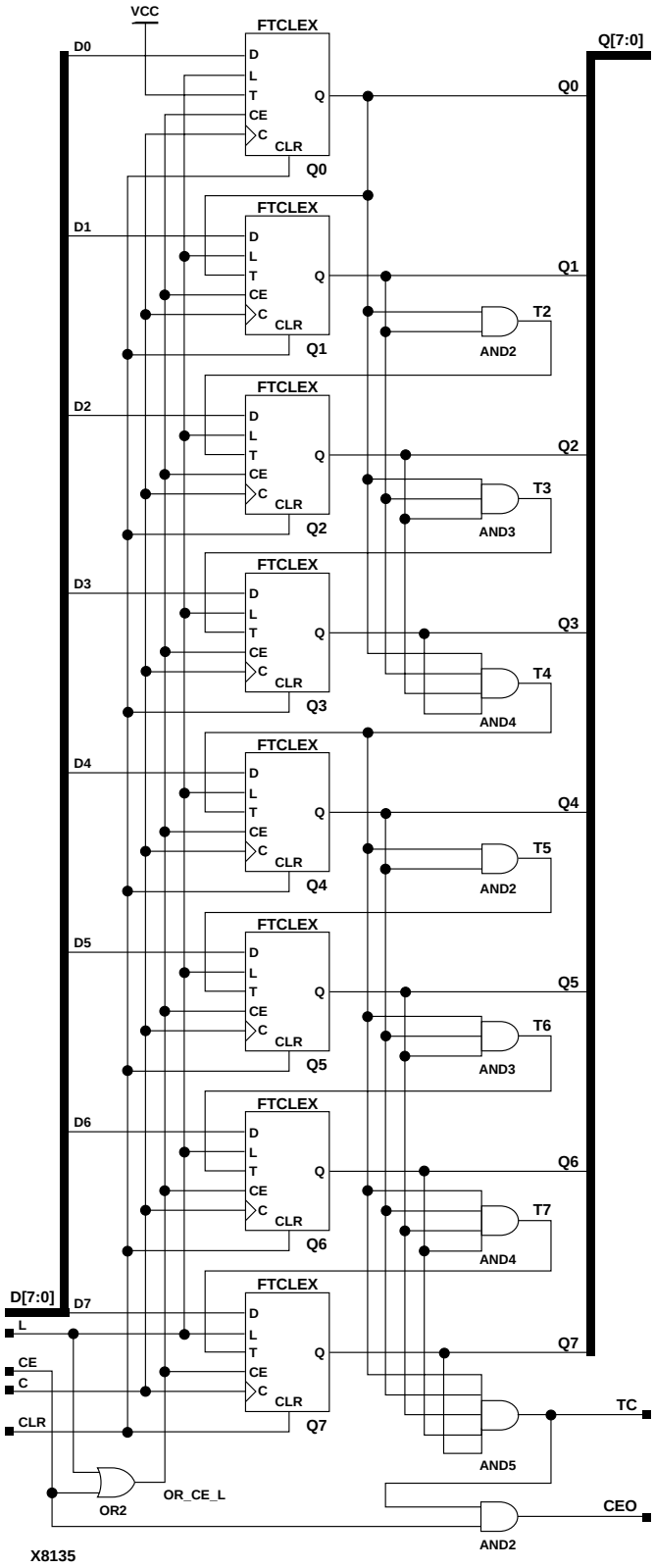
Inputs					Outputs		
CLR	**L**	**CE**	**C**	**Dz – D0**	**Qz – Q0**	**TC**	**CEO**
1	X	X	X	X	0	0	0
0	1	X	↑	Dn	dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

z= 1 for CB2CLE; z = 3 for CB4CLE; z = 7 for CB8CLE; z = 15 for CB16CLE

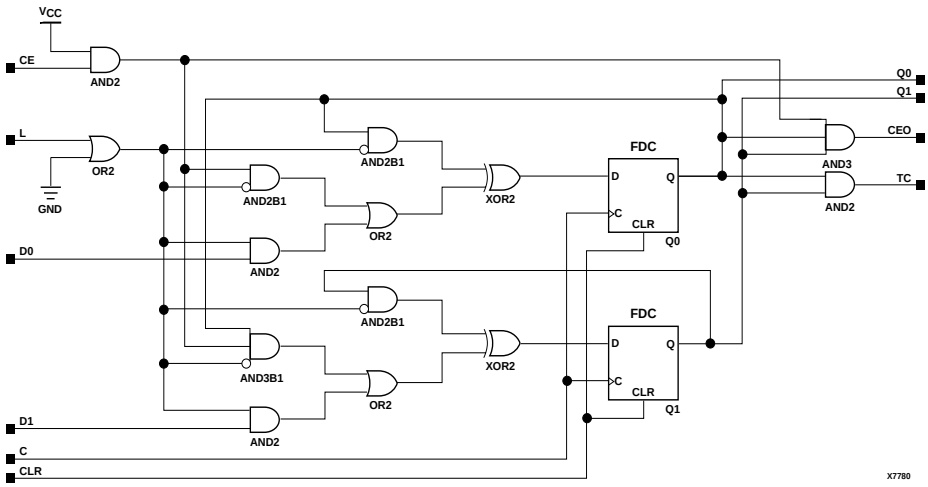
dn = state of referenced input (Dn) one setup time prior to active clock transition.

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0

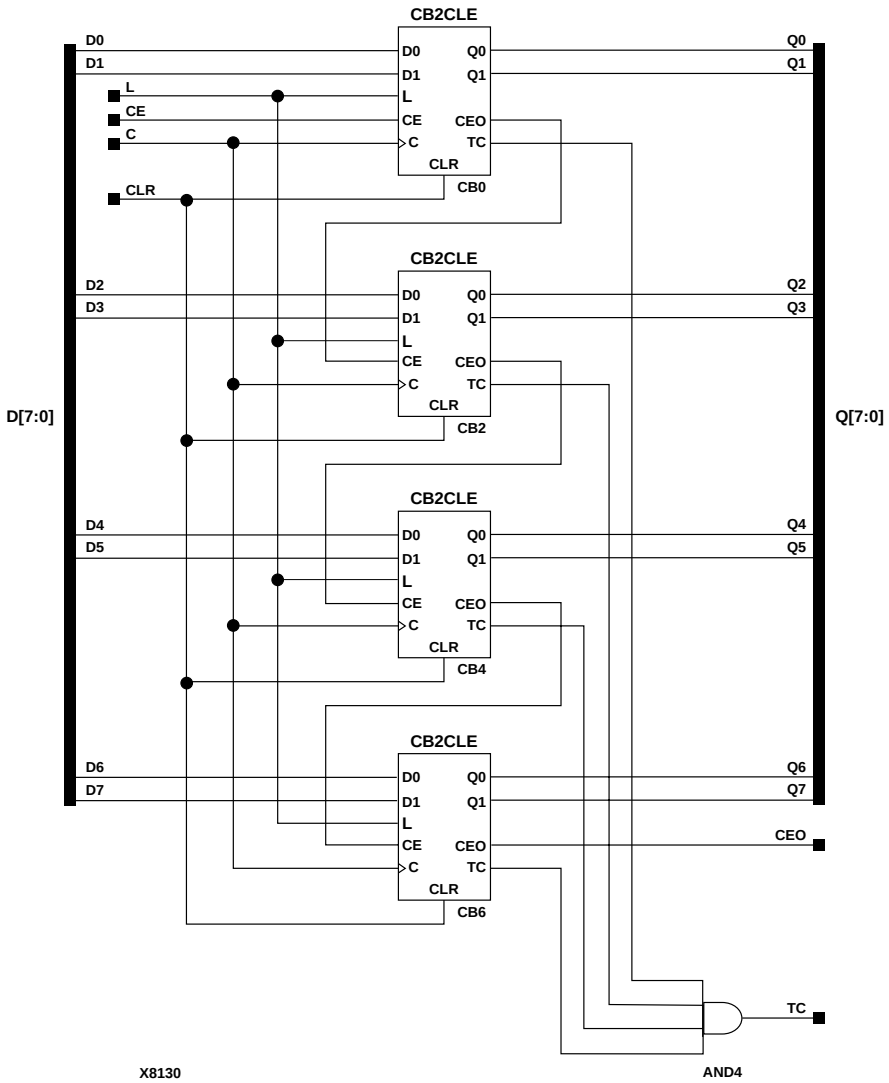
CEO = TC • CE



CB8CLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CB2CLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



CB8CLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cb2cle is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
```

```
 elsif (C'event and C='1') then
  if (L = '1') then
   Q <= D;
  elsif (CE='1') then
   Q <= COUNT+1;
  end if;
 end if;
end process;

process(Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

TC <= TC_INT;
CEO <= TC and CE;

end Behavioral;
```

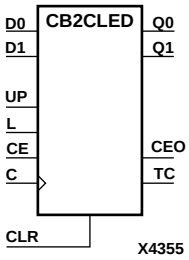
Verilog Inference Code

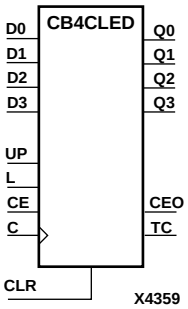
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
    Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

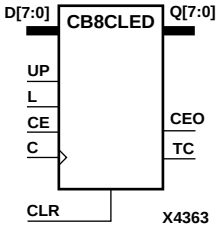
CB2CLED, CB4CLED, CB8CLED, CB16CLED

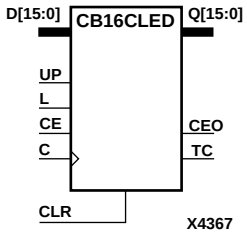
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro









CB2CLED, CB4CLED, CB8CLED, and CB16CLED are, respectively, 2-, 4-, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.

For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the CEO output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage.

When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not. For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, see the "CB2X1, CB4X1, CB8X1, CB16X1" section for high-performance cascadable, bidirectional counters.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted with an inverter in front of the GSR input of STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2.

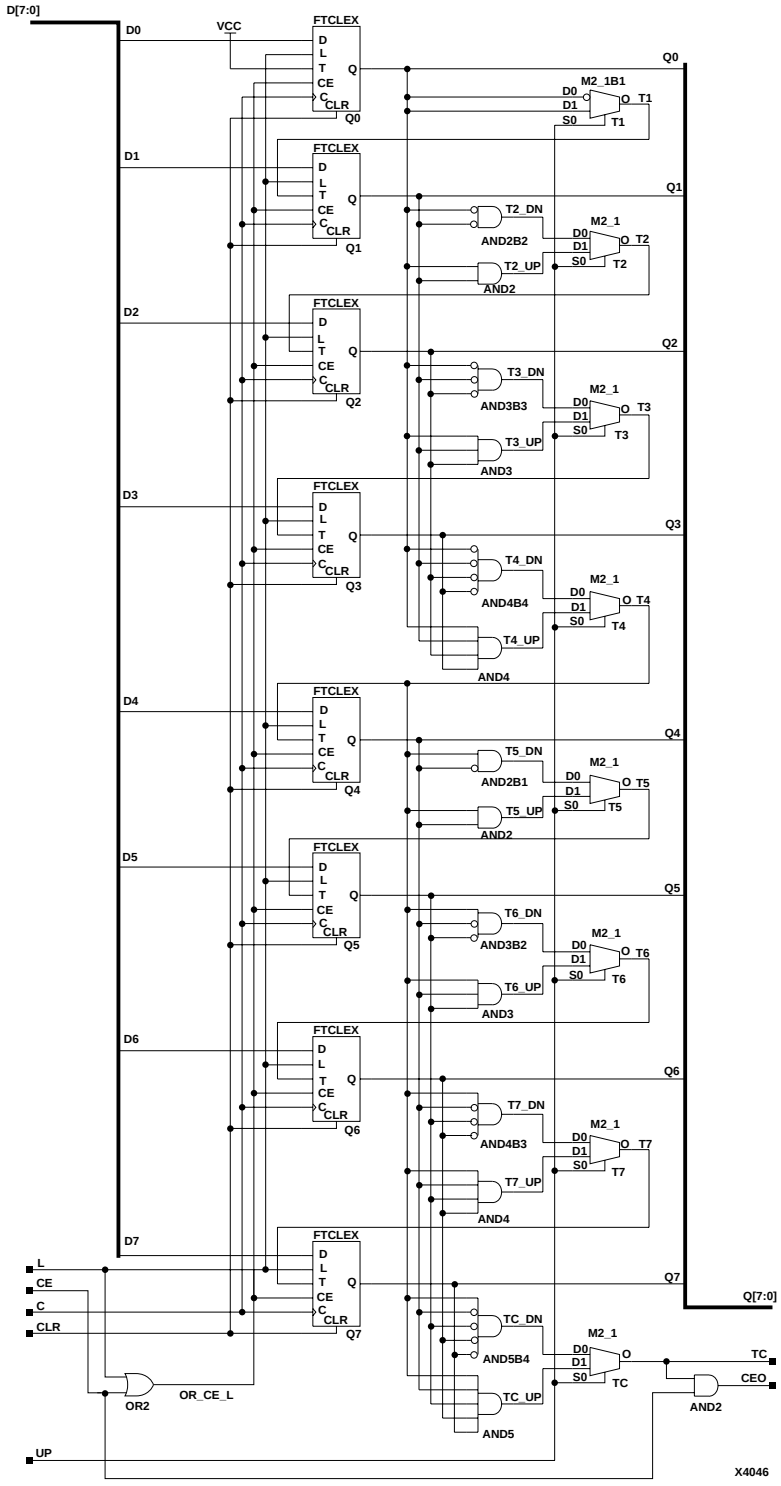
Inputs						Outputs		
CLR	**L**	**CE**	**C**	**UP**	**Dz – D0**	**Qz – Q0**	**TC**	**CEO**
1	X	X	X	X	X	0	↑	↑*CE
0	1	X	↑	X	Dn	dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO

z = 1 for CB2CLED; z = 3 for CB4CLED; z = 7 for CB8CLED; z = 15 for CB16CLED

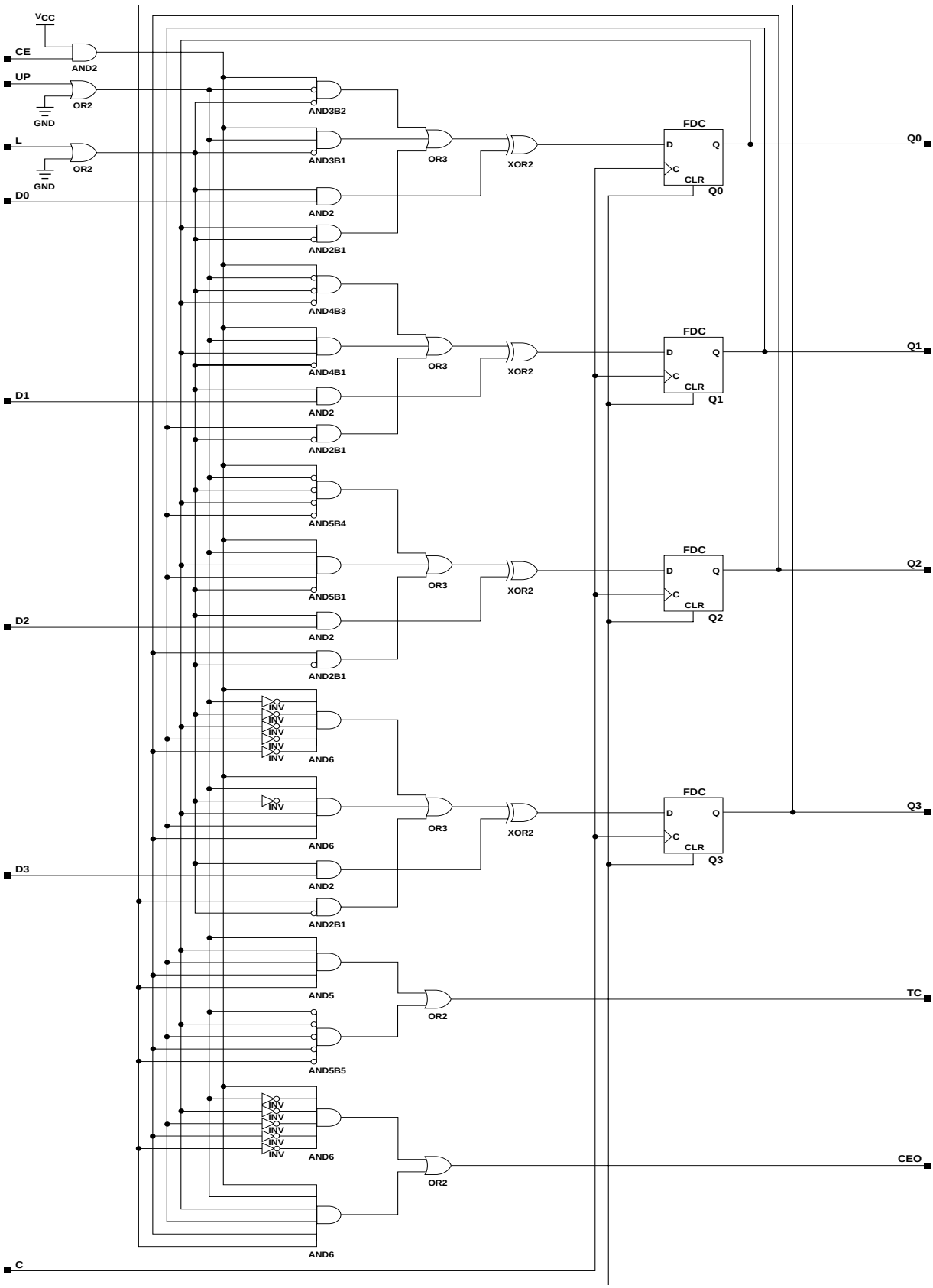
dn = state of referenced input (Dn), one setup time prior to active clock transition

TC = (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP) + (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP)

CEO = TC • CE



CB8CLED Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CB4CLED Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cb2cled is

 constant TERMINAL_COUNT_UP : std_logic_vector(WIDTH-1 downto
   0) := (others => '1');
 constant TERMINAL_COUNT_DOWN : std_logic_vector(WIDTH-1 downto
   0) := (others => '0');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
 elsif C'event and C='1' then
  if (L = '1') then
   Q <= D;
  elsif (CE='1') then
   if (UP='1') then
    Q <= Q+1;
   elsif (UP='0') then
    Q <= Q-1;
   end if;
  end if;
 end if;
end process;

process(Q, UP)
begin
 if (((Q = TERMINAL_COUNT_UP) and (UP = '1')) or
  ((Q = TERMINAL_COUNT_DOWN) and (UP = '0'))) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

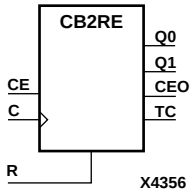
Verilog Inference Code

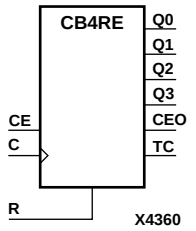
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  begin
   if (UP)
      Q <= Q + 1;
   else if (!UP)
      Q <= Q - 1;
  end
end

always @ (Q or UP)
begin
 if ((Q == TERMINAL_COUNT_UP && UP) || (Q == TERMINAL_COUNT_DOWN
   && !UP))
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

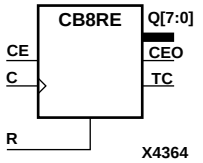
CB2RE, CB4RE, CB8RE, CB16RE

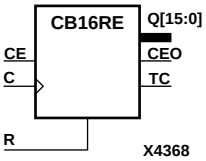
2-, 4-, 8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro









CB2RE, CB4RE, CB8RE, and CB16RE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, resettable, cascadable binary counters. The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero during the Low-to-High clock transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when both Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and R inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

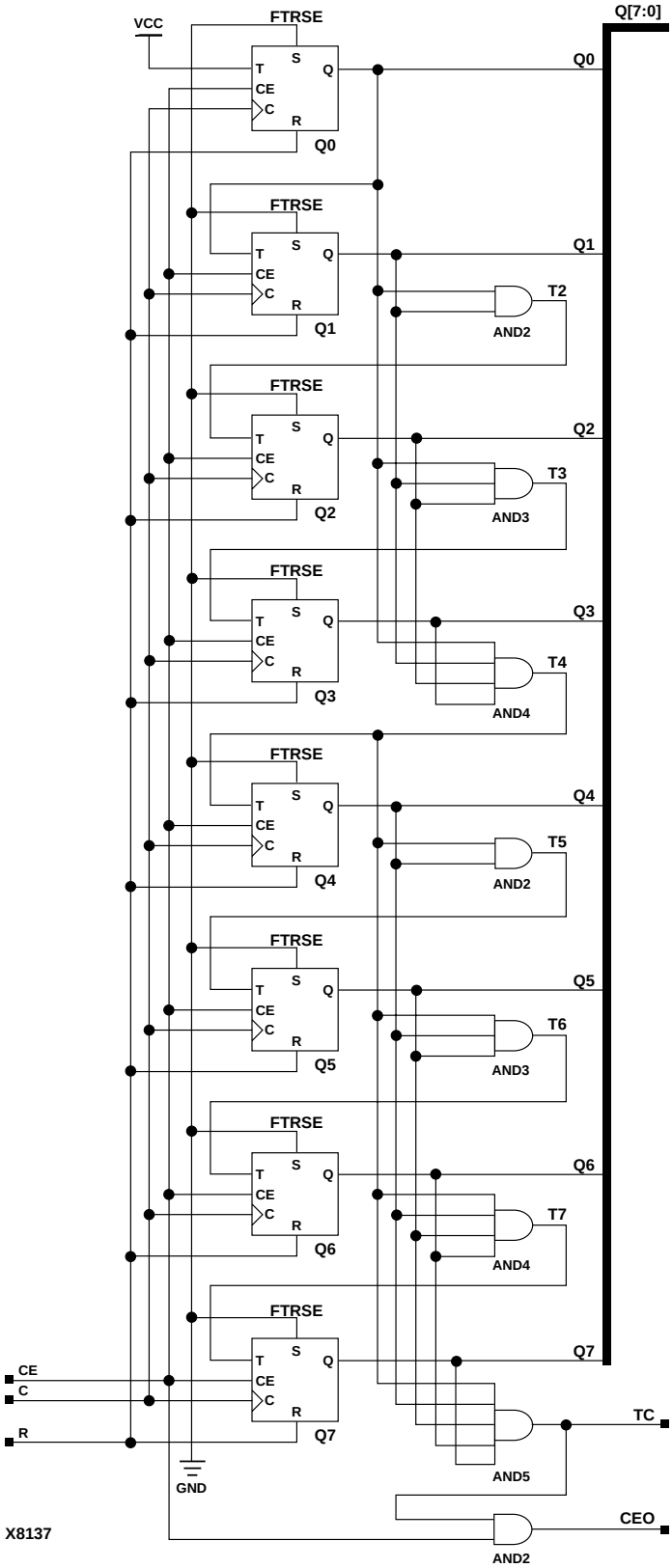
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
1	X	↑	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

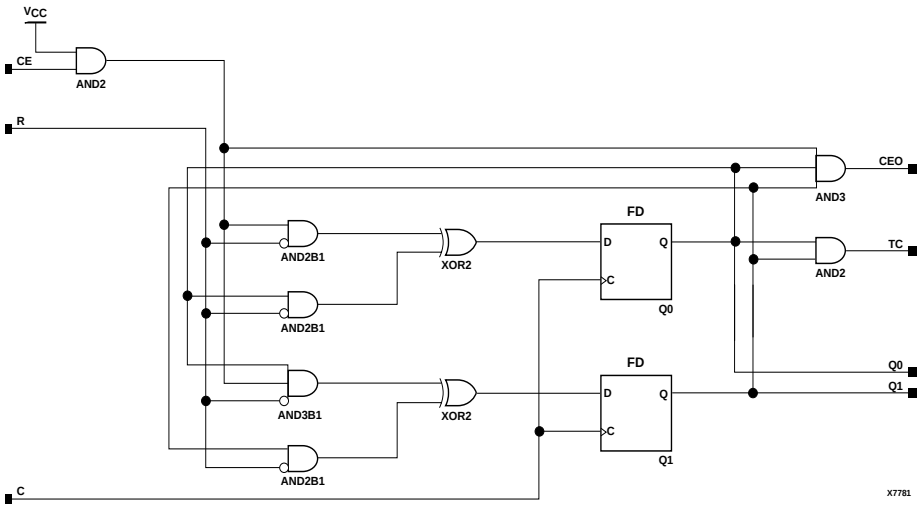
z = 1 for CB2RE; z = 3 for CB4RE; z = 7 for CB8RE; z = 15 for CB16RE

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0)

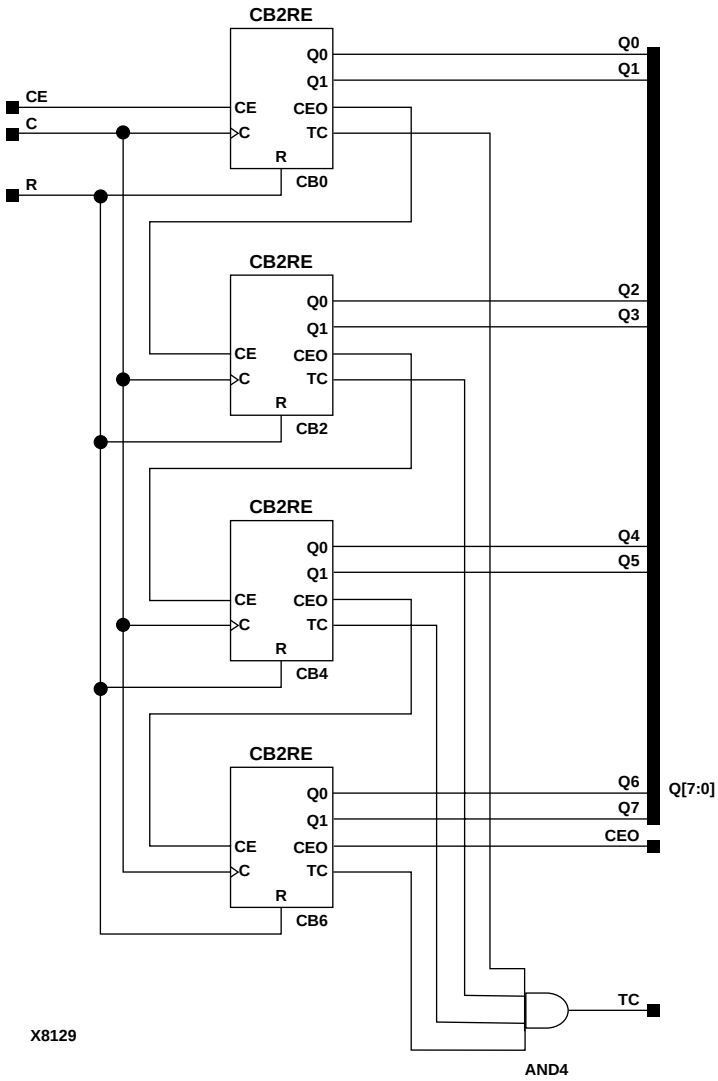
CEO = TC • CE



CB8RE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CB2RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



CB8RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cb2re is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C)
begin
 if (C'event and C='1') then
  if (R='1') then
   Q <= (others => '0');
  elsif (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process(Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

Verilog Inference Code

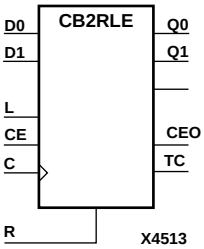
```
always @ (posedge C)
begin
 if (R)
  Q <= 0;
 else if (CE)
    Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end
```

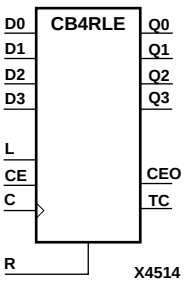
```
always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

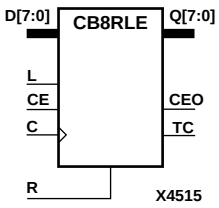
CB2RLE, CB4RLE, CB8RLE, CB16RLE

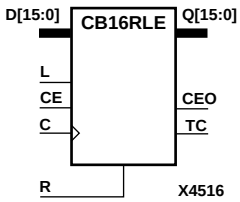
2-, 4-, 8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro









CB2RLE, CB4RLE, CB8RLE, and CB16RLE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, cascadable binary counters. The synchronous reset (R) is the highest priority input. The synchronous R, when High, overrides all other inputs and resets the Q outputs, terminal count (TC), and clock enable out (CEO) outputs to Low on the Low-to-High clock (C) transition.

The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of CE. The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High. The CEO output is High when all Q outputs and CE are High to allow direct cascading of counters.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and by connecting the C, L, and R inputs in parallel. The maximum length of the counter is determined by the accumulated CE-to-CEO propagation delays versus the clock period. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

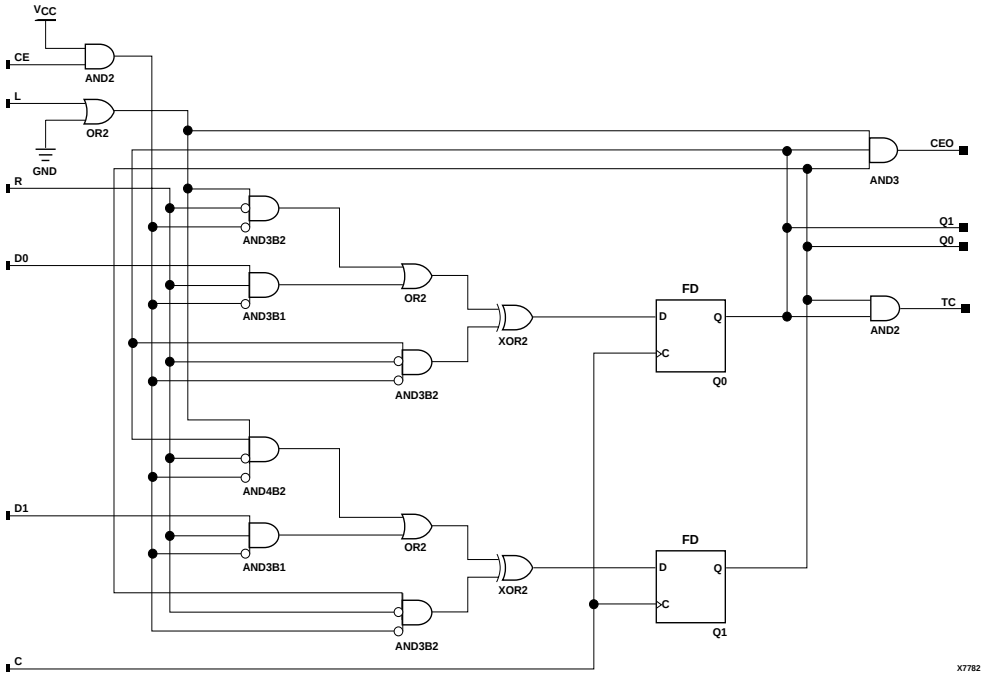
Inputs					Outputs		
R	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	↑	X	0	0	0
0	1	X	↑	Dn	dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

z = 1 for CB2RLE; z = 3 for CB4RLE; z = 7 for CB8RLE; z = 15 for CB16RLE

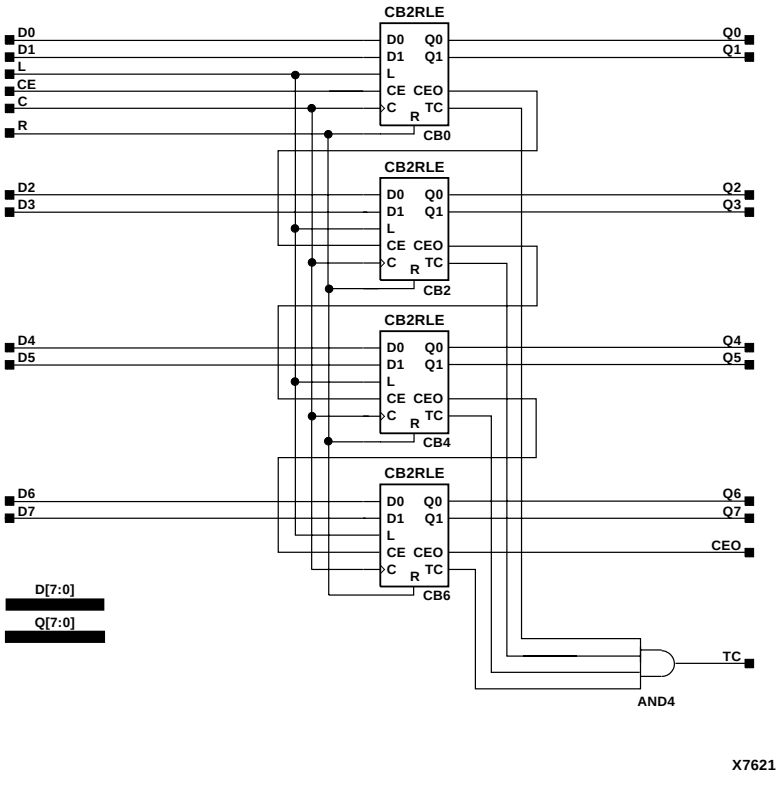
dn = state of referenced input (Dn) one setup time prior to active clock transition

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEO = TC • CE



CB2RLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



CB8RLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cb2rle is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C)
begin
 if (C'event and C='1') then
  if (R='1') then
   Q <= (others => '0');
  elsif (L='1') then
   Q <= D;
  elsif (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process(Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

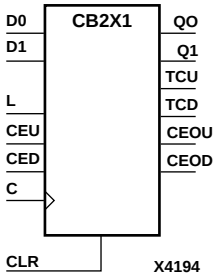
Verilog Inference Code

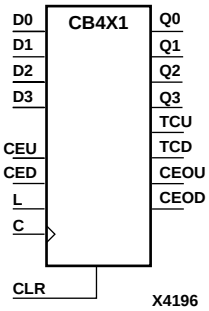
```
always @ (posedge C)
begin
 if (R)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
    Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

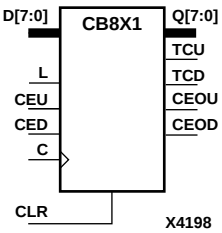
CB2X1, CB4X1, CB8X1, CB16X1

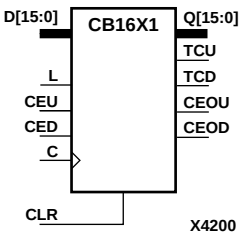
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro









CB2X1, CB4X1, CB8X1, and CB16X1 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronously loadable, asynchronously clearable, bidirectional binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, clock enable outputs CEOU and CEOD go to Low and High, respectively, independent of clock transitions. The data on the D inputs loads into the counter on the Low-to-High clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

The Q outputs increment when CEU is High, provided CLR and L are Low, during the Low-to-High clock transition. The Q outputs decrement when CED is High, provided CLR and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are connected directly to the CEU and CED inputs, respectively, of the next stage. The clock, L, and CLR inputs are connected in parallel.

The maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
CLR	**L**	**CEU**	**CED**	**C**	**Dz–D0**	**Qz–Q0**	**TCU**	**TCD**	**CEOU**	**CEOD**
1	X	X	X	X	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid

z = 1 for CB2X1; z = 3 for CB4X1; z = 7 for CB8X1; z = 15 for CB16X1

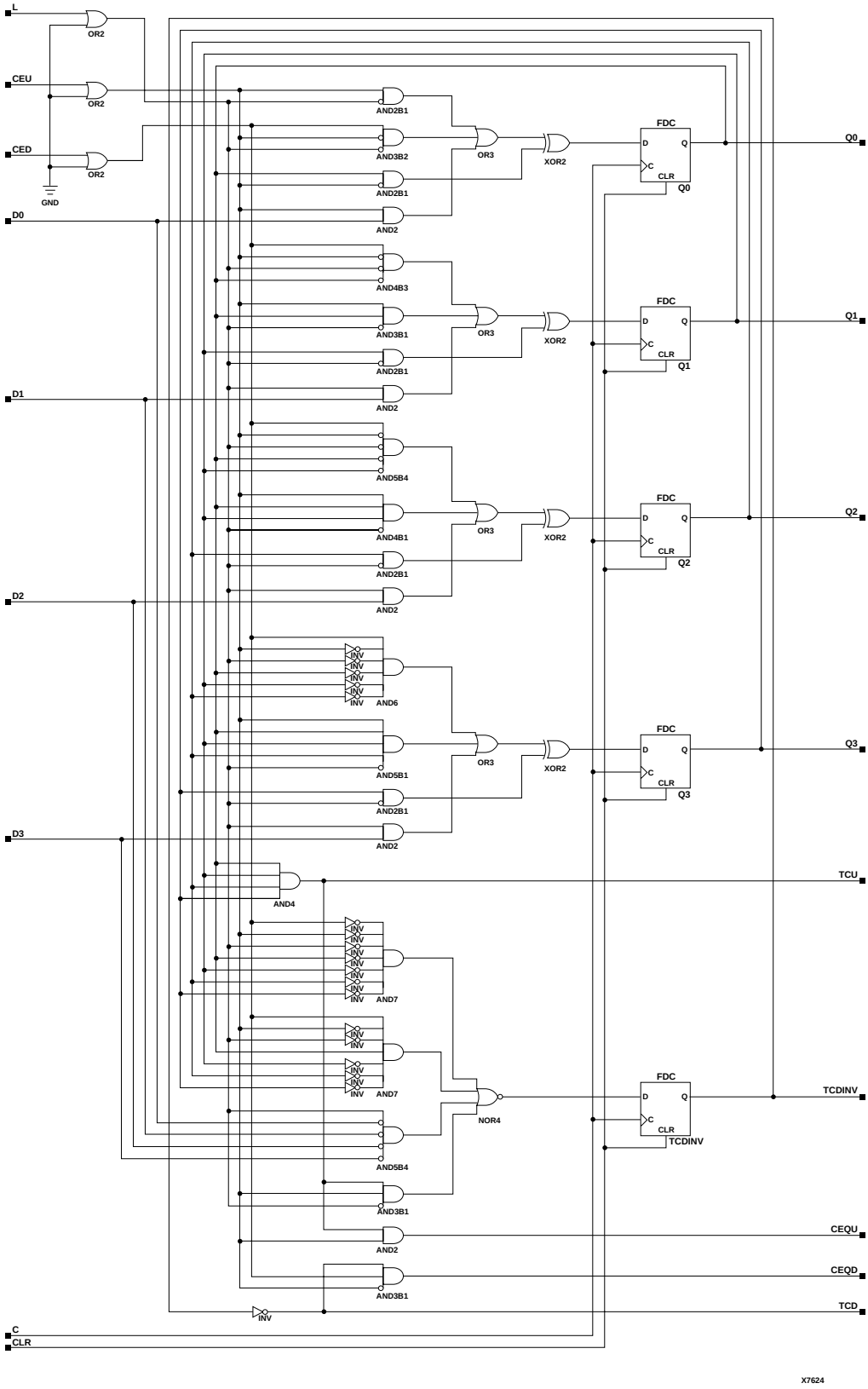
dn = state of referenced input (Dn) one setup time prior to active clock transition

TCU = Qz • Q(z-1) • Q(z-2) • ... • Q0

TCD = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEOU = TCU • CEU

CEOD = TCD • CED



CB4X1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cb2x1 is

   signal CEU_INT : std_logic;
   signal CED_INT : std_logic;

   constant TERMINAL_COUNT_UP : std_logic_vector(WIDTH-1 downto
   0) := (others => '1');
   constant TERMINAL_COUNT_DOWN : std_logic_vector(WIDTH-1
   downto 0) := (others => '0');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
  CEU_INT <= '0';
  CED_INT <= '1';
 elsif ((C'event) and (C='1')) then
  if (L = '1') then
   Q <= D;
   CEU_INT <= CEU;
   CED_INT <= CED;
  elsif (CEU='1') then
   Q <= Q+1;
   CEU_INT <= '1';
   CED_INT <= '0';
  elsif (CED='1') then
   Q <= Q-1;
   CEU_INT <= '0';
   CED_INT <= '1';
  end if;
 end if;
end process;

process (Q, CEU_INT, CED_INT)
begin
 if ((Q = TERMINAL_COUNT_UP) and (CEU_INT = '1')) then
  TCU <= '1';
  TCD <= '0';
 elsif ((Q = TERMINAL_COUNT_DOWN) and (CED_INT = '1')) then
  TCU <= '0';
  TCD <= '1';
 else
  TCU <= '0';
  TCD <= '0';
 end if;
end process;
```

```
CEOU <= TCU and CEU;
CEOD <= TCD and CED;

end Behavioral;
```

Verilog Inference Code

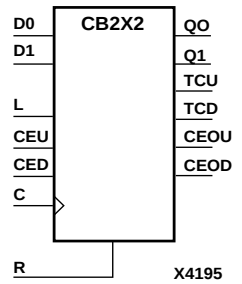
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
 begin
  Q <= 0;
  CEU_INT <= 1'b0;
  CED_INT <= 1'b1;
 end
 else if (L)
 begin
  Q <= D;
  CEU_INT <= CEU;
  CED_INT <= CED;
 end
 else if (CEU)
 begin
   Q <= Q + 1;
   CEU_INT <= 1;
   CED_INT <= 0;
 end
   else if (CED)
 begin
  Q <= Q - 1;
  CEU_INT <= 1'b0;
  CED_INT <= 1'b1;
 end
end

always @ (Q or CEU_INT or CED_INT)
begin
 if (Q == TERMINAL_COUNT_UP && CEU_INT)
 begin
 TCU_INT <= 1;
 TCD_INT <= 0;
 end
 else if ((Q == TERMINAL_COUNT_DOWN) && CED_INT)
 begin
 TCU_INT <= 0;
 TCD_INT <= 1;
   end
 else
 begin
 TCU_INT <= 0;
 TCD_INT <= 0;
 end
end
```

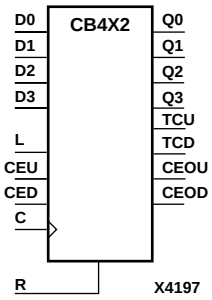
```
always @ (TCU_INT or CEU or TCD_INT or CED)
begin
CEOU <= TCU_INT && CEU;
CEOD <= TCD_INT && CED;
TCU <= TCU_INT;
TCD <= TCD_INT;
end
```

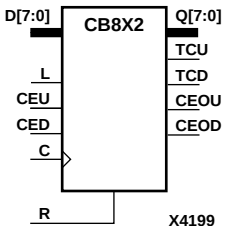
CB2X2, CB4X2, CB8X2, CB16X2

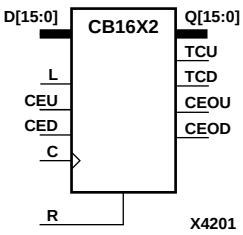
2-, 4-, 8-, and 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro









CB2X2, CB4X2, CB8X2, and CB16X2 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, bidirectional binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in CPLD architectures.

The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, and clock enable outputs CEOU and CEOD go to Low and High, respectively, on the Low-to-High clock (C) transition. The data on the D inputs loads into the counter on the Low-to-High clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

All Q outputs increment when CEU is High, provided R and L are Low during the Low-to-High clock transition. All Q outputs decrement when CED is High, provided R and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are, respectively, connected directly to the CEU and CED inputs of the next stage. The C, L, and R inputs are connected in parallel.

The maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
R	**L**	**CEU**	**CED**	**C**	**Dz – D0**	**Qz – Q0**	**TCU**	**TCD**	**CEOU**	**CEOD**
1	X	X	X	↑	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid

z = 1 for CB2X2; z = 3 for CB4X2; z = 7 for CB8X2; z = 15 for CB16X2

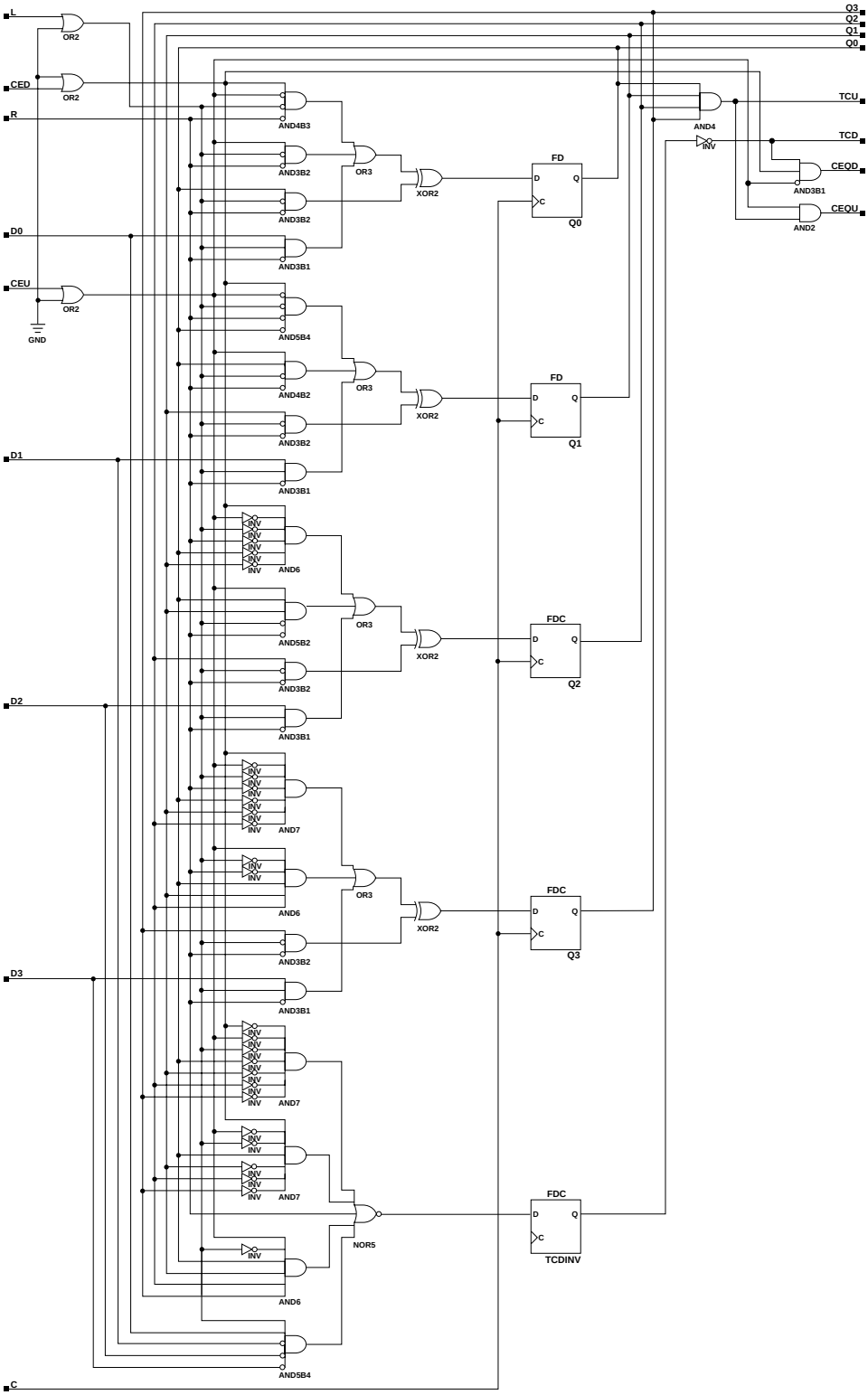
dn = state of referenced input (Dn) one setup time prior to active clock transition

TCU = Qz • Q(z-1) • Q(z-2) • ... • Q0

TCD = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEOU = TCU • CEU

CEOD = TCD • CED



CB4X2 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cb2x2 is

   signal CEU_INT : std_logic;
   signal CED_INT : std_logic;

   constant TERMINAL_COUNT_UP : std_logic_vector(WIDTH-1 downto
   0) := (others => '1');
   constant TERMINAL_COUNT_DOWN : std_logic_vector(WIDTH-1
   downto 0) := (others => '0');

begin

process(C)
begin
 if ((C'event) and (C='1'))then
  if (R='1') then
   Q <= (others => '0');
   CEU_INT <= '0';
   CED_INT <= '1';
  elsif (L = '1') then
   Q <= D;
   CEU_INT <= CEU;
   CED_INT <= CED;
  elsif (CEU='1') then
   Q <= Q+1;
   CEU_INT <= '1';
   CED_INT <= '0';
  elsif (CED='1') then
   Q <= Q-1;
   CEU_INT <= '0';
   CED_INT <= '1';
  end if;
 end if;
end process;

process(Q, CEU_INT, CED_INT)
begin
 if ((Q = TERMINAL_COUNT_UP) and (CEU_INT = '1')) then
  TCU <= '1';
  TCD <= '0';
 elsif ((Q = TERMINAL_COUNT_DOWN) and (CED_INT = '1')) then
  TCU <= '0';
  TCD <= '1';
 else
  TCU <= '0';
  TCD <= '0';
 end if;
end process;
```

```
CEOU<=TCU and CEU;
CEOD<=TCD and CED;

end Behavioral;
```

Verilog Inference Code

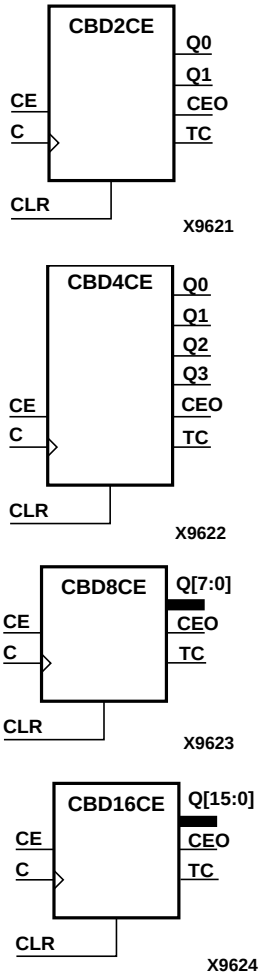
```
always @ (posedge C)
begin
 if (R)
 begin
  Q <= 0;
  CEU_INT <= 1'b0;
  CED_INT <= 1'b1;
 end
 else if (L)
 begin
  Q <= D;
  CEU_INT <= CEU;
  CED_INT <= CED;
 end
 else if (CEU)
  begin
   Q <= Q + 1;
   CEU_INT <= 1;
   CED_INT <= 0;
 end
   else if (CED)
 begin
  Q <= Q - 1;
  CEU_INT <= 1'b0;
  CED_INT <= 1'b1;
 end
end

always @ (Q or CEU_INT or CED_INT)
begin
 if (Q == TERMINAL_COUNT_UP && CEU_INT)
 begin
   TCU_INT <= 1;
   TCD_INT <= 0;
 end
 else if ((Q == TERMINAL_COUNT_DOWN) && CED_INT)
 begin
   TCU_INT <= 0;
   TCD_INT <= 1;
 end
 else
 begin
   TCU_INT <= 0;
   TCD_INT <= 0;
 end
end
```

```
always @(TCU_INT or CEU or TCD_INT or CED)
begin
   CEOU <= TCU_INT && CEU;
   CEOD <= TCD_INT && CED;
   TCU <= TCU_INT;
   TCD <= TCD_INT;
end
```

CBD2CE, CBD4CE, CBD8CE, CBD16CE

2-, 4-, 8-,16-Bit Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear

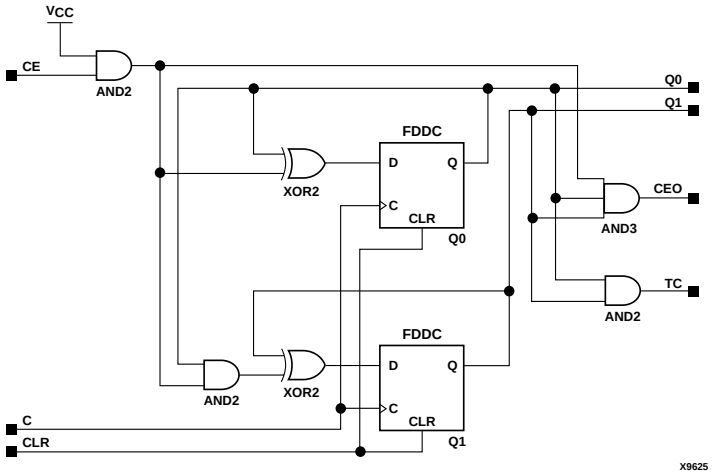
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



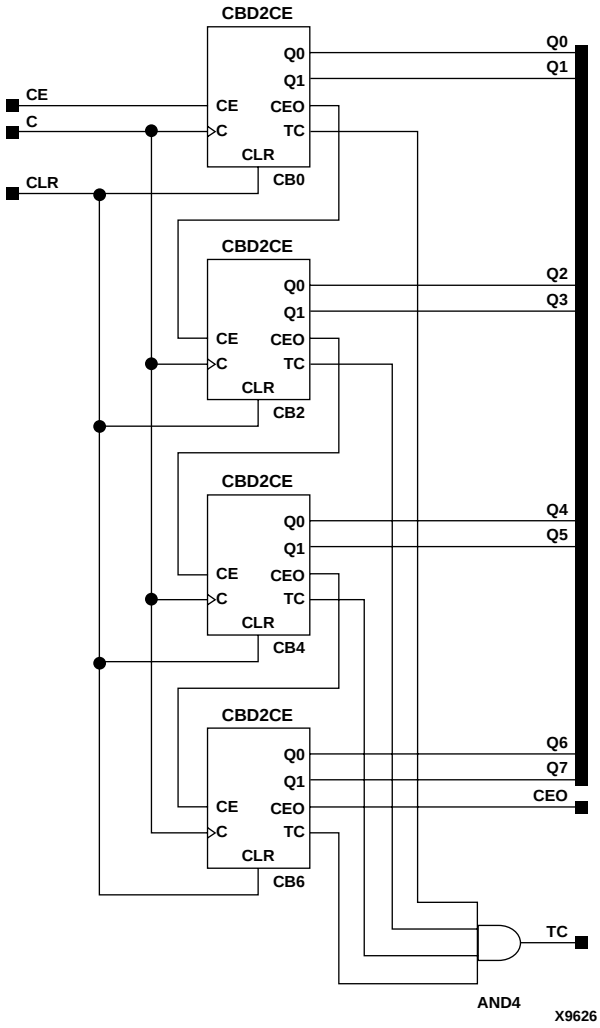
CBD2CE, CBD4CE, CBD8CE, and CBD16CE are, respectively, 2-, 4-, 8-, and 16-bit (stage), asynchronous, clearable, cascadable dual edge triggered binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.



CBD2CE Implementation CoolRunner-II



CBD8CE Implementation CoolRunner-II

Usage

For HDL, these design elements are supported for inference but not instantiation.

VHDL Inference Code

```
architecture Behavioral of cbd2ce is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
 elsif C'event then
  if (CE='1') then
```

```
   Q <= Q+1;
  end if;
 end if;
end process;

process (Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

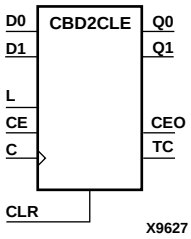
Verilog Inference Code

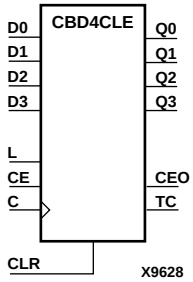
```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
   Q <= 0;
 else if (CE)
   Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

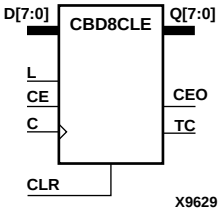
CBD2CLE, CBD4CLE, CBD8CLE, CBD16CLE

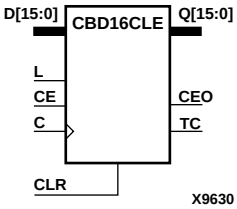
2-, 4-, 8-, 16-Bit Loadable Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro









CBD2CLE, CBD4CLE, CBD8CLE, and CBD16CLE are, respectively, 2-, 4-, 8-, and 16-bit (stage) synchronously loadable, asynchronously clearable, cascadable dual edge triggered binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock transition, independent of the state of clock enable (CE). The Q outputs increment when CE is High during the Low-to-High and High-to-Low clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C, L, and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

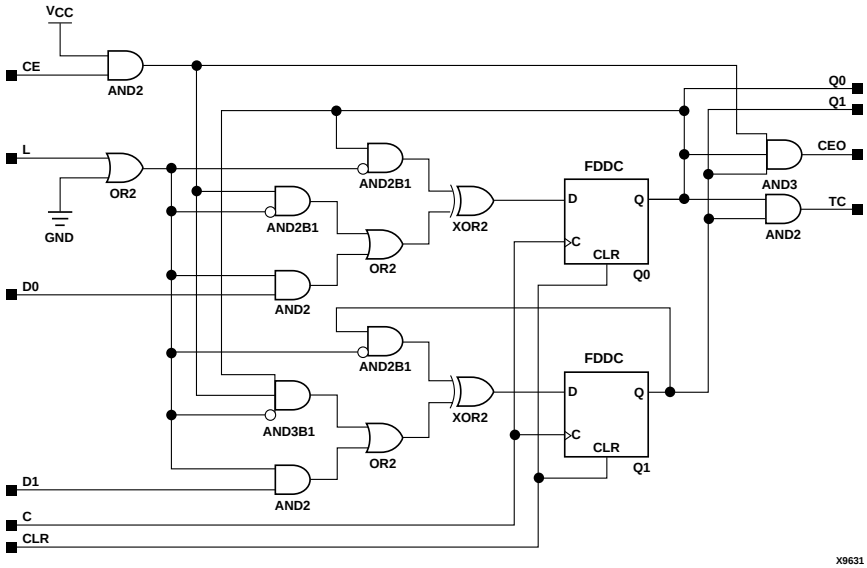
Inputs					Outputs		
CLR	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	0	0	0
0	1	X	↑	Dn	dn	TC	CEO
0	1	X	↓	Dn	dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO
0	0	1	↓	X	Inc	TC	CEO

z= 1 for CBD2CLE; z = 3 for CBD4CLE; z = 7 for CBD8CLE; z = 15 for CBD16CLE

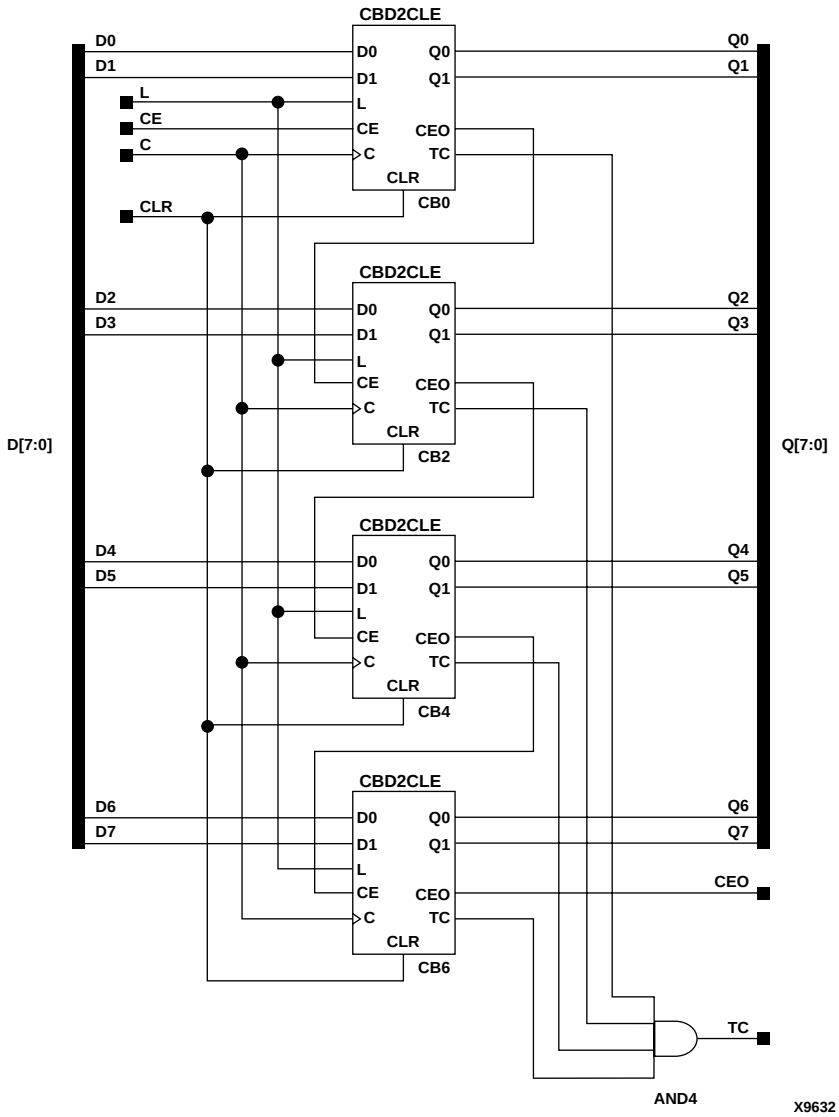
dn = state of referenced input (Dn) one setup time prior to active clock transition.

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEO = TC • CE



CBD2CLE Implementation CoolRunner-II



CBD8CLE Implementation CoolRunner-II

Usage

For HDL, these design elements are supported for inference but not instantiation.

VHDL Inference Code

```
architecture Behavioral of cbd2cle is

   constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C, CLR)
begin
 if (CLR='1') then
```

```
  Q <= (others => '0');
 elsif C'event then
  if (L = '1') then
   Q <= D;
  elsif (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process(Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

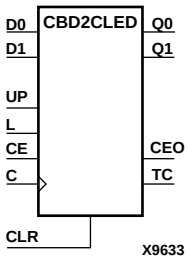
Verilog Inference Code

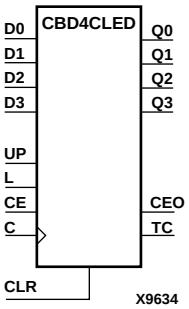
```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

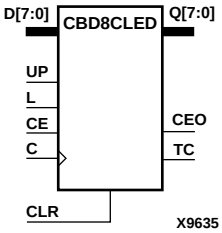
CBD2CLED, CBD4CLED, CBD8CLED, CBD16CLED

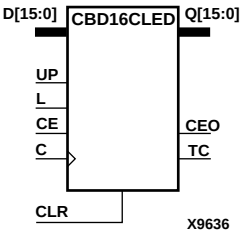
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro









CBD2CLED, CBD4CLED, CBD8CLED, and CBD16CLED are, respectively, 2-, 4-, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional dual edge triggered binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High and High-to-Low clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.

For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the CEO output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage.

When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not. For CoolRunner-II, see the "CB2X1, CB4X1, CB8X1, CB16X1" section for high-performance cascadable, bidirectional counters.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

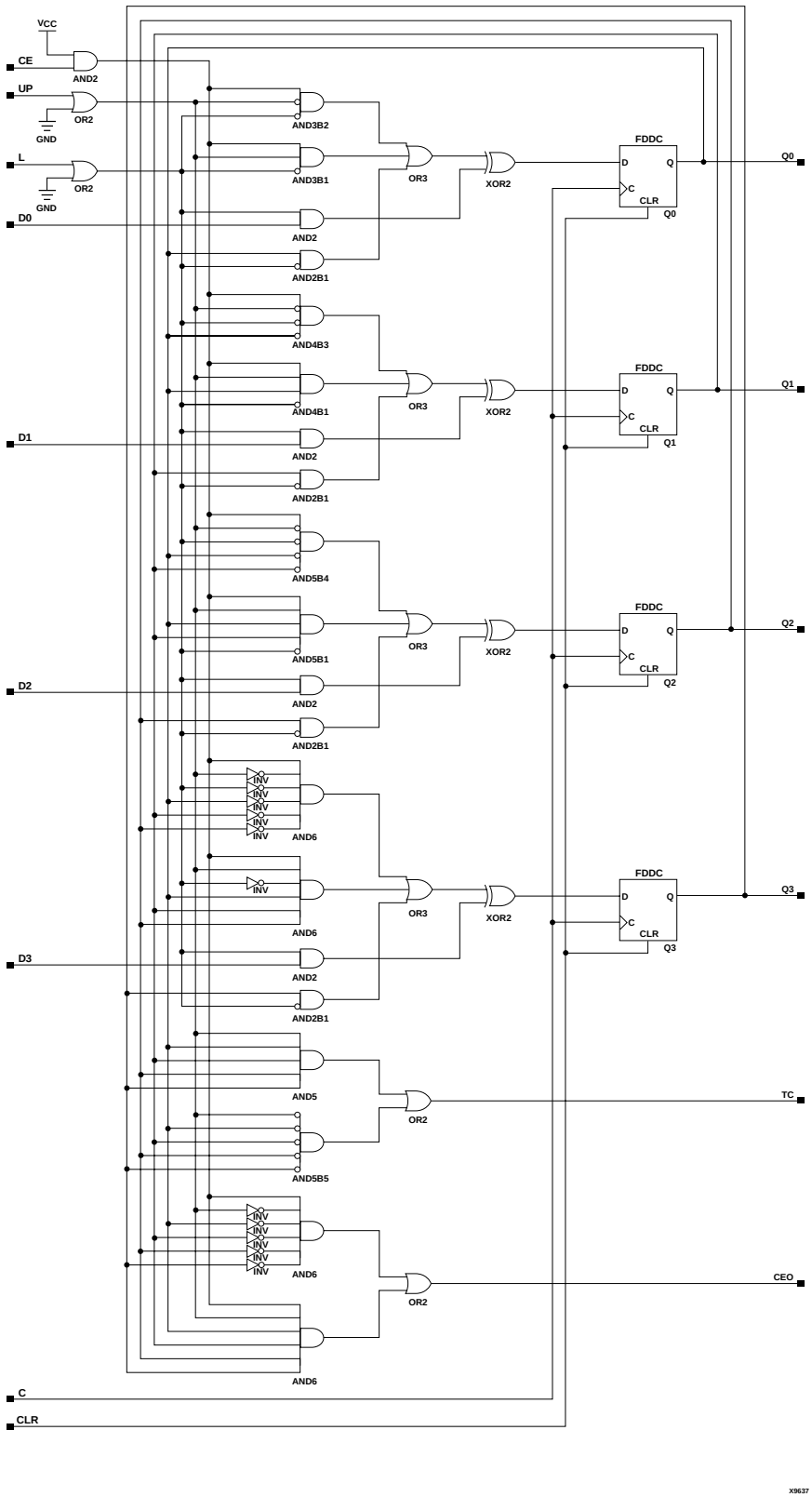
Inputs						Outputs		
CLR	**L**	**CE**	**C**	**UP**	**Dz – D0**	**Qz – Q0**	**TC**	**CEO**
1	X	X	X	X	X	0	0	0
0	1	X	↑	X	Dn	dn	TC	CEO
0	1	X	↓	X	Dn	dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↓	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO
0	0	1	↓	0	X	Dec	TC	CEO

z = 1 for CBD2CLED; z = 3 for CBD4CLED; z = 7 for CBD8CLED; z = 15 for CBD16CLED

dn = state of referenced input (Dn), one setup time prior to active clock transition

TC = (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP) + (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP)

CEO = TC • CE



CBD4CLED Implementation CoolRunner-II

Usage

For HDL, these design elements are supported for inference but not instantiation.

VHDL Inference Code

```
architecture Behavioral of cbd2cled is

 constant TERMINAL_COUNT_UP : std_logic_vector(WIDTH-1 downto
   0) := (others => '1');
 constant TERMINAL_COUNT_DOWN : std_logic_vector(WIDTH-1 downto
   0) := (others => '0');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
 elsif (C'event) then
  if (L = '1') then
   Q <= D;
  elsif (CE='1') then
   if (UP='1') then
    Q <= Q+1;
   elsif (UP='0') then
    Q <= Q-1;
   end if;
  end if;
 end if;
end process;

process(Q, UP)
begin
 if (((Q = TERMINAL_COUNT_UP) and (UP = '1')) or
  ((Q = TERMINAL_COUNT_DOWN) and (UP = '0'))) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO <=TC and CE;

end Behavioral;
```

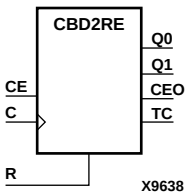
Verilog Inference Code

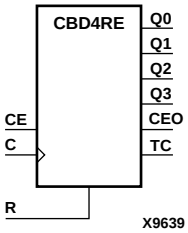
```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  begin
   if (UP)
      Q <= Q + 1;
   else if (!UP)
    Q <= Q - 1;
  end
end

always @ (Q or UP)
begin
 if ((Q == TERMINAL_COUNT_UP && UP) || (Q == TERMINAL_COUNT_DOWN
   && !UP))
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

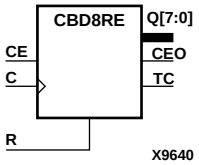
CBD2RE, CBD4RE, CBD8RE, CBD16RE

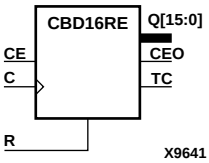
2-, 4-, 8-, 16-Bit Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro









CBD2RE, CBD4RE, CBD8RE, and CBD16RE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, resettable, cascadable dual edge triggered binary counters. The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero during the Low-to-High or High-to-Low clock transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when both Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and R inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

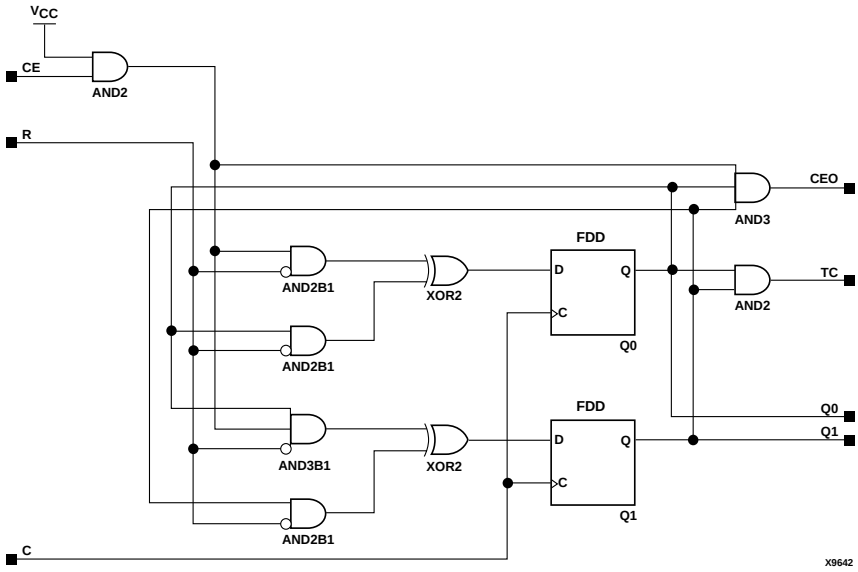
The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
1	X	↑	0	0	0
1	X	↓	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO
0	1	↓	Inc	TC	CEO

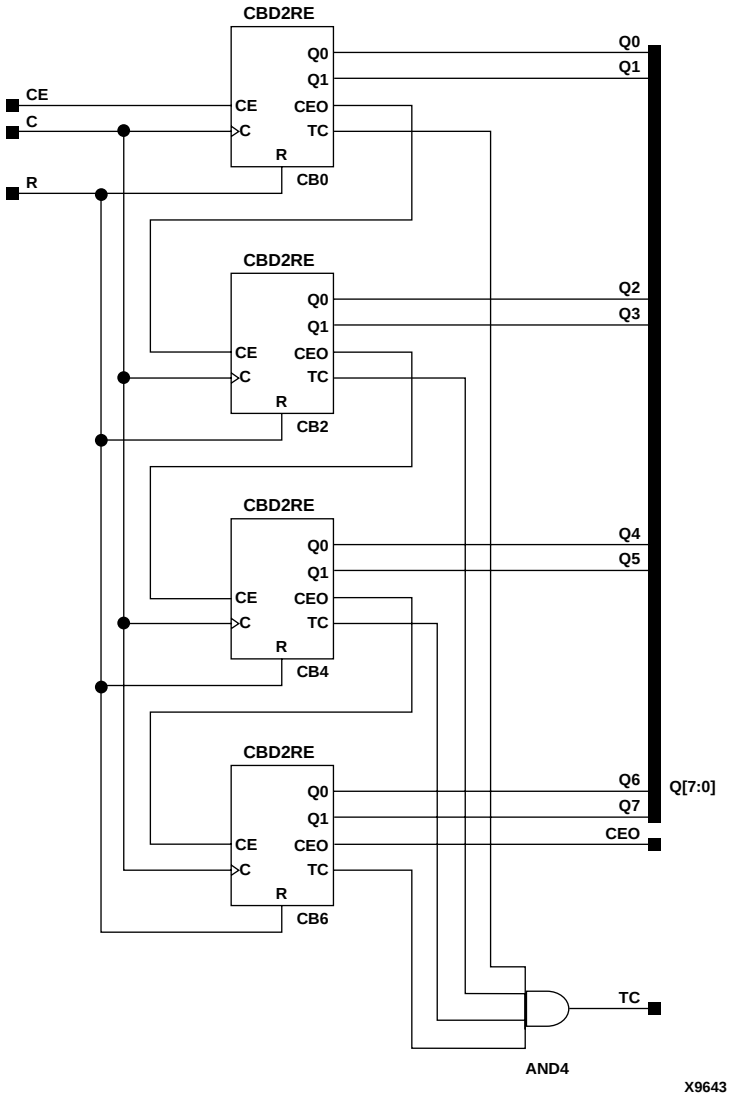
z = 1 for CBD2RE; z = 3 for CBD4RE; z = 7 for CBD8RE; z = 15 for CBD16RE

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0)

CEO = TC • CE



CBD2RE Implementation CoolRunner-II



CBD8RE Implementation CoolRunner-II

Usage

For HDL, these design elements are supported for inference but not instantiation.

VHDL Inference Code

```
architecture Behavioral of cbd2re is

   constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin
process(C, R)
begin
 if (C'event) then
  if (R='1') then
   Q <= (others => '0');
  elsif (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process(Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO <= TC and CE;

end Behavioral;
```

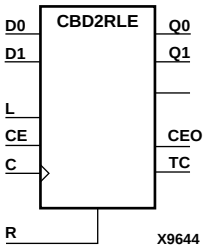
Verilog Inference Code

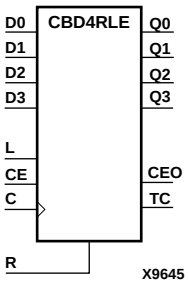
```
always @ (posedge C or negedge C)
begin
 if (R)
  Q <= 0;
 else if (CE)
    Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

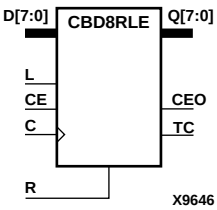
CBD2RLE, CBD4RLE, CBD8RLE, CBD16RLE

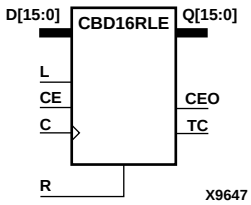
2-, 4-, 8-, 16-Bit Loadable Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro









CBD2RLE, CBD4RLE, CBD8RLE, and CBD16RLE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, cascadable dual edge triggered binary counters. The synchronous reset (R) is the highest priority input. The synchronous R, when High, overrides all other inputs and resets the Q outputs, terminal count (TC), and clock enable out (CEO) outputs to Low on the Low-to-High or High-to-Low clock (C) transition.

The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High and High-to-Low clock (C) transition, independent of the state of CE. The Q outputs increment when CE is High during the Low-to-High and High-to-Low clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High. The CEO output is High when all Q outputs and CE are High to allow direct cascading of counters.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and by connecting the C, L, and R inputs in parallel. The maximum length of the counter is determined by the accumulated CE-to-CEO propagation delays versus the clock period. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

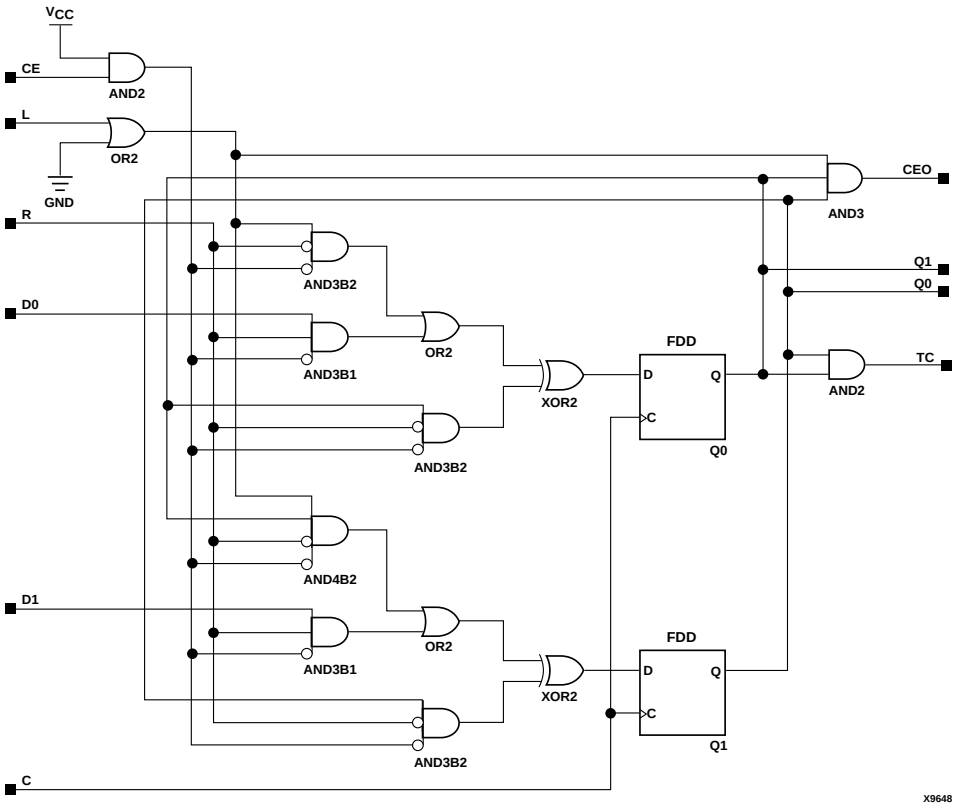
Inputs					Outputs		
R	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	↑	X	0	0	0
1	X	X	↓	X	0	0	0
0	1	X	↑	Dn	dn	TC	CEO
0	1	X	↓	Dn	dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO
0	0	1	↓	X	Inc	TC	CEO

z = 1 for CBD2RLE; z = 3 for CBD4RLE; z = 7 for CBD8RLE; z = 15 for CBD16RLE

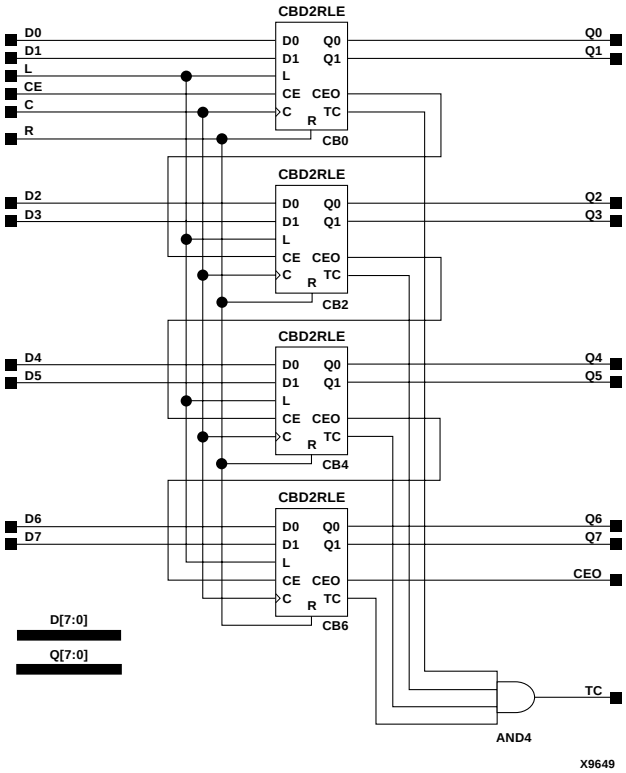
dn = state of referenced input (Dn) one setup time prior to active clock transition

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEO = TC • CE



CBD2RLE Implementation CoolRunner-II



CBD8RLE Implementation CoolRunner-II

Usage

For HDL, these design elements are supported for inference but not instantiation.

VHDL Inference Code

```
architecture Behavioral of cbd2rle is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C, R)
begin
 if (C'event) then
  if (R='1') then
   Q <= (others => '0');
  elsif (L='1') then
   Q <= D;
  elsif (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process(Q)
```

```
begin
 if (Q = TERMINAL_COUNT) then
  TC <= '1';
 else
  TC <= '0';
 end if;
end process;

CEO <= TC and CE;

end Behavioral;
```

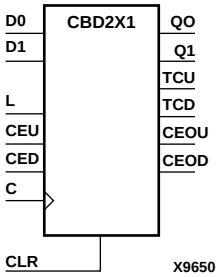
Verilog Inference Code

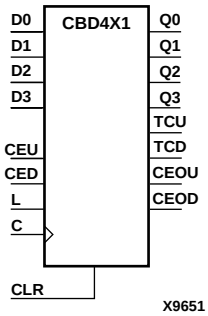
```
always @ (posedge C or negedge C)
begin
 if (R)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
    Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

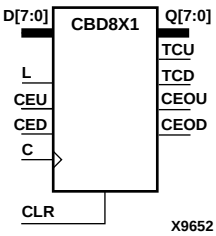
CBD2X1, CBD4X1, CBD8X1, CBD16X1

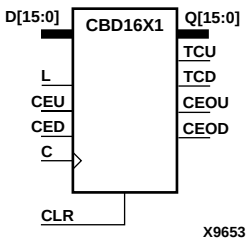
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro









CBD2X1, CBD4X1, CBD8X1, and CBD16X1 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronously loadable, asynchronously clearable, bidirectional dual edge triggered binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in the CoolRunner-II architecture.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, clock enable outputs CEOU and CEOD go to Low and High, respectively, independent of clock transitions. The data on the D inputs loads into the counter on the Low-to-High and High-to-Low clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

The Q outputs increment when CEU is High, provided CLR and L are Low, during the Low-to-High and High-to-Low clock transition. The Q outputs decrement when CED is High, provided CLR and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are connected directly to the CEU and CED inputs, respectively, of the next stage. The clock, L, and CLR inputs are connected in parallel.

In CoolRunner-II, the maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
CLR	**L**	**CEU**	**CED**	**C**	**Dz–D0**	**Qz–Q0**	**TCU**	**TCD**	**CEOU**	**CEOD**
1	X	X	X	X	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	dn	TCU	TCD	CEOU	CEOD
0	1	X	X	↓	Dn	dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	1	0	↓	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	0	1	↓	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid
0	0	1	1	↓	X	Inc	TCU	TCD	Invalid	Invalid

z = 1 for CBD2X1; z = 3 for CBD4X1; z = 7 for CBD8X1; z = 15 for CBD16X1

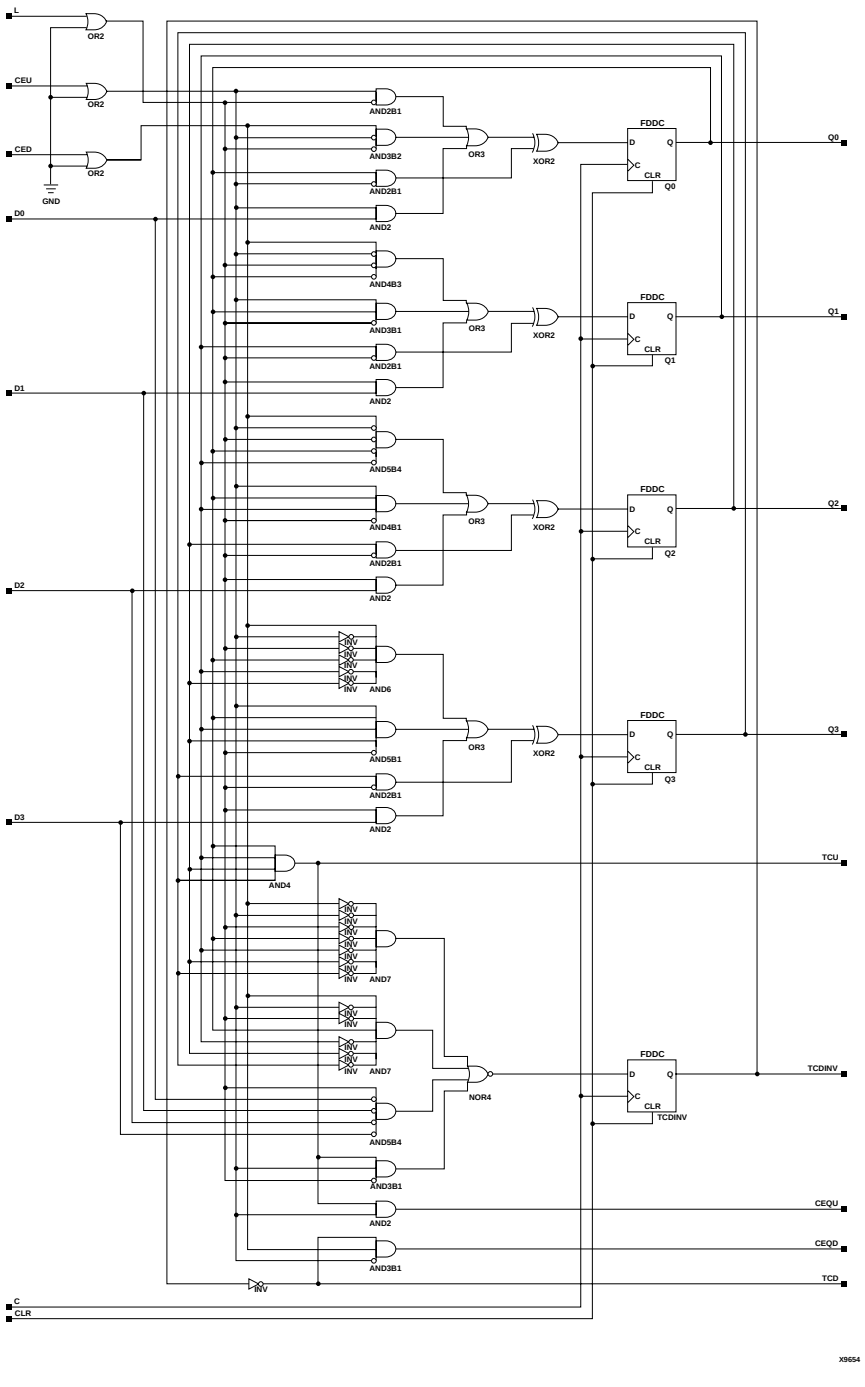
dn = state of referenced input (Dn) one setup time prior to active clock transition

TCU = Qz • Q(z-1) • Q(z-2) • ... • Q0

TCD = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEOU = TCU • CEU

CEOD = TCD • CED



CBD4X1 Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cbd2x1 is

 signal CEU_INT : std_logic;
 signal CED_INT : std_logic;

 constant TERMINAL_COUNT_UP : std_logic_vector(WIDTH-1 downto
   0) := (others => '1');
 constant TERMINAL_COUNT_DOWN : std_logic_vector(WIDTH-1 downto
   0) := (others => '0');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
  CEU_INT <= '0';
  CED_INT <= '1';
 elsif C'event then
  if (L = '1') then
   Q <= D;
   CEU_INT <= CEU;
   CED_INT <= CED;
  elsif (CEU='1') then
   Q <= Q+1;
   CEU_INT <= '1';
   CED_INT <= '0';
  elsif (CED='1') then
   Q <= Q-1;
   CEU_INT <= '0';
   CED_INT <= '1';
  end if;
 end if;
end process;

process(Q, CEU_INT, CED_INT)
begin
 if ((Q = TERMINAL_COUNT_UP) and (CEU_INT = '1')) then
  TCU_INT <= '1';
  TCD_INT <= '0';
 elsif ((Q = TERMINAL_COUNT_DOWN) and (CED_INT = '1')) then
  TCU <= '0';
  TCD <= '1';
 else
  TCU <= '0';
  TCD <= '0';
 end if;
end process;

CEOU <= TCU and CEU;
CEOD <= TCD and CED;
```

```
end Behavioral;
```

Verilog Inference Code

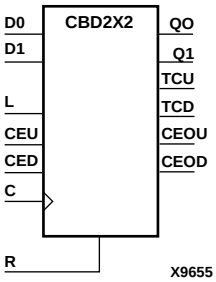
```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
 begin
  Q <= 0;
  CEU_INT <= 1'b0;
  CED_INT <= 1'b1;
 end
 else if (L)
 begin
  Q <= D;
  CEU_INT <= CEU;
  CED_INT <= CED;
 end
 else if (CEU)
  begin
          Q <= Q + 1;
  CEU_INT <= 1;
  CED_INT <= 0;
 end
   else if (CED)
 begin
  Q <= Q - 1;
  CEU_INT <= 1'b0;
  CED_INT <= 1'b1;
 end
end

always @ (Q or CEU_INT or CED_INT)
begin
 if (Q == TERMINAL_COUNT_UP && CEU_INT)
 begin
 TCU_INT <= 1;
 TCD_INT <= 0;
 end
 else if ((Q == TERMINAL_COUNT_DOWN) && CED_INT)
 begin
 TCU_INT <= 0;
 TCD_INT <= 1;
   end
 else
 begin
 TCU_INT <= 0;
 TCD_INT <= 0;
 end
end

always @ (TCU_INT or CEU or TCD_INT or CED)
begin
CEOU <= TCU_INT && CEU;
```

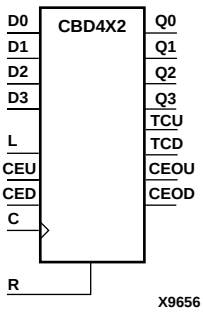
```
CEOD <= TCD_INT && CED;
TCU <= TCU_INT;
TCD <= TCD_INT;
end
```

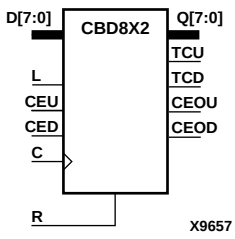
CBD2X2, CBD4X2, CBD8X2, CBD16X2

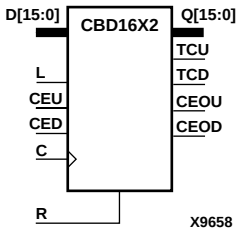
2-, 4-, 8-, and 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro









CBD2X2, CBD4X2, CBD8X2, and CBD16X2 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, bidirectional dual edge triggered binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in the CoolRunner-II architecture.

The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, and clock enable outputs CEOU and CEOD go to Low and High, respectively, on the Low-to-High and High-to-Low clock (C) transition. The data on the D inputs loads into the counter on the Low-to-High and High-to-Low clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

All Q outputs increment when CEU is High, provided R and L are Low during the Low-to-High and High-to-Low clock transition. All Q outputs decrement when CED is High, provided R and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are, respectively, connected directly to the CEU and CED inputs of the next stage. The C, L, and R inputs are connected in parallel.

In CoolRunner-II, the maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
R	**L**	**CEU**	**CED**	**C**	**Dz – D0**	**Qz – Q0**	**TCU**	**TCD**	**CEOU**	**CEOD**
1	X	X	X	↑	X	0	0	1	0	CEOD
1	X	X	X	↓	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	dn	TCU	TCD	CEOU	CEOD
0	1	X	X	↓	Dn	dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	1	0	↓	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	0	1	↓	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid
0	0	1	1	↓	X	Inc	TCU	TCD	Invalid	Invalid

z = 1 for CBD2X2; z = 3 for CBD4X2; z = 7 for CBD8X2; z = 15 for CBD16X2

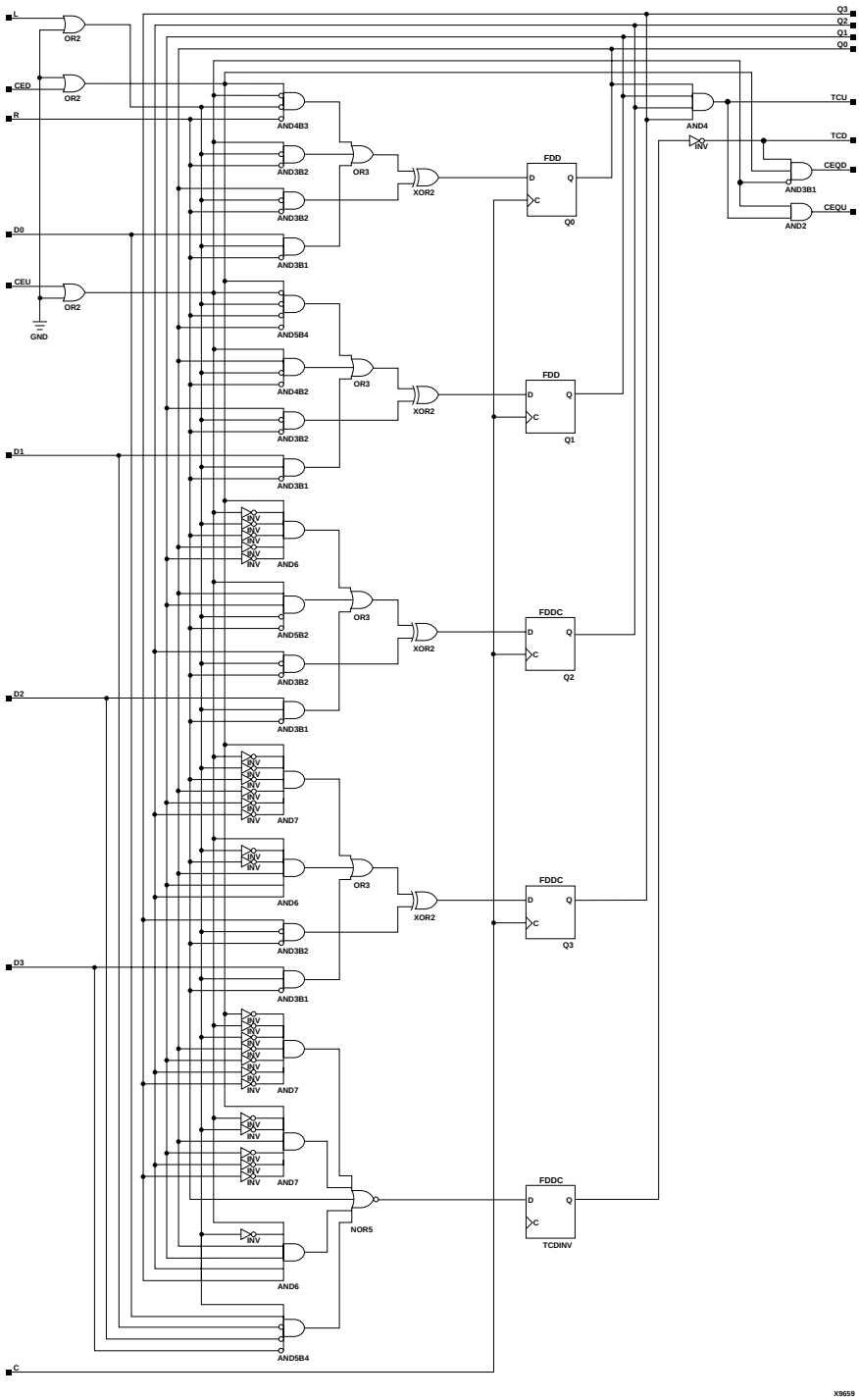
dn = state of referenced input (Dn) one setup time prior to active clock transition

TCU = Qz • Q(z-1) • Q(z-2) • ... • Q0

TCD = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEOU = TCU • CEU

CEOD = TCD • CED



CBD4X2 Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cbd2x2 is

 signal CEU_INT : std_logic;
 signal CED_INT : std_logic;

 constant TERMINAL_COUNT_UP : std_logic_vector(WIDTH-1 downto
   0) := (others => '1');
 constant TERMINAL_COUNT_DOWN : std_logic_vector(WIDTH-1 downto
   0) := (others => '0');

begin

process(C)
begin
 if (C'event) then
  if (R='1') then
   Q <= (others => '0');
   CEU_INT <= '0';
   CED_INT <= '1';
  elsif (L = '1') then
   Q <= D;
   CEU_INT <= CEU;
   CED_INT <= CED;
  elsif (CEU='1') then
   Q <= Q+1;
   CEU_INT <= '1';
   CED_INT <= '0';
  elsif (CED='1') then
   Q <= Q-1;
   CEU_INT <= '0';
   CED_INT <= '1';
  end if;
 end if;
end process;

process(Q, CEU_INT, CED_INT)
begin
 if ((Q = TERMINAL_COUNT_UP) and (CEU_INT = '1')) then
  TCU_INT <= '1';
  TCD_INT <= '0';
 elsif ((Q = TERMINAL_COUNT_DOWN) and (CED_INT = '1')) then
  TCU <= '0';
  TCD <= '1';
 else
  TCU <= '0';
  TCD <= '0';
 end if;
end process;

CEOU <= TCU and CEU;
CEOD <= TCD and CED;
```

```
end Behavioral;
```

Verilog Inference Code

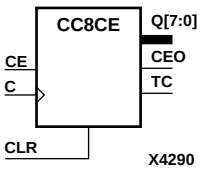
```
always @ (posedge C or negedge C)
begin
 if (R)
 begin
  Q <= 0;
  CEU_INT <= 1'b0;
  CED_INT <= 1'b1;
 end
 else if (L)
 begin
  Q <= D;
  CEU_INT <= CEU;
  CED_INT <= CED;
 end
 else if (CEU)
  begin
          Q <= Q + 1;
  CEU_INT <= 1;
  CED_INT <= 0;
 end
   else if (CED)
 begin
  Q <= Q - 1;
  CEU_INT <= 1'b0;
  CED_INT <= 1'b1;
 end
end

always @ (Q or CEU_INT or CED_INT)
begin
 if (Q == TERMINAL_COUNT_UP && CEU_INT)
 begin
 TCU_INT <= 1;
 TCD_INT <= 0;
 end
 else if ((Q == TERMINAL_COUNT_DOWN) && CED_INT)
 begin
 TCU_INT <= 0;
 TCD_INT <= 1;
   end
 else
 begin
   TCU_INT <= 0;
   TCD_INT <= 0;
 end
end

always @ (TCU_INT or CEU or TCD_INT or CED)
begin
   CEOU <= TCU_INT && CEU;
```

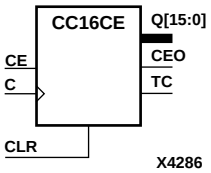
```
   CEOD <= TCD_INT && CED;
   TCU <= TCU_INT;
   TCD <= TCD_INT;
end
```

CC8CE, CC16CE

8-, 16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A





CC8CE and CC16CE are, respectively, 8- and 16-bit (stage), asynchronous clearable, cascadable binary counters. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, with Low outputs, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

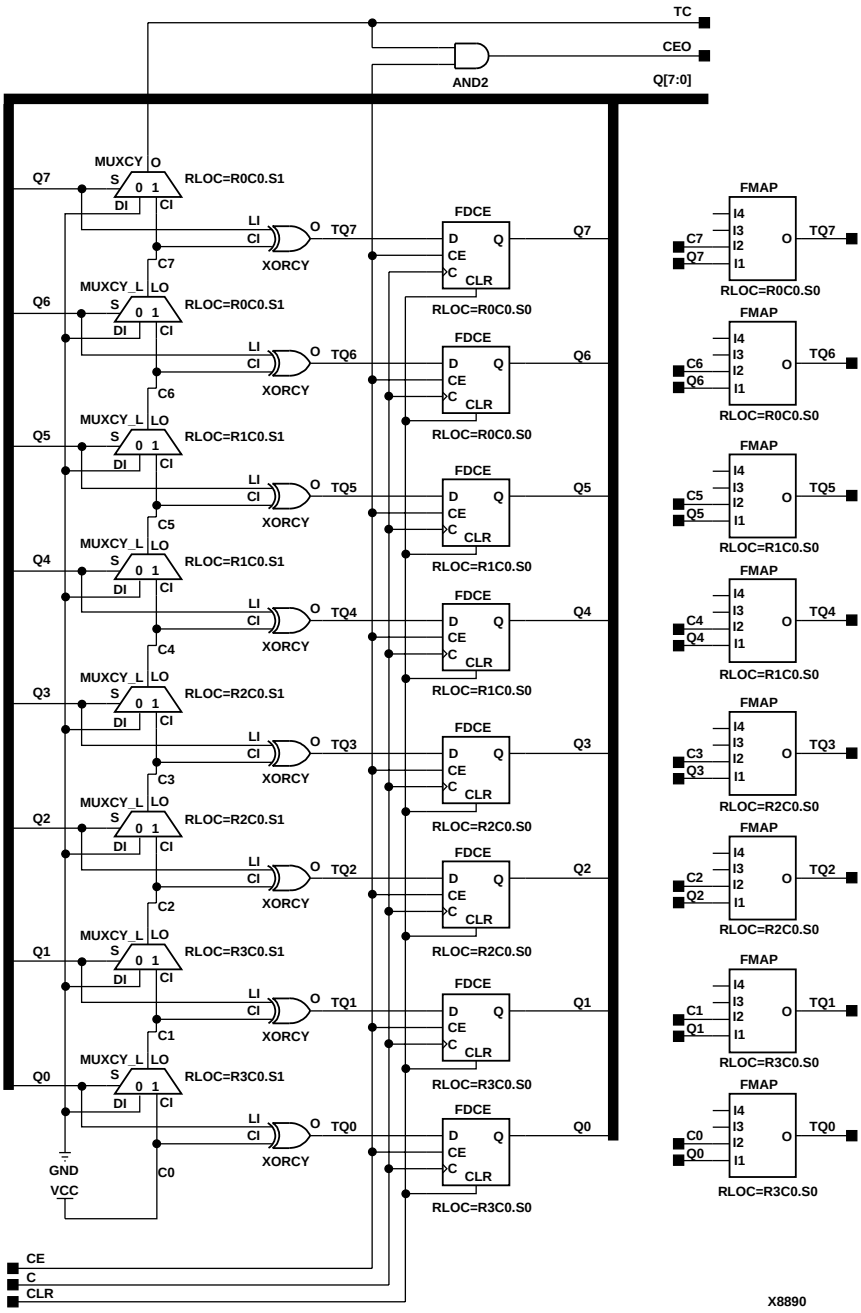
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs		
CLR	CE	C	Qz – Q0	TC	CEO
1	X	X	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

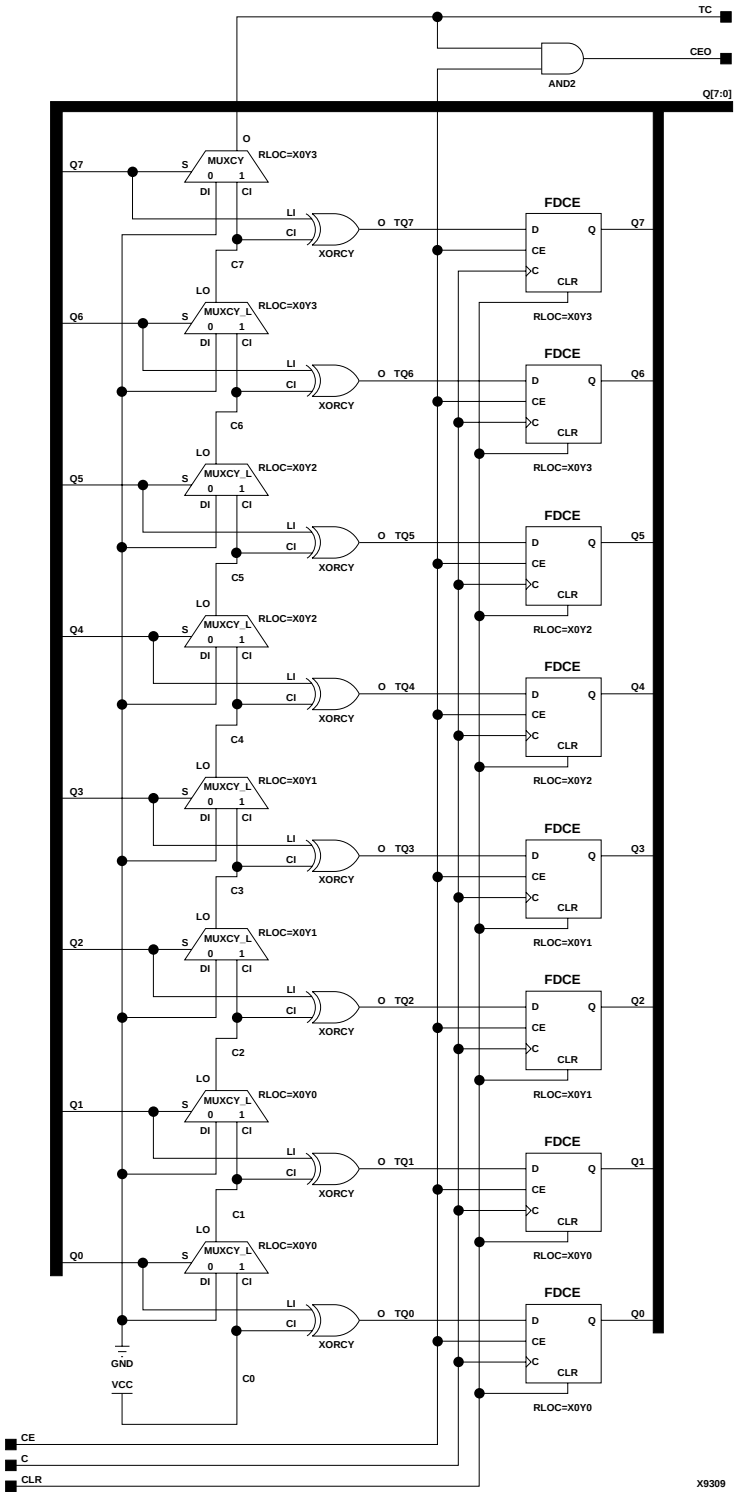
z = 7 for CC8CE; z = 15 for CC16CE

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEO = TC • CE



CC8CE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



CC8CE Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cc8ce is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
 elsif (C'event and C='1') then
  if (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process (Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <= '1';
 else
  TC <= '0';
 end if;
end process;

CEO <= TC and CE;

end Behavioral;
```

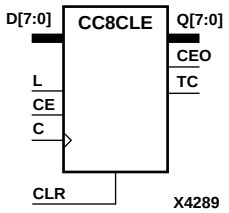
Verilog Inference Code

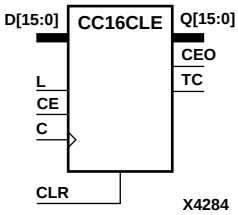
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
    Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

CC8CLE, CC16CLE

8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A





CC8CLE and CC16CLE are, respectively, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable binary counters. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the C, L, and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, with Low output, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

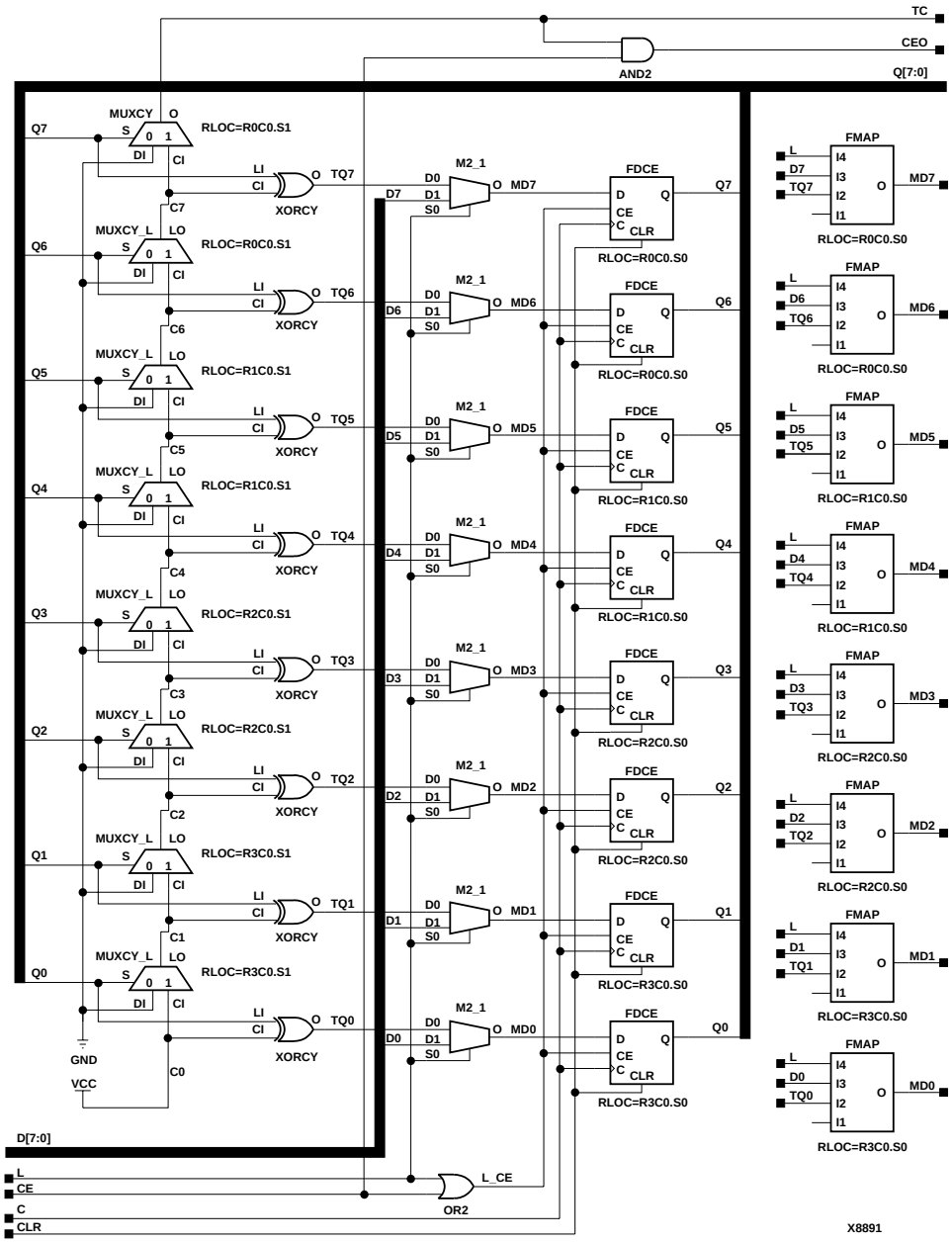
Inputs					Outputs		
CLR	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	0	0	0
0	1	X	↑	Dn	dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

z = 7 for CC8CLE; z = 15 for CC16CLE

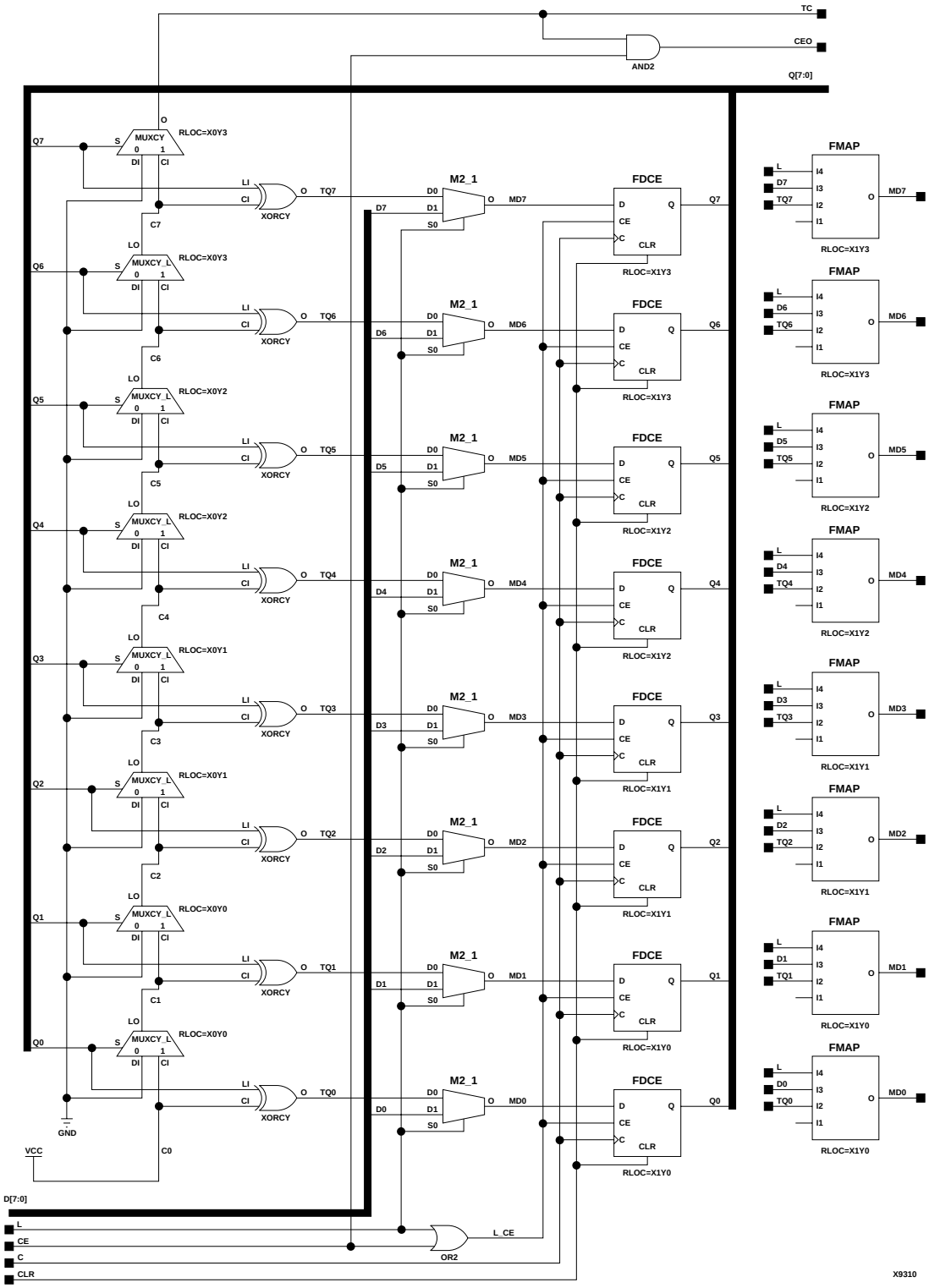
dn = state of referenced input (Dn) one setup time prior to active clock transition

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEO = TC • CE



CC8CLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



CC8CLE Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cc8cle is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
 elsif (C'event and C='1') then
  if (L = '1') then
   Q <= D;
  elsif (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process(Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC <= '1';
 else
  TC <= '0';
 end if;
end process;

CEO <= TC and CE;

end Behavioral;
```

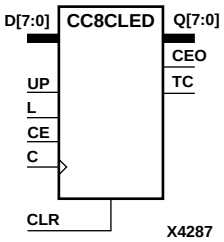
Verilog Inference Code

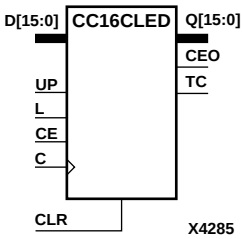
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  Q <= Q + 1;
end

always @ (Q)
begin
 if (Q == TERMINAL_COUNT)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

CC8CLED, CC16CLED

8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A





CC8CLED and CC16CLED are, respectively, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional binary counters. These counters are implemented using carry logic with relative location constraints, which assures most efficient logic placement.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.

For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the count enable out (CEO) output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, outputs Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

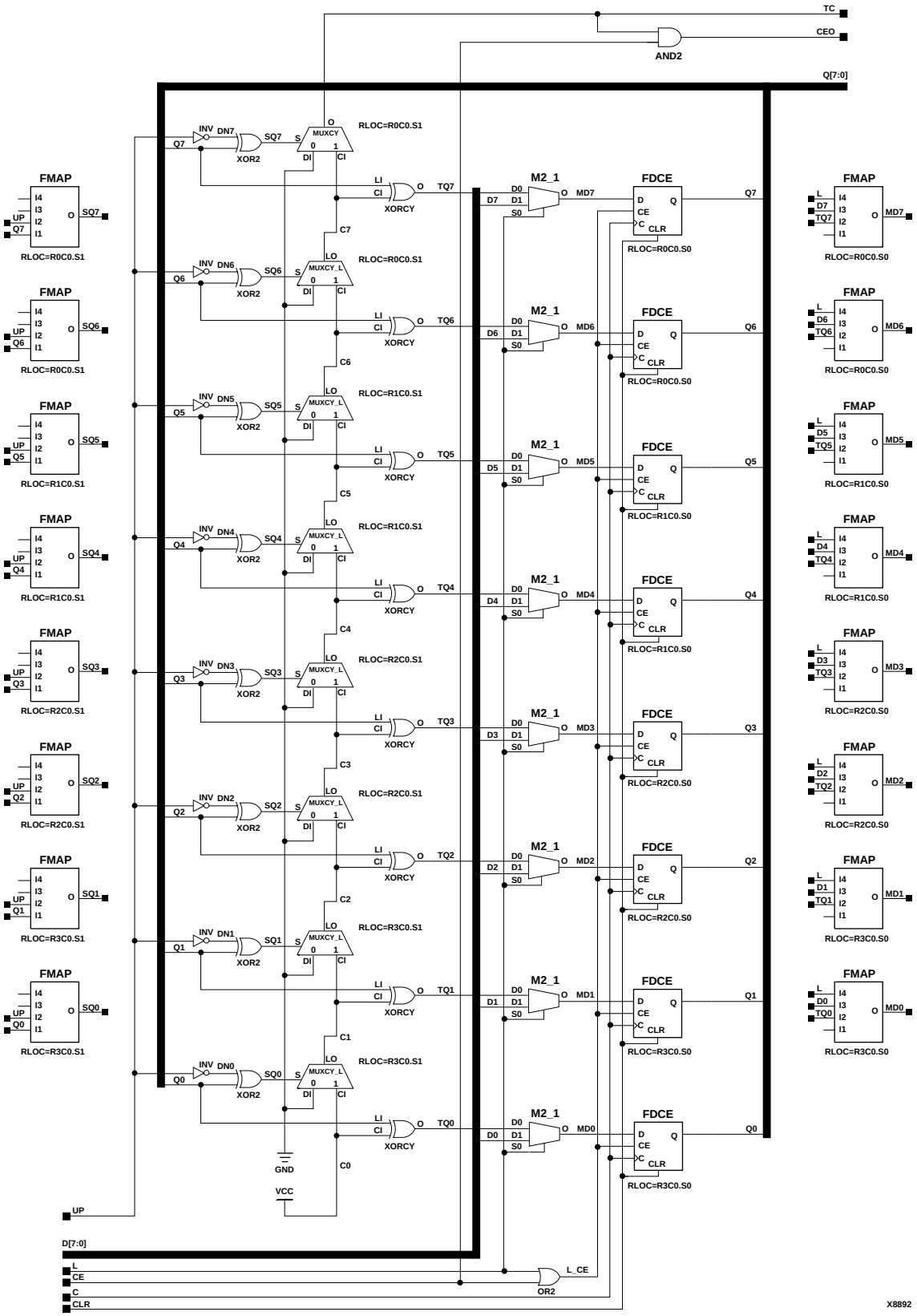
Inputs						Outputs		
CLR	**L**	**CE**	**C**	**UP**	**Dz – D0**	**Qz – Q0**	**TC**	**CEO**
1	X	X	X	X	X	0	0	0
0	1	X	↑	X	Dn	dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO

z = 7 for CC8CLED; z = 15 for CC16CLED

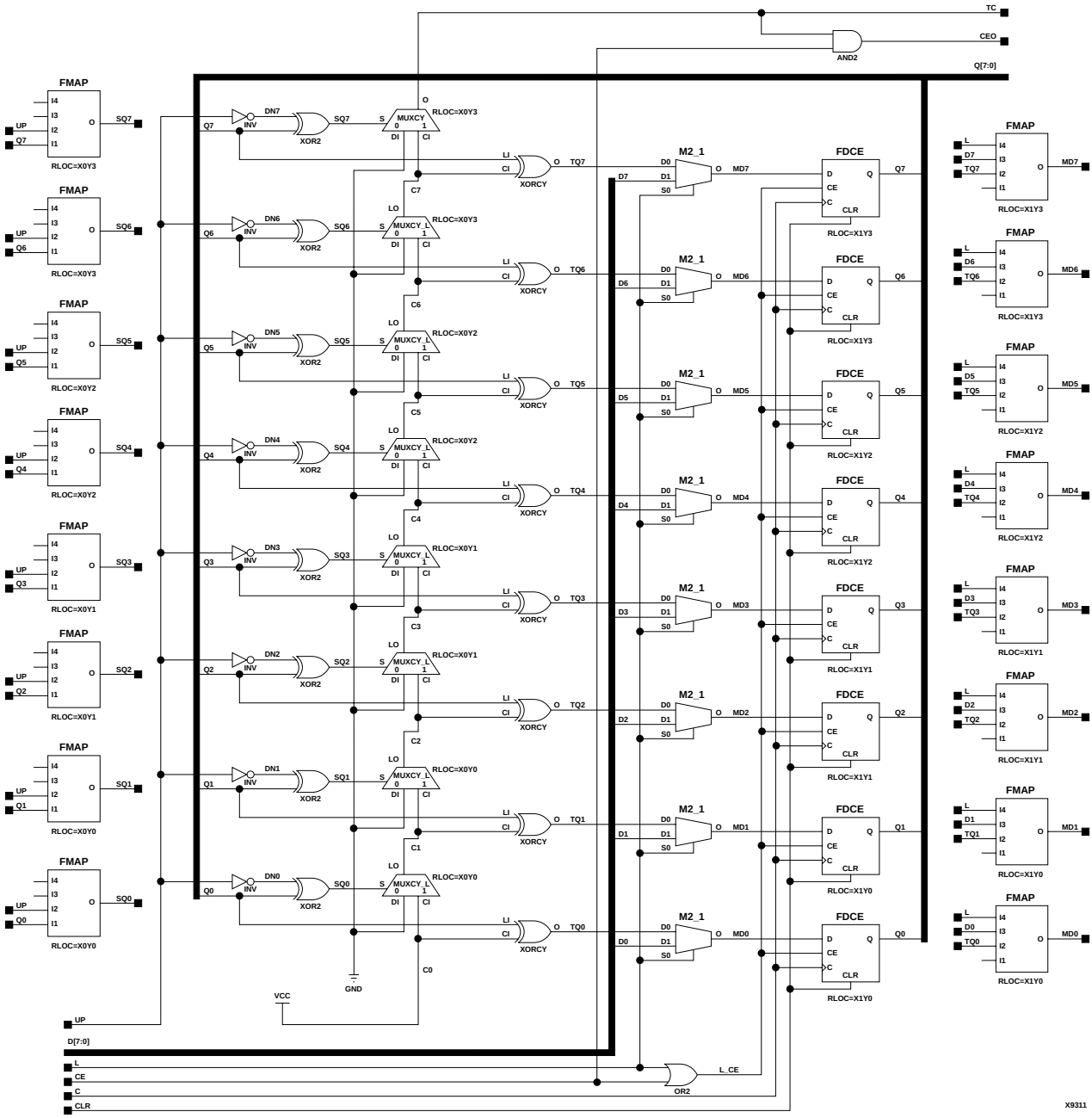
dn = state of referenced input (Dn) one setup time prior to active clock transition

TC = (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP) + (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP)

CEO = TC • CE



CC8CLED Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



CC8CLED Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cc8cled is

 constant TERMINAL_COUNT_UP : std_logic_vector(WIDTH-1 downto
   0) := (others => '1');
 constant TERMINAL_COUNT_DOWN : std_logic_vector(WIDTH-1 downto
   0) := (others => '0');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
 elsif (C'event and C='1') then
  if (L = '1') then
   Q <= D;
  elsif (CE='1') then
   if (UP='1') then
    Q <= Q+1;
   elsif (UP='0') then
    Q <= Q-1;
   end if;
  end if;
 end if;
end process;

process(Q, UP)
begin
 if (((Q = TERMINAL_COUNT_UP) and (UP = '1')) or
  ((Q = TERMINAL_COUNT_DOWN) and (UP = '0'))) then
  TC<='1';
 else
  TC<='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

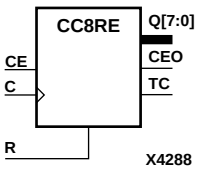
Verilog Inference Code

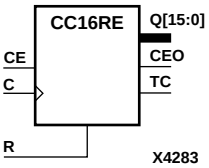
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  begin
   if (UP)
      Q <= Q + 1;
   else if (!UP)
      Q <= Q - 1;
  end
end

always @ (Q or UP)
begin
 if ((Q == TERMINAL_COUNT_UP && UP) || (Q == TERMINAL_COUNT_DOWN
   && !UP))
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```

CC8RE, CC16RE

8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A





CC8RE and CC16RE are, respectively, 8- and 16-bit (stage), synchronous resettable, cascadable binary counters. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic. The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs and CE are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and R inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, with Low outputs, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

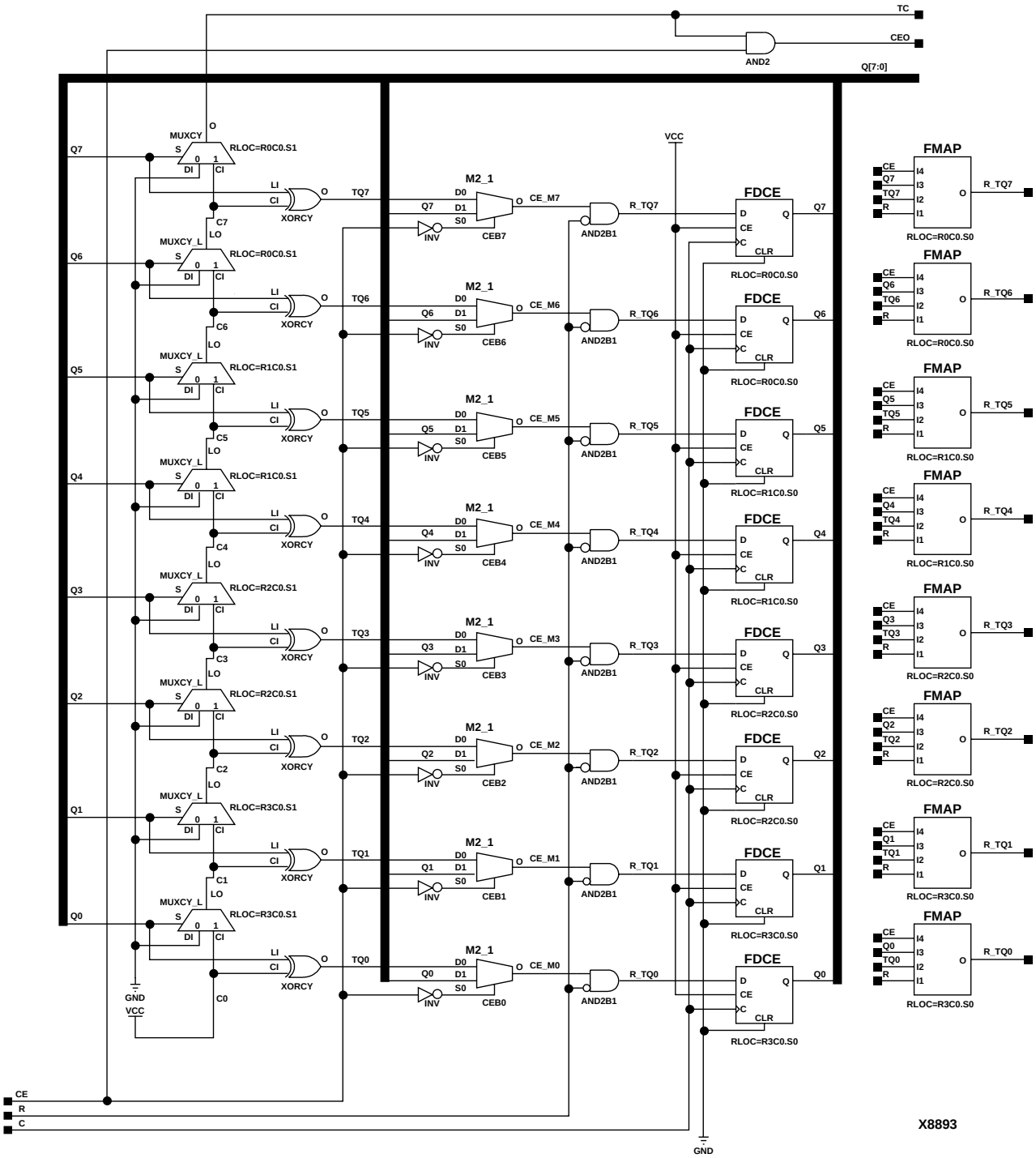
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
1	X	↑	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

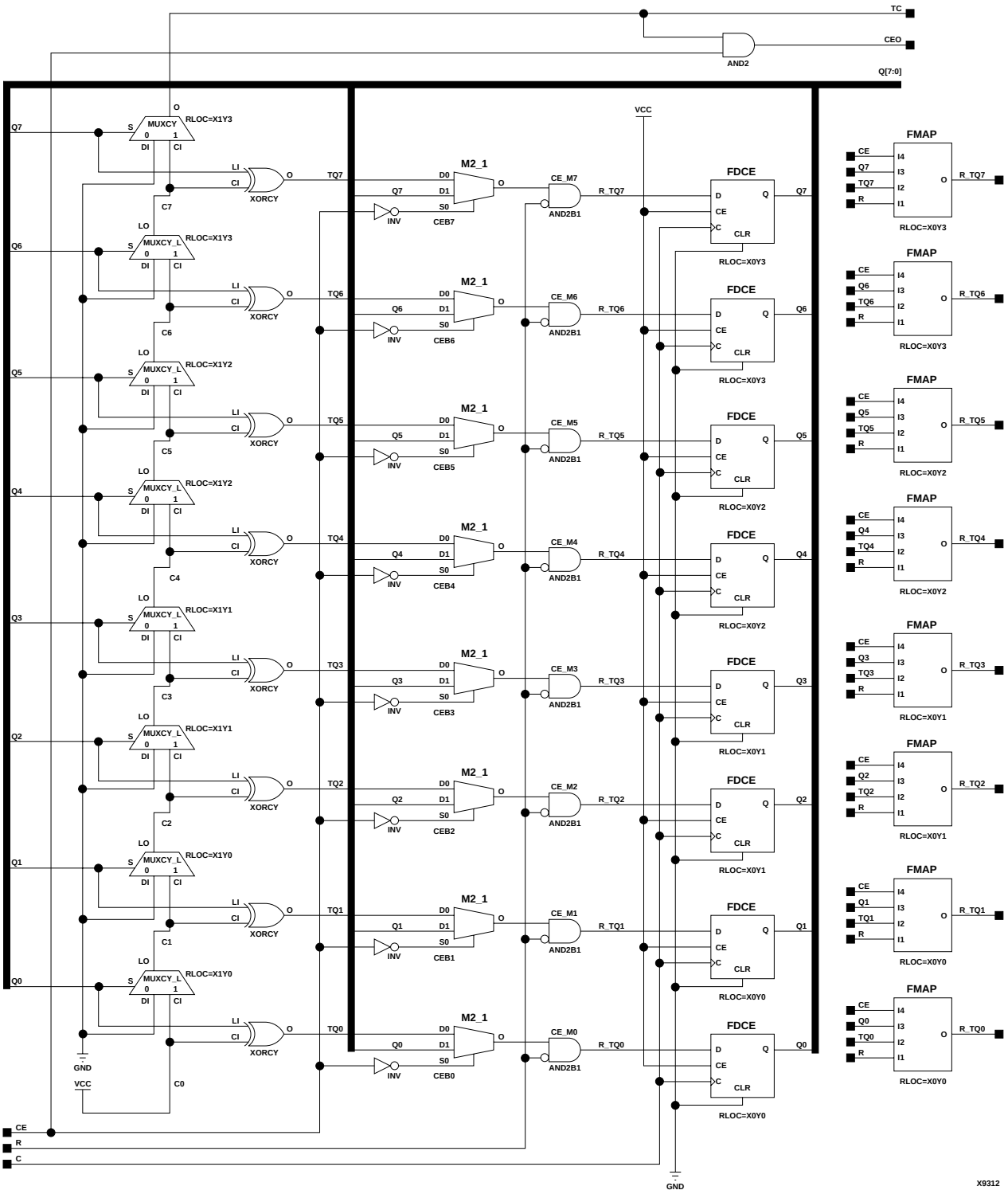
z = 7 for CC8RE; z = 15 for CC16RE

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0 • CE

CEO = TC • CE



CC8RE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



CC8RE Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cc8re is

 constant TERMINAL_COUNT : std_logic_vector(WIDTH-1 downto 0)
   := (others => '1');

begin

process(C, R)
begin
 if (C'event and C='1') then
  if (R='1') then
   Q <= (others => '0');
  elsif (CE='1') then
   Q <= Q+1;
  end if;
 end if;
end process;

process(Q)
begin
 if (Q = TERMINAL_COUNT) then
  TC<='1';
 else
  TC<='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

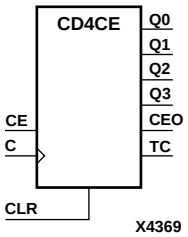
Verilog Inference Code

```
always @(posedge C)
begin
        if (R)
                Q <= 0;
        else if (CE)
                Q <= Q + 1;
end

always @(Q)
begin
        if (Q == TERMINAL_COUNT)
                TC <= 1;
        else
                TC <= 0;
end

always @(TC or CE)
begin
        CEO <= TC & CE;
end
```

CD4CE

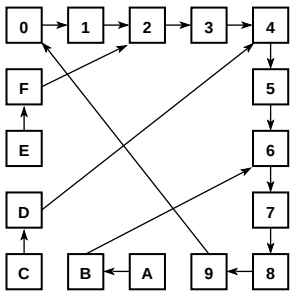
4-Bit Cascadable BCD Counter with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



CD4CE is a 4-bit (stage), asynchronous clearable, cascadable binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when clock enable (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, as shown in the following state diagram. For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse to the PRLD global net.

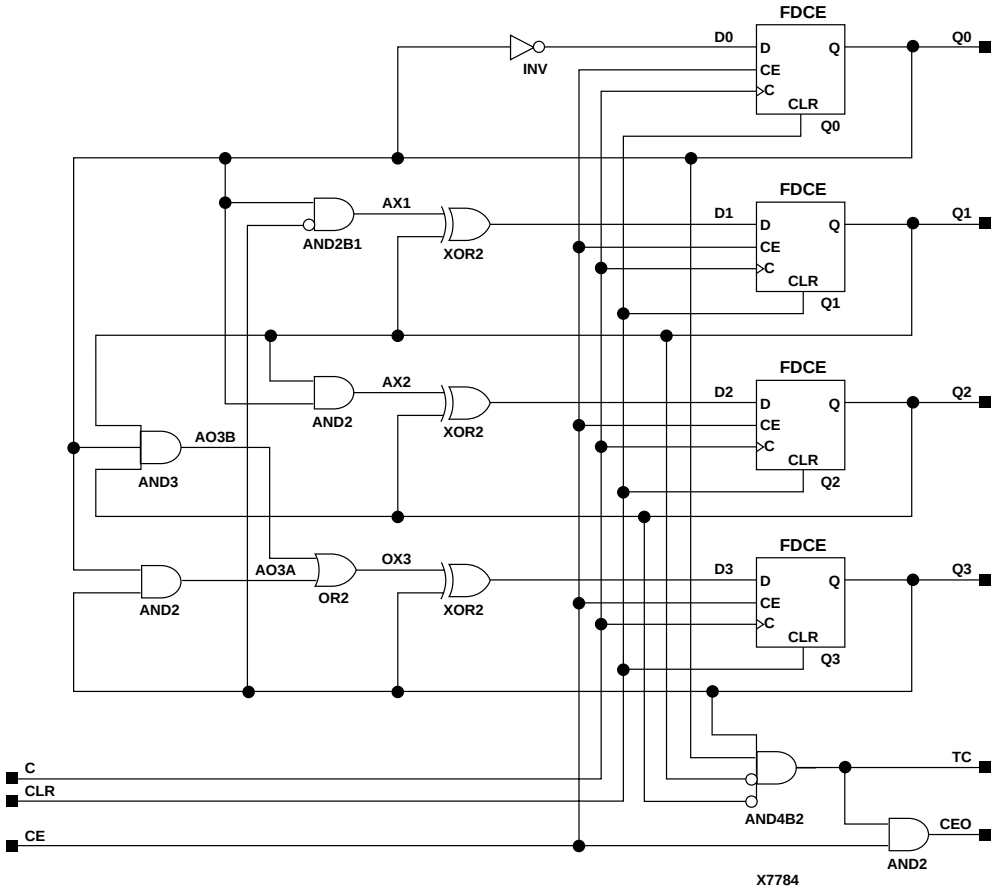
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

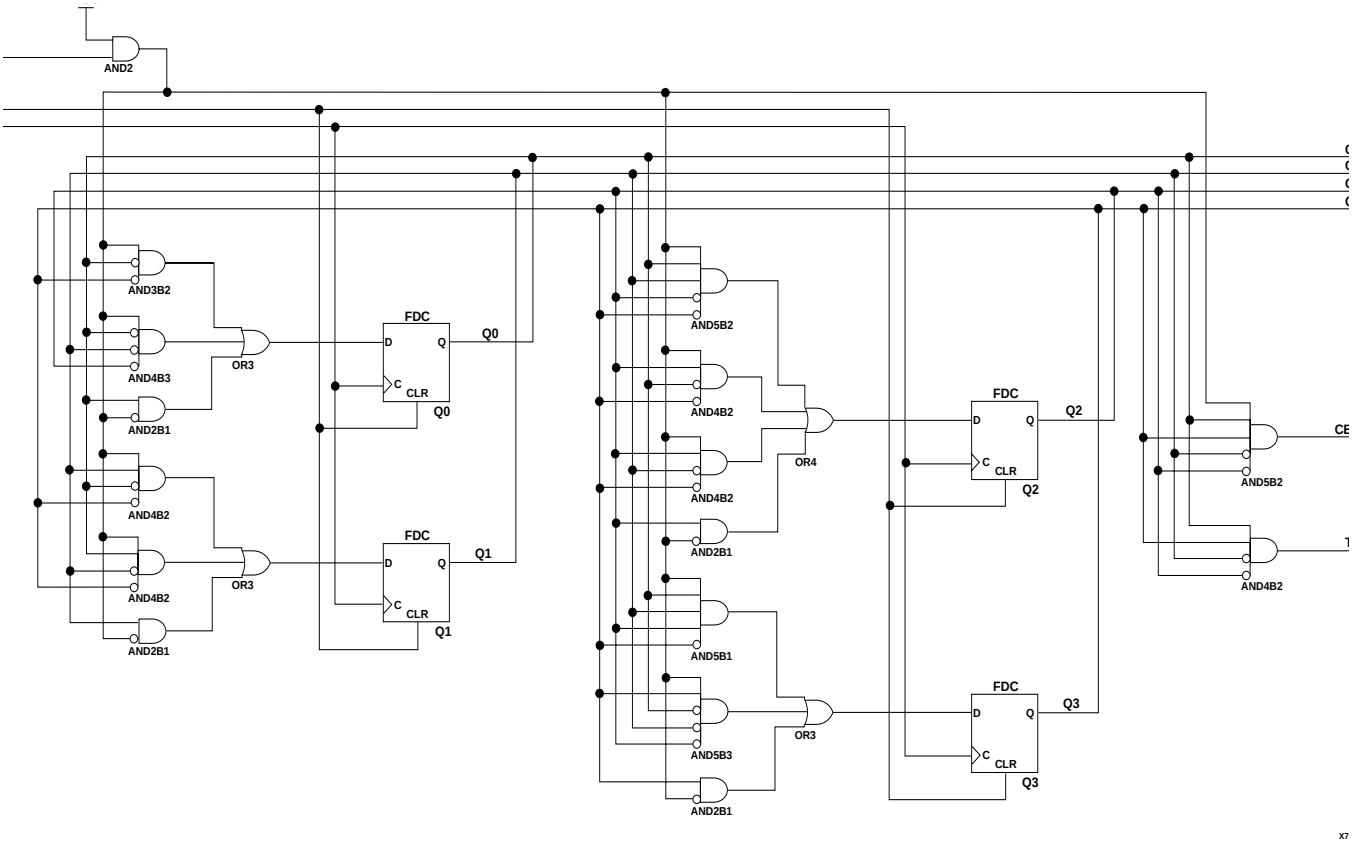
Inputs			Outputs					
CLR	CE	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

TC = Q3•!Q2•!Q1•Q0

CEO = TC•CE



CD4CE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CD4CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of cd4ce is

begin

 process (C, CLR)
 begin
  if (CLR = ’1’) then
   Q <= "0000";
  elsif (C’event and C = ’1’) then
   if (CE = ’1’) then
    if (Q = "1001") then
     Q <= "0000";
    elsif (Q = "1011") then
     Q <= "0110";
    elsif (Q = "1101") then
     Q <= "0100";
    elsif (Q = "1111") then
     Q <= "0010";
    else
     Q <= Q + 1;
```

```
      end if;
    end if;
  end if;
 end process;

 process (Q)
 begin
  if (Q = "1001") then
   TC_INT <= '1';
  else
   TC_INT <= '0';
  end if;
 end process;

 TC <= TC;

end Behavioral;
```

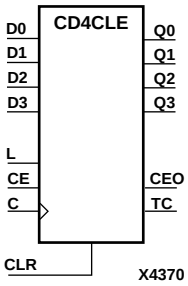
Verilog Inference Code

```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
  begin
  if (Q == 4'b1001)
   Q <= 4'b0000;
  else if (Q == 4'b1011)
   Q <= 4'b0110;
  else if (Q == 4'b1101)
   Q <= 4'b0100;
  else if (Q == 4'b1111)
   Q <= 4'b0010;
  else
   Q <= Q + 1;
 end
end

always @ (Q)
begin
 if (Q == 4'b1001)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC && CE;
end
```

CD4CLE

4-Bit Loadable Cascadable BCD Counter with Clock Enable and Asynchronous Clear

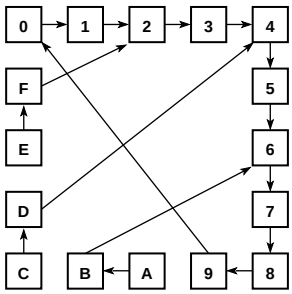
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



CD4CLE is a 4-bit (stage), synchronously loadable, asynchronously clearable, binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition. The Q outputs increment when clock enable input (CE) is High during the Low- to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, as shown in the following state diagram.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

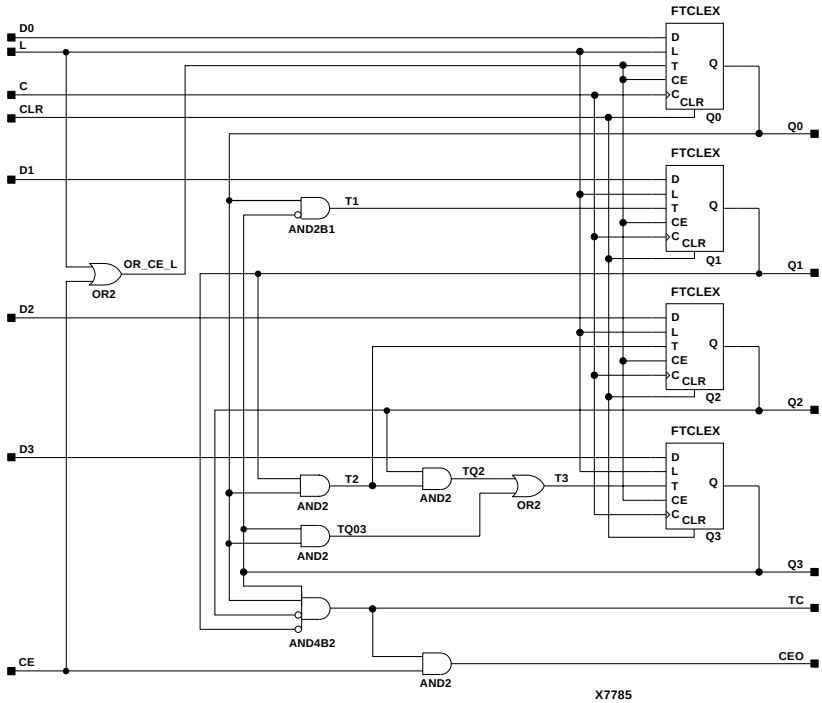
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs					
CLR	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	X	X	0	0	0	0	0	0
0	1	X	D3 – D0	↑	d3	d2	d1	d0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

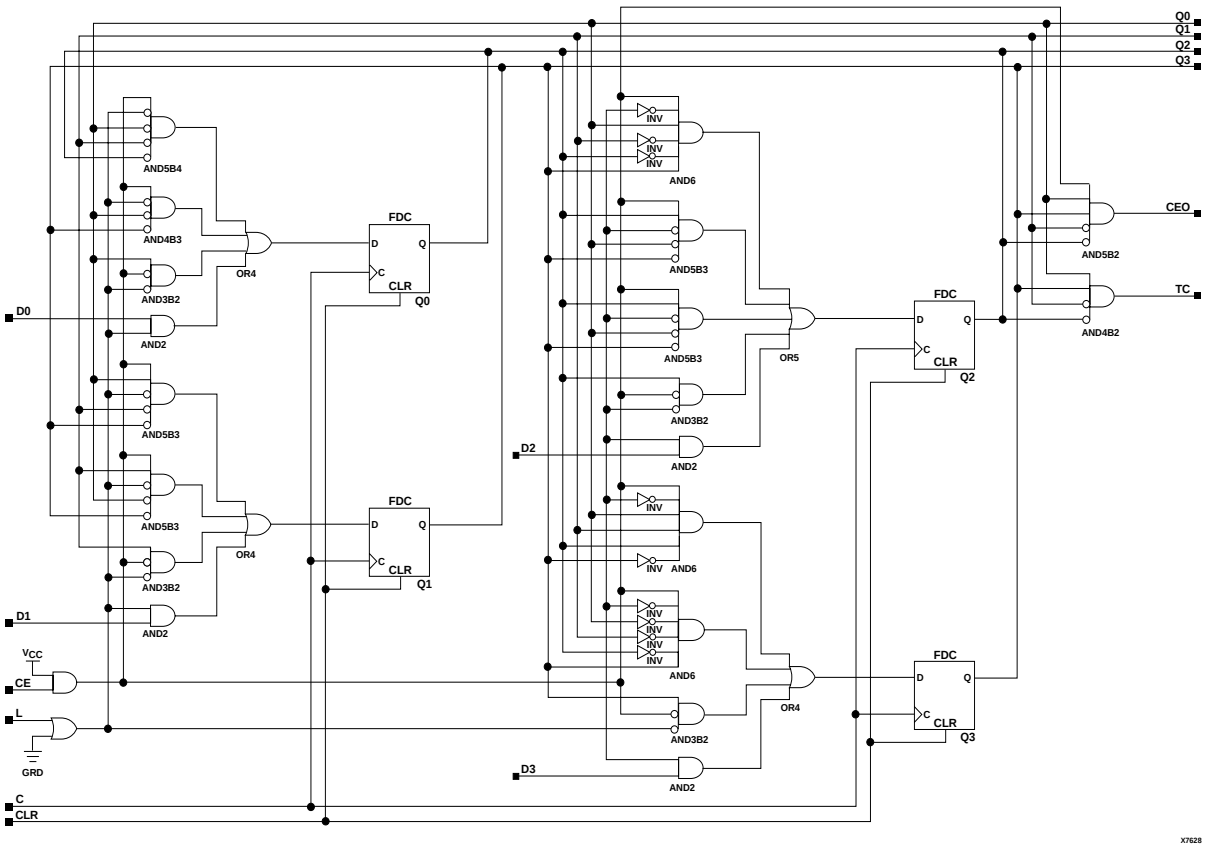
d = state of referenced input one setup time prior to active clock transition

TC = Q3•!Q2•!Q1•Q0

CEO = TC•CE



CD4CLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CD4CLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of cd4cle is

begin

 process (C, CLR)
 begin
  if (CLR = ’1’) then
   Q <= "0000";
  elsif (C’event and C = ’1’) then
   if (L = ’1’) then
    Q <= D;
   elsif (CE = ’1’) then
    if (Q = "1001") then
     Q <= "0000";
    elsif (Q = "1011") then
     Q <= "0110";
    elsif (Q = "1101") then
     Q <= "0100";
    elsif (Q = "1111") then
```

```
    Q <= "0010";
   else
    Q <= Q + 1;
     end if;
   end if;
 end if;
end process;

process (Q)
begin
 if (Q = "1001") then
  TC_INT <= '1';
 else
  TC_INT <= '0';
 end if;
end process;

CEO <= TC and CE;

end Behavioral;
```

Verilog Inference Code

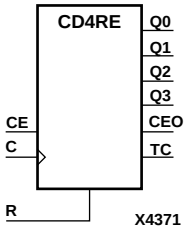
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  begin
  if (Q == 4'b1001)
   Q <= 4'b0000;
  else if (Q == 4'b1011)
   Q <= 4'b0110;
  else if (Q == 4'b1101)
   Q <= 4'b0100;
  else if (Q == 4'b1111)
   Q <= 4'b0010;
  else
   Q <= Q + 1;
 end
end

always @ (Q)
begin
 if (Q == 4'b1001)
  TC <= 1;
 else
  TC <= 0;
end

always @(TC or CE)
begin
```

```
 CEO <= TC && CE;
end
```

CD4RE

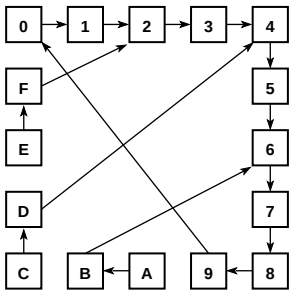
4-Bit Cascadable BCD Counter with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



CD4RE is a 4-bit (stage), synchronous resettable, cascadable binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, as shown in the following state diagram. For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

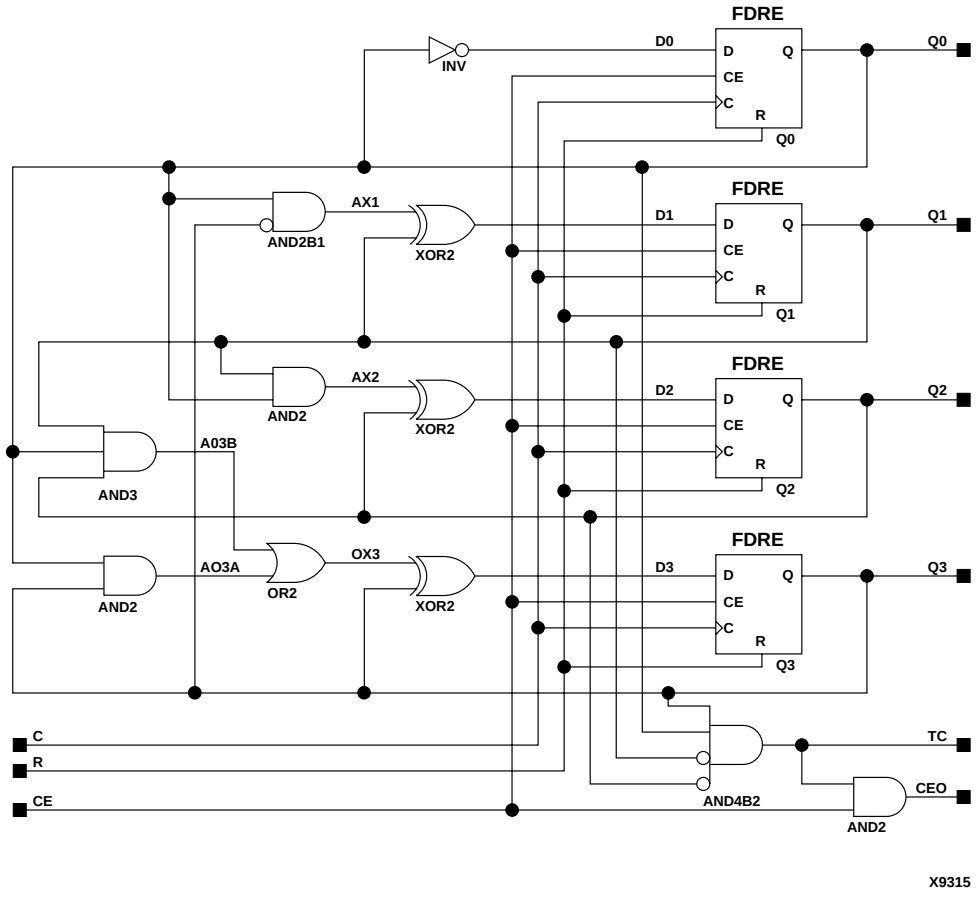
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

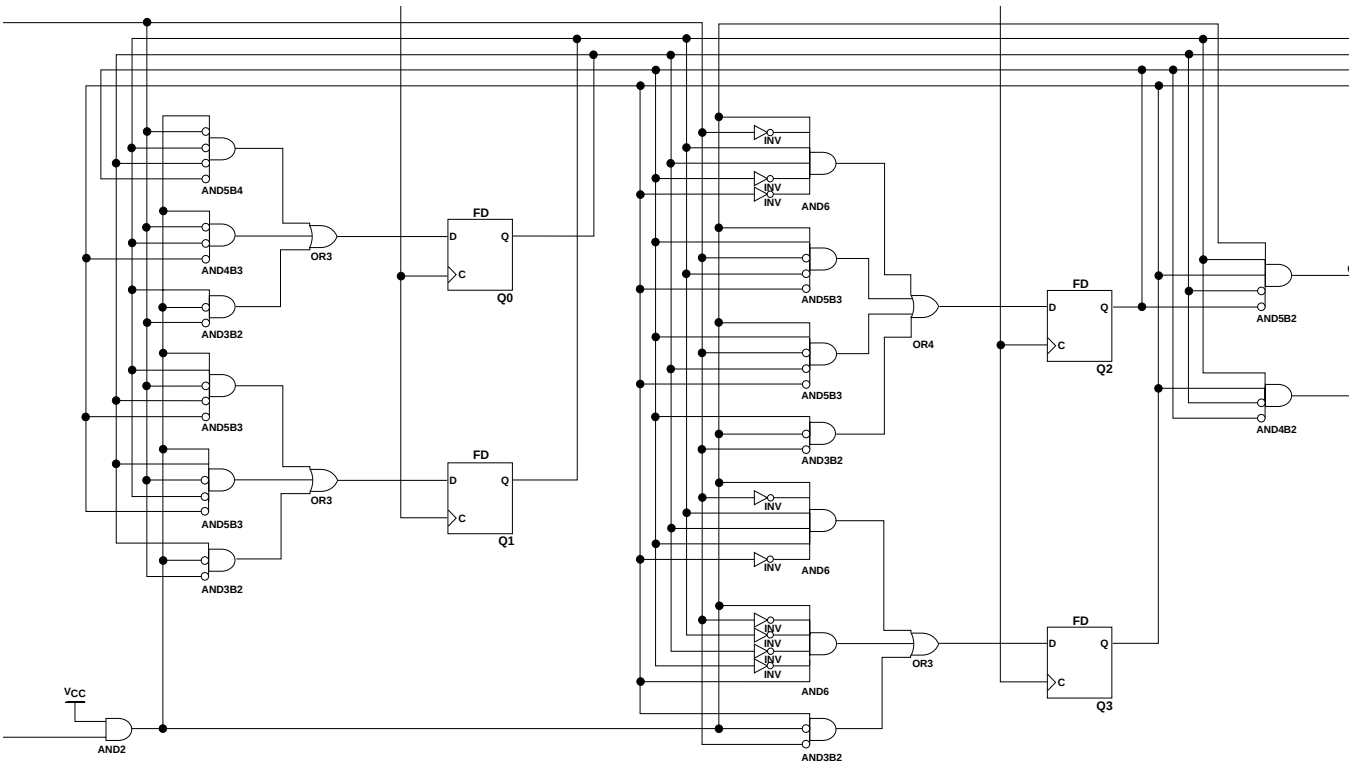
Inputs			Outputs					
R	CE	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	↑	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

TC = Q3•!Q2•!Q1•Q0

CEO = TC•CE



CD4RE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CD4RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of cd4re is

begin
 process (C)
 begin
  if (C'event and C = '1') then
   if (R = '1') then
    Q <= "0000";
   elsif (CE = '1') then
    if (Q = "1001") then
     Q <= "0000";
    elsif (Q = "1011") then
     Q <= "0110";
    elsif (Q = "1101") then
     Q <= "0100";
    elsif (Q = "1111") then
     Q <= "0010";
    else
     Q <= Q + 1;
      end if;
    end if;
```

```
  end if;
 end process;

 process (Q)
 begin
  if (Q = "1001") then
   TC <= '1';
  else
   TC <= '0';
  end if;
 end process;

 CEO <= TC and CE;

end Behavioral;
```

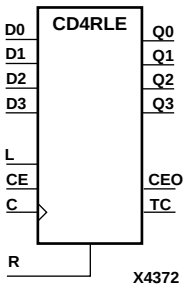
Verilog Inference Code

```
always @ (posedge C)
begin
 if (R)
  Q <= 0;
 else if (CE)
  begin
  if (Q == 4'b1001)
   Q <= 4'b0000;
  else if (Q == 4'b1011)
   Q <= 4'b0110;
  else if (Q == 4'b1101)
   Q <= 4'b0100;
  else if (Q == 4'b1111)
   Q <= 4'b0010;
  else
   Q <= Q + 1;
 end
end

always @ (Q)
begin
 if (Q == 4'b1001)
  TC <= 1;
 else
  TC <= 0;
end

always @(TC or CE)
begin
 CEO <= TC && CE;
end
```

CD4RLE

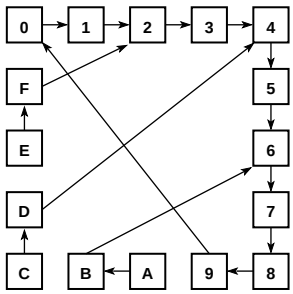
4-Bit Loadable Cascadable BCD Counter with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



CD4RLE is a 4-bit (stage), synchronous loadable, resettable, binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, as shown in the following state diagram. For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

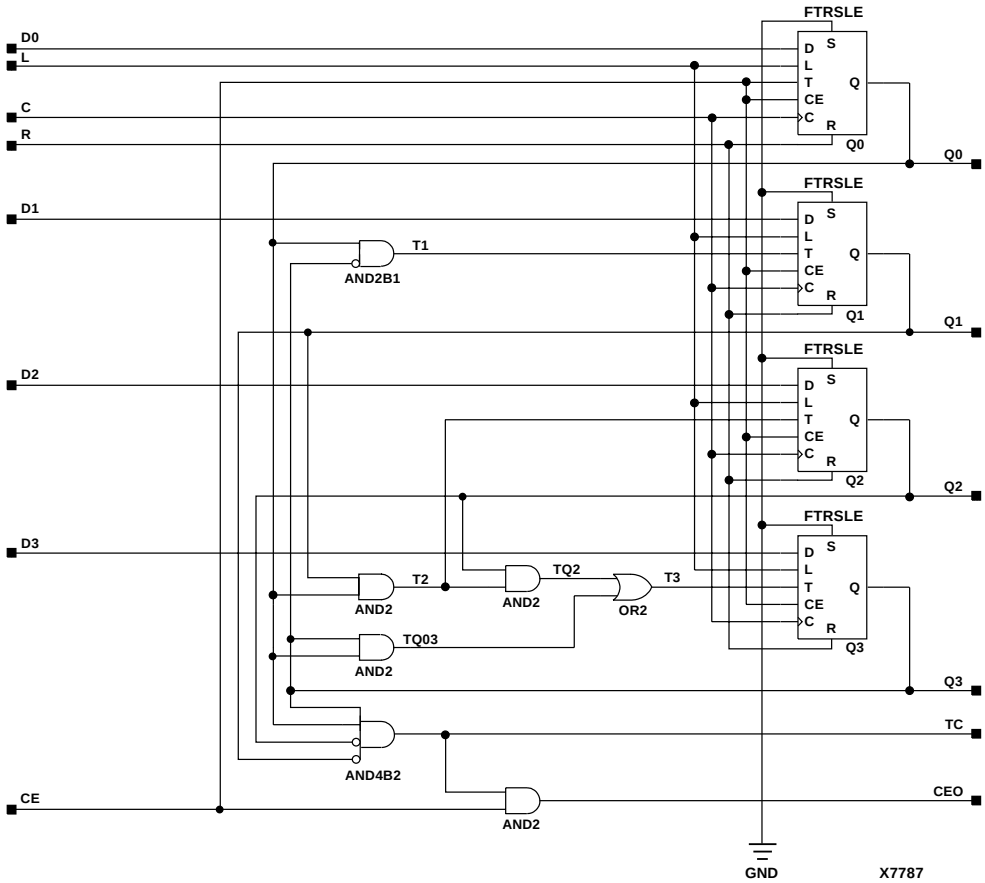
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs					
R	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	X	↑	0	0	0	0	0	0
0	1	X	D3 – D0	↑	d3	d2	d1	d0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

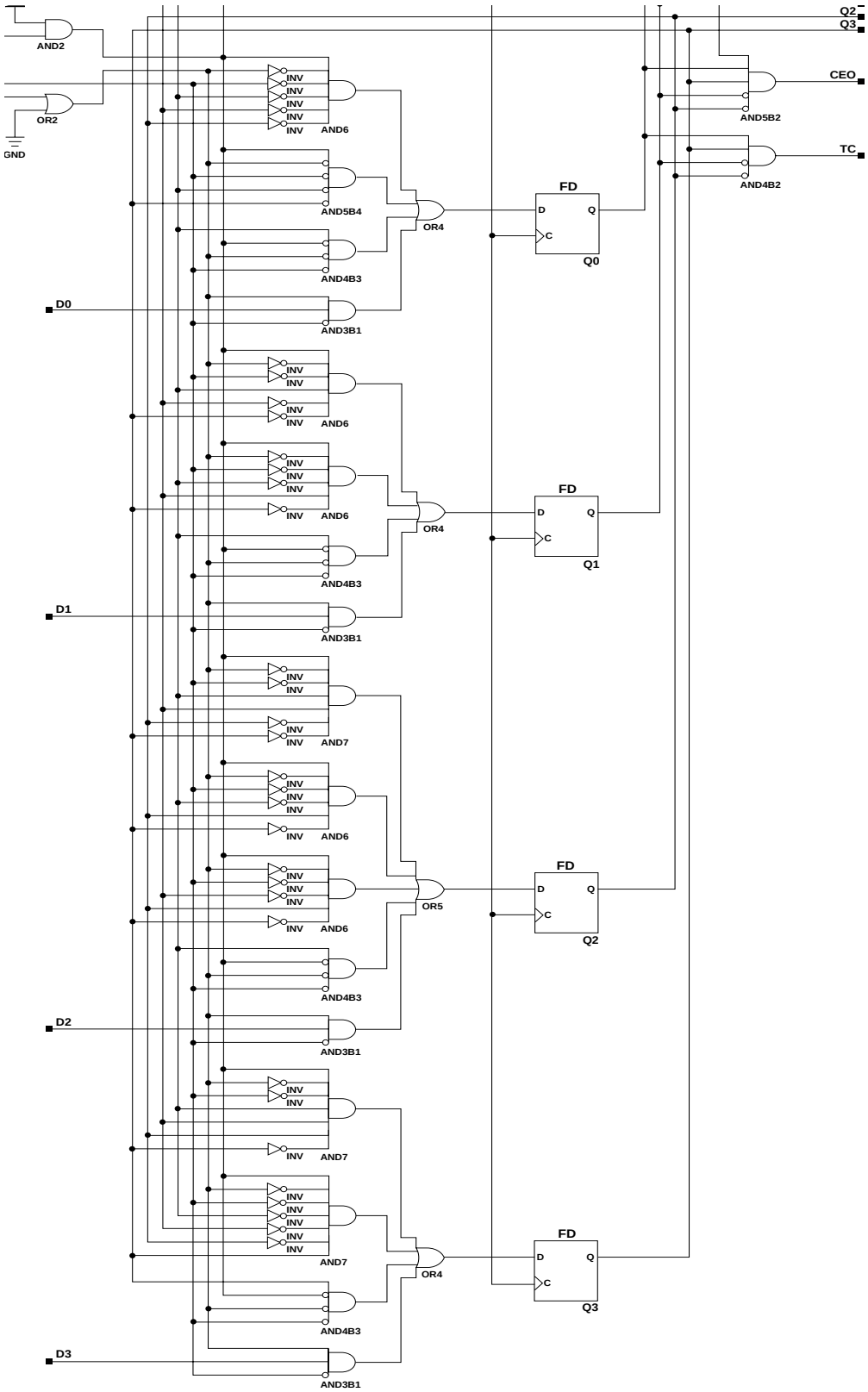
d = state of referenced input one setup time prior to active clock transition

TC = Q3•!Q2•!Q1•Q0

CEO = TC•CE



CD4RLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CD4RLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cd4rle is

begin

 process (C)
 begin
  if (C'event and C = '1') then
   if (R = '1') then
    Q <= "0000";
   elsif (L = '1') then
    Q <= D;
   elsif (CE = '1') then
    if (Q = "1001") then
     Q <= "0000";
    elsif (Q = "1011") then
     Q <= "0110";
    elsif (Q = "1101") then
     Q <= "0100";
    elsif (Q = "1111") then
     Q <= "0010";
    else
     Q <= Q + 1;
      end if;
    end if;
  end if;
 end process;

 process (Q)
 begin
  if (Q = "1001") then
   TC <= '1';
  else
   TC <= '0';
  end if;
 end process;

 CEO <= TC and CE;

end Behavioral;
```

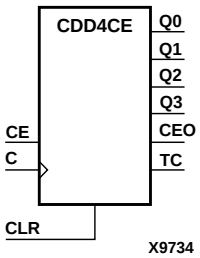
Verilog Inference Code

```
always @ (posedge C)
begin
 if (R)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  begin
  if (Q == 4'b1001)
   Q <= 4'b0000;
  else if (Q == 4'b1011)
   Q <= 4'b0110;
  else if (Q == 4'b1101)
   Q <= 4'b0100;
  else if (Q == 4'b1111)
   Q <= 4'b0010;
  else
   Q <= Q + 1;
 end
end

always @ (Q)
begin
 if (Q == 4'b1001)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC && CE;
end
```

CDD4CE

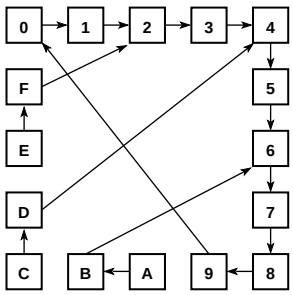
4-Bit Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



CDD4CE is a 4-bit (stage), asynchronous clearable, cascadable dual edge triggered Binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when clock enable (CE) is High during the Low-to-High and High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles, as shown in the following state diagram. The counter resets to zero or recovers within the first clock cycle.



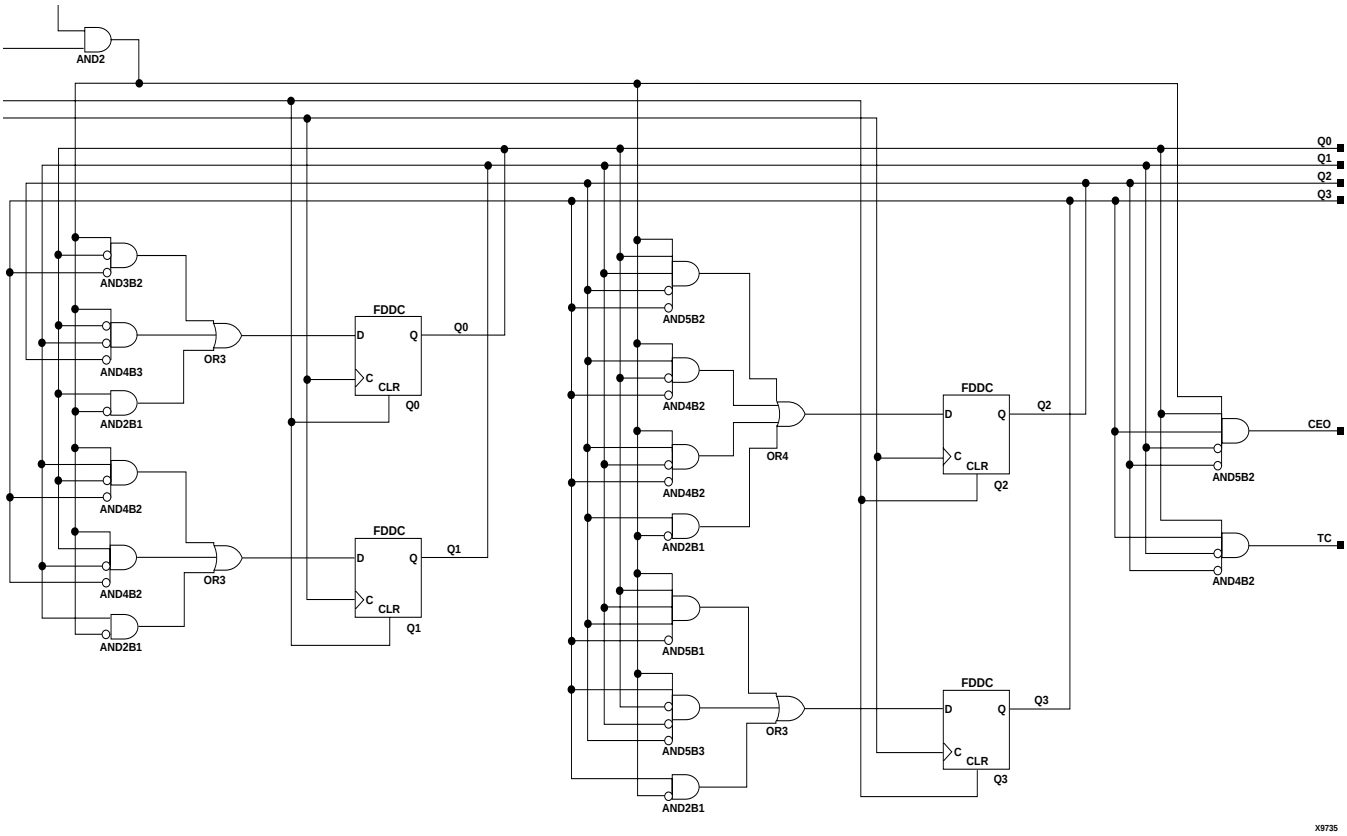
Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse to the PRLD global net.

Inputs			Outputs					
CLR	**CE**	**C**	**Q3**	**Q2**	**Q1**	**Q0**	**TC**	**CEO**
1	X	X	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	1	↓	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

TC = Q3 • !Q2 • !Q1 • Q0

CEO = TC • CE



CDD4CE Implementation CoolRunner-II

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cdd4ce is

begin

 process (C, CLR)
 begin
  if (CLR = '1') then
   Q <= "0000";
  elsif (C'event) then
   if (CE = '1') then
    if (Q = "1001") then
     Q <= "0000";
    elsif (Q = "1011") then
     Q <= "0110";
    elsif (Q = "1101") then
     Q <= "0100";
    elsif (Q = "1111") then
     Q <= "0010";
    else
     Q <= Q + 1;
      end if;
    end if;
  end if;
 end process;

 process (Q)
 begin
  if (Q = "1001") then
   TC <= '1';
  else
   TC <= '0';
  end if;
 end process;

 CEO <= TC and CE;

end Behavioral;
```

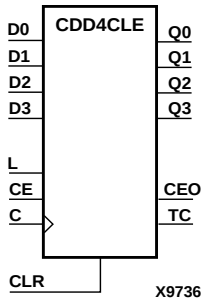
Verilog Inference Code

```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
  begin
  if (Q == 4'b1001)
   Q <= 4'b0000;
  else if (Q == 4'b1011)
   Q <= 4'b0110;
  else if (Q == 4'b1101)
   Q <= 4'b0100;
  else if (Q == 4'b1111)
   Q <= 4'b0010;
  else
   Q <= Q + 1;
 end
end

always @ (Q)
begin
 if (Q == 4'b1001)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC && CE;
end
```

CDD4CLE

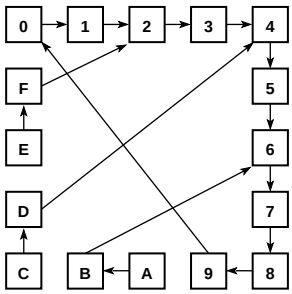
4-Bit Loadable Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



CDD4CLE is a 4-bit (stage), synchronously loadable, asynchronously clearable, dual edge triggered Binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High and High-to-Low clock (C) transitions. The Q outputs increment when clock enable input (CE) is High during the Low- to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles, as shown in the following state diagram. The counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

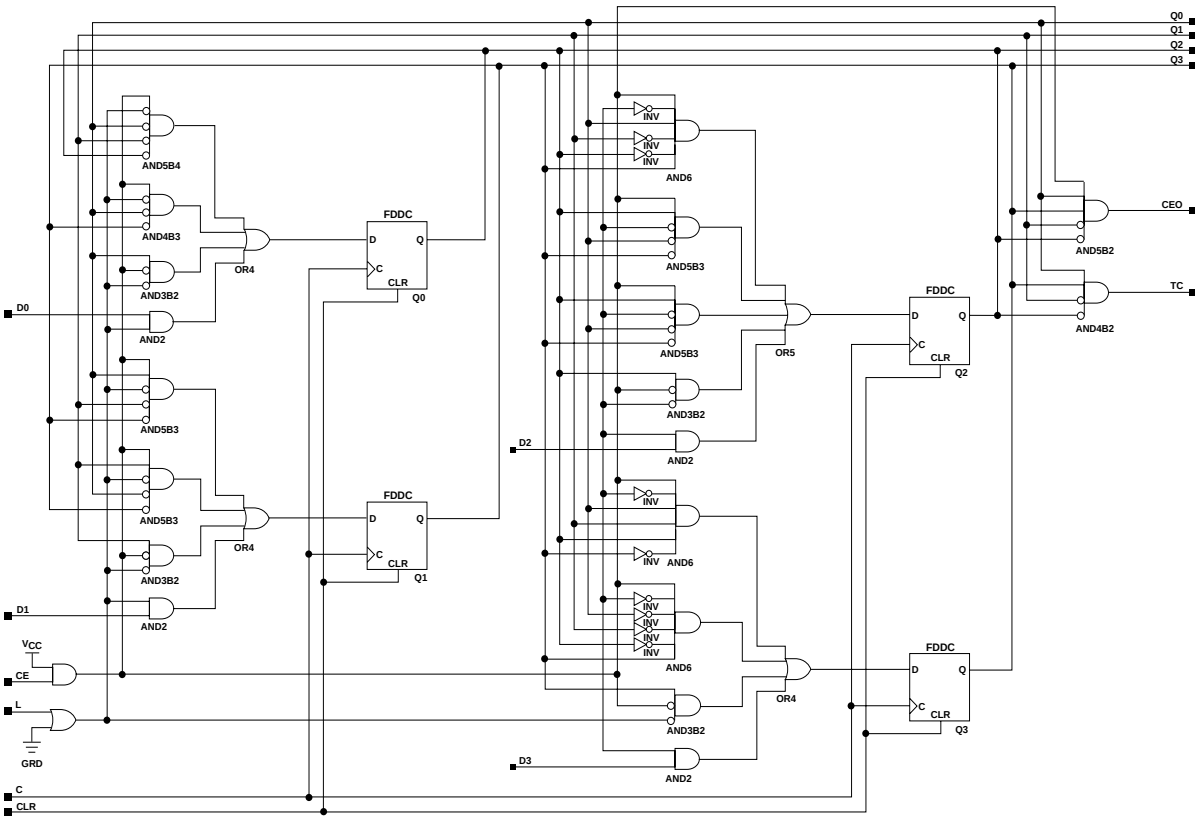
The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs					
CLR	**L**	**CE**	**D3 – D0**	**C**	**Q3**	**Q2**	**Q1**	**Q0**	**TC**	**CEO**
1	X	X	X	X	0	0	0	0	0	0
0	1	X	D3 – D0	↑	d3	d2	d1	d0	TC	CEO
0	1	X	D3 – D0	↓	d3	d2	d1	d0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	1	X	↓	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

d = state of referenced input one setup time prior to active clock transition

TC = Q3•!Q2•!Q1•Q0

CEO = TC•CE



CDD4CLE Implementation CoolRunner-II

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cdd4cle is

begin

 process (C, CLR)
 begin
  if (CLR = '1') then
   Q <= "0000";
  elsif (C'event) then
   if (L = '1') then
    Q <= D;
   elsif (CE = '1') then
    if (Q = "1001") then
     Q <= "0000";
    elsif (Q = "1011") then
     Q <= "0110";
    elsif (Q = "1101") then
     Q <= "0100";
    elsif (Q = "1111") then
     Q <= "0010";
    else
     Q <= Q + 1;
      end if;
    end if;
  end if;
 end process;

 process (Q)
 begin
  if (Q = "1001") then
   TC <= '1';
  else
   TC <= '0';
  end if;
 end process;

 CEO <= TC and CE;

end Behavioral;
```

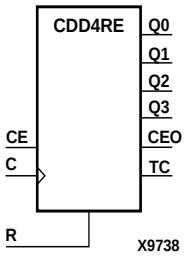
Verilog Inference Code

```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  begin
  if (Q == 4'b1001)
   Q <= 4'b0000;
  else if (Q == 4'b1011)
   Q <= 4'b0110;
  else if (Q == 4'b1101)
   Q <= 4'b0100;
  else if (Q == 4'b1111)
   Q <= 4'b0010;
  else
   Q <= Q + 1;
 end
end

always @ (Q)
begin
 if (Q == 4'b1001)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC && CE;
end
```

CDD4RE

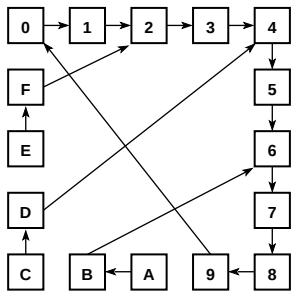
4-Bit Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



CDD4RE is a 4-bit (stage), synchronous resettable, cascadable dual edge triggered binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High or High-to-Low clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles, as shown in the following state diagram. The counter resets to zero or recovers within the first clock cycle.



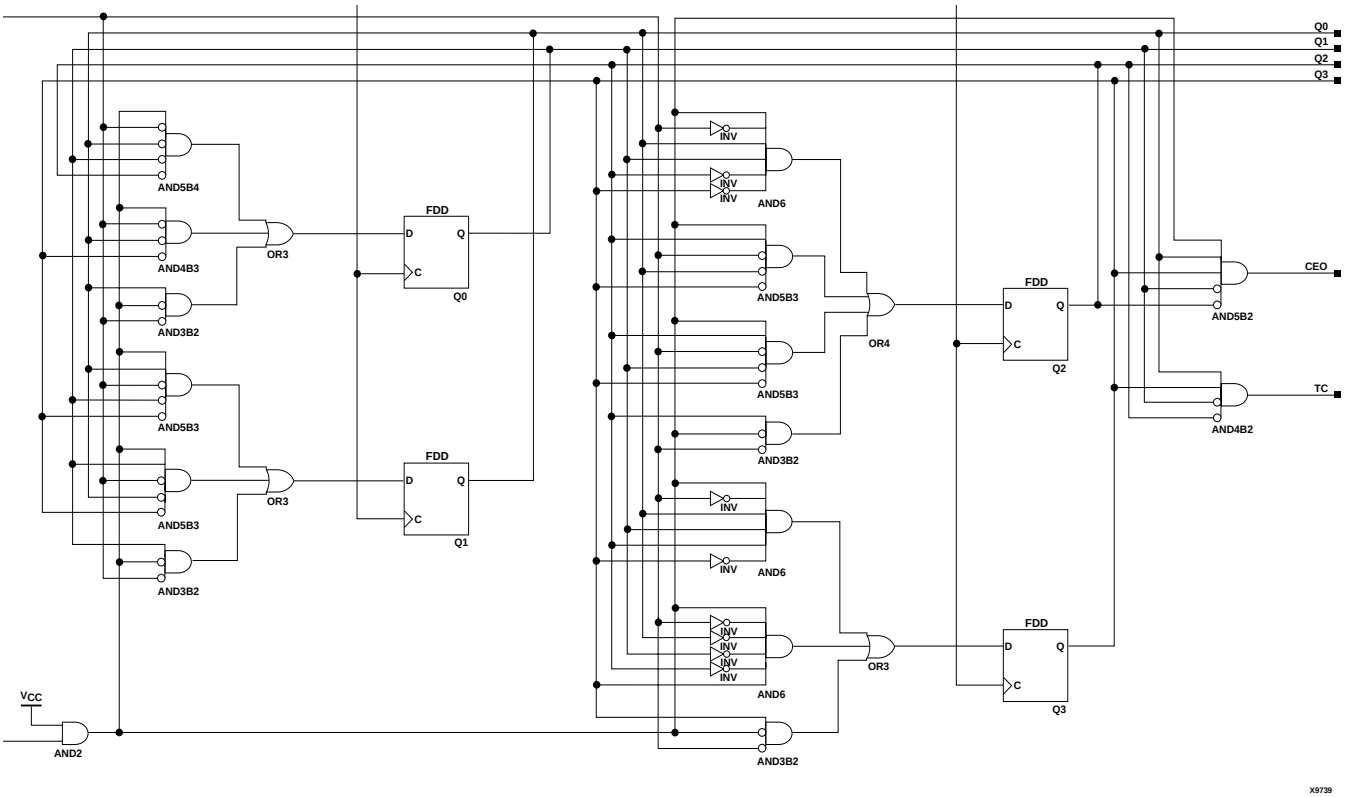
Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs			Outputs					
R	**CE**	**C**	**Q3**	**Q2**	**Q1**	**Q0**	**TC**	**CEO**
1	X	↑	0	0	0	0	0	0
1	X	↓	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	1	↓	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

TC = Q3 • !Q2 • !Q1 • Q0

CEO = TC • CE



CDD4RE Implementation CoolRunner-II

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cdd4re is

begin

 process (C)
 begin
  if (C'event) then
   if (R = '1') then
    Q <= "0000";
   elsif (CE = '1') then
    if (Q = "1001") then
     Q <= "0000";
    elsif (Q = "1011") then
     Q <= "0110";
    elsif (Q = "1101") then
     Q <= "0100";
    elsif (Q = "1111") then
     Q <= "0010";
    else
     Q <= Q + 1;
      end if;
    end if;
  end if;
 end process;

 process (Q)
 begin
  if (Q = "1001") then
   TC <= '1';
  else
   TC <= '0';
  end if;
 end process;

 CEO <= TC and CE;

end Behavioral;
```

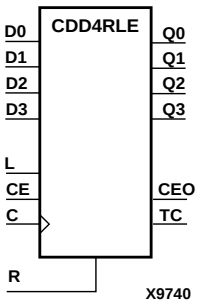
Verilog Inference Code

```
always @ (posedge C or negedge C)
begin
 if (R)
  Q <= 0;
 else if (CE)
  begin
  if (Q == 4'b1001)
   Q <= 4'b0000;
  else if (Q == 4'b1011)
   Q <= 4'b0110;
  else if (Q == 4'b1101)
   Q <= 4'b0100;
  else if (Q == 4'b1111)
   Q <= 4'b0010;
  else
   Q <= Q + 1;
 end
end

always @ (Q)
begin
 if (Q == 4'b1001)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC && CE;
end
```

CDD4RLE

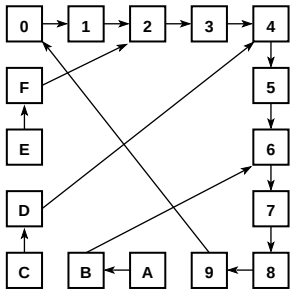
4-Bit Loadable Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



CDD4RLE is a 4-bit (stage), synchronous loadable, resettable, dual edge triggered binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High or High-to-Low clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High and High-to-Low clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles, as shown in the following state diagram. The counter resets to zero or recovers within the first clock cycle.



Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

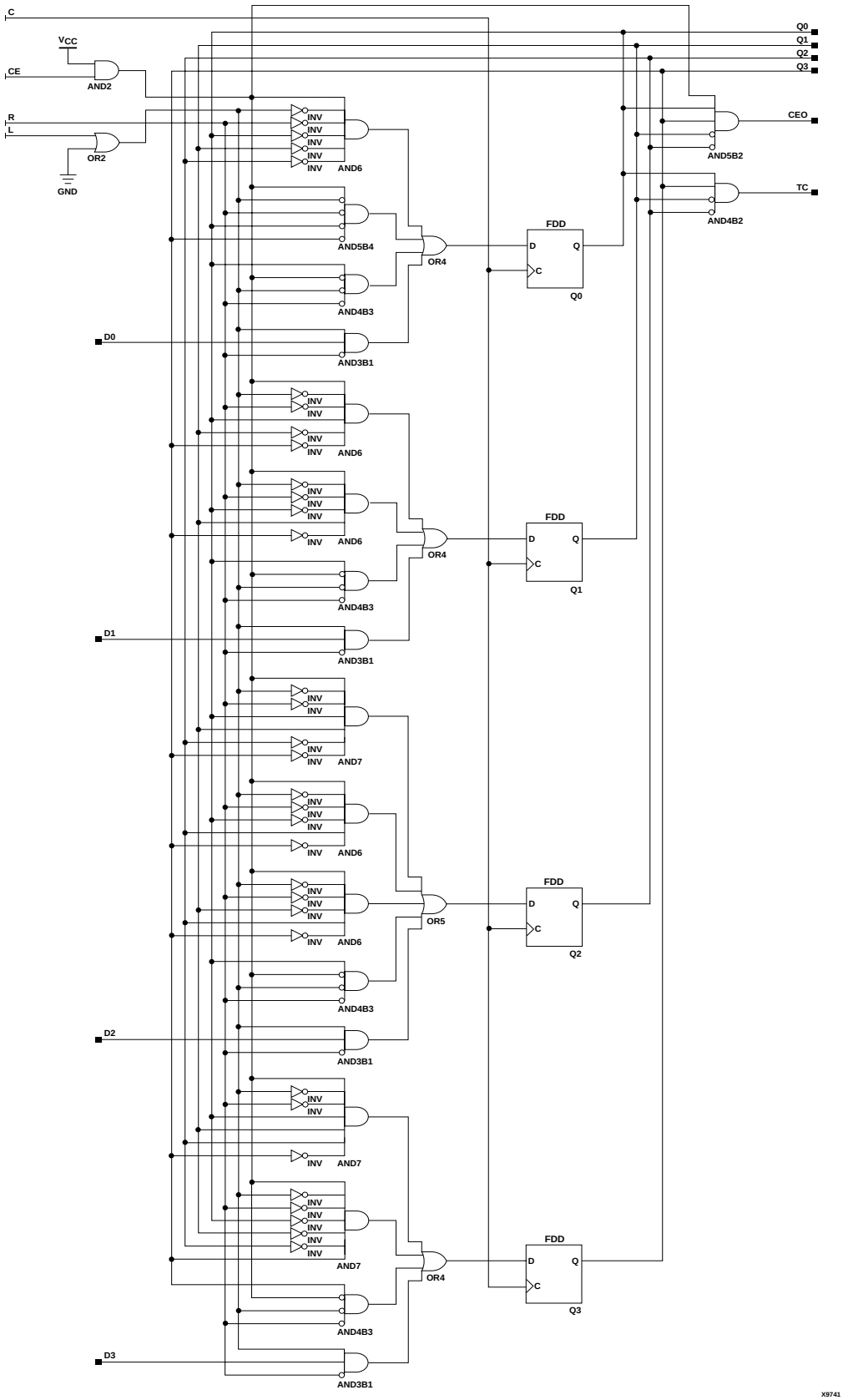
The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs					
R	**L**	**CE**	**D3 – D0**	**C**	**Q3**	**Q2**	**Q1**	**Q0**	**TC**	**CEO**
1	X	X	X	↑	0	0	0	0	0	0
1	X	X	X	↓	0	0	0	0	0	0
0	1	X	D3 – D0	↑	d3	d2	d1	d0	TC	CEO
0	1	X	D3 – D0	↓	d3	d2	d1	d0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	1	X	↓	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

d = state of referenced input one setup time prior to active clock transition

TC = Q3•!Q2•!Q1•Q0

CEO = TC•CE



CDD4RLE Implementation CoolRunner-II

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cdd4rle is

begin

 process (C)
 begin
  if (C'event) then
   if (R = '1') then
    Q <= "0000";
   elsif (L = '1') then
    Q <= D;
   elsif (CE = '1') then
    if (Q = "1001") then
     Q <= "0000";
    elsif (Q = "1011") then
     Q <= "0110";
    elsif (Q = "1101") then
     Q <= "0100";
    elsif (Q = "1111") then
     Q <= "0010";
    else
     Q <= Q + 1;
      end if;
    end if;
  end if;
 end process;

 process (Q)
 begin
  if (Q = "1001") then
   TC <= '1';
  else
   TC <= '0';
  end if;
 end process;

 CEO <= TC and CE;

end Behavioral;
```

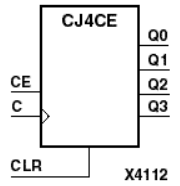
Verilog Inference Code

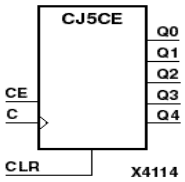
```
always @ (posedge C or negedge C)
begin
 if (R)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  begin
  if (Q == 4'b1001)
   Q <= 4'b0000;
  else if (Q == 4'b1011)
   Q <= 4'b0110;
  else if (Q == 4'b1101)
   Q <= 4'b0100;
  else if (Q == 4'b1111)
   Q <= 4'b0010;
  else
   Q <= Q + 1;
 end
end

always @ (Q)
begin
 if (Q == 4'b1001)
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC && CE;
end
```

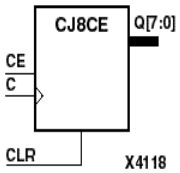
CJ4CE, CJ5CE, CJ8CE

4-, 5-, 8-Bit Johnson Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







CJ4CE, CJ5CE, and CJ8CE are clearable Johnson / shift counters. The asynchronous clear (CLR) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero, independent of clock (C) transitions. The counter increments (shifts Q0 to Q1, Q1 to Q2,and so forth) when the clock enable input (CE) is High during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.

For CJ4CE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operation. For CJ5CE, the Q4 output is inverted and fed back to input Q0. For CJ8CE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

CJ4CE Truth Table

Inputs			Outputs			
CLR	CE	C	Q0	Q1	Q2	Q3
1	X	X	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	!q3	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

CJ5CE Truth Table

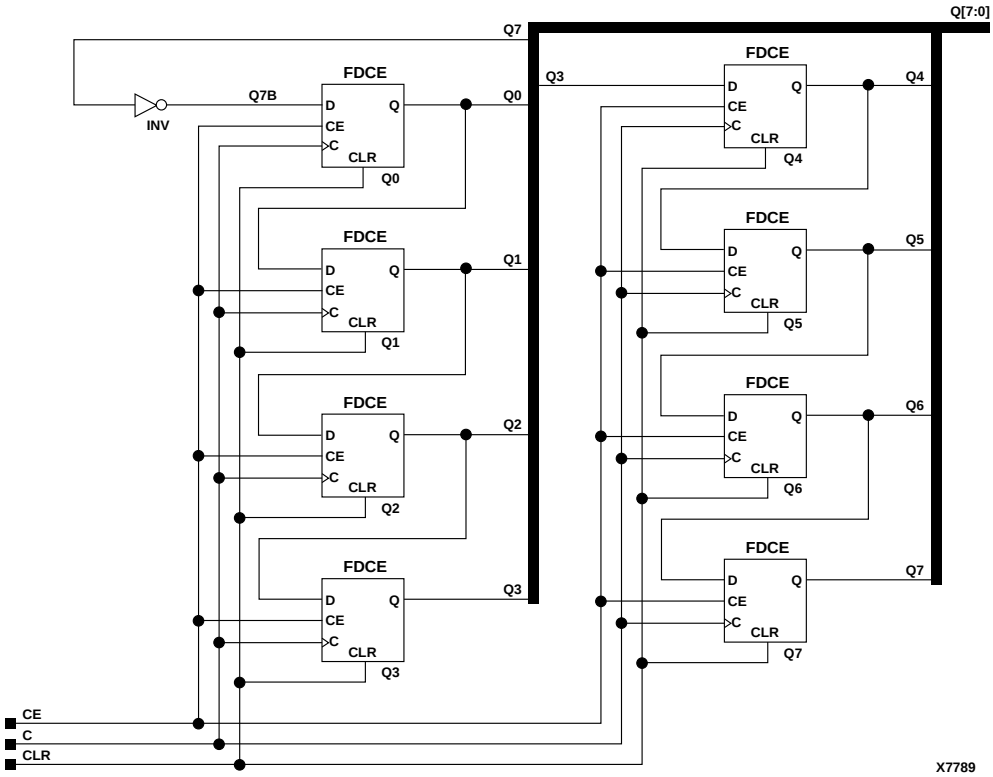
Inputs			Outputs				
CLR	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	X	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	!q4	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

CJ8CE Truth Table

Inputs			Outputs	
CLR	CE	C	Q0	Q1 – Q7
1	X	X	0	0
0	0	X	No Chg	No Chg
0	1	↑	!q7	q0 – q6

q = state of referenced output one setup time prior to active clock transition



CJ8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cj4ce is

begin

process (C, CLR)
begin
 if (CLR = '1') then
  Q <= (others => '0');
 elsif (C'event and C = '1') then
  if (CE = '1') then
   Q(0) <= not Q(WIDTH-1);
   Q(WIDTH-1 downto 1) <= Q(WIDTH-2 downto 0);
  end if;
 end if;
end process;

end Behavioral;
```

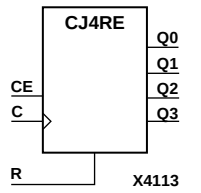
Verilog Inference Code

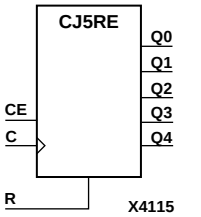
```
always @ (posedge C or posedge CLR)
  begin
   if (CLR)
    Q <= 0;
   else
    begin
     if (CE)
      begin
       Q[0] <= !Q[WIDTH-1];
       Q[WIDTH-1:1] <= Q[WIDTH-2:0];
      end
    end
  end
```

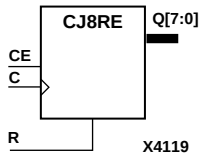
CJ4RE, CJ5RE, CJ8RE

4-, 5-, 8-Bit Johnson Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







CJ4RE, CJ5RE, and CJ8RE are resettable Johnson / shift counters. The synchronous reset (R) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero during the Low-to-High clock (C) transition. The counter increments (shifts Q0 to Q1, Q1 to Q2, and so forth) when the clock enable input (CE) is High during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.

For CJ4RE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operations. For CJ5RE, the Q4 output is inverted and fed back to input Q0. For CJ8RE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

CJ4RE Truth Table

Inputs			Outputs			
R	CE	C	Q0	Q1	Q2	Q3
1	X	↑	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q3}$	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

CJ5RE Truth Table

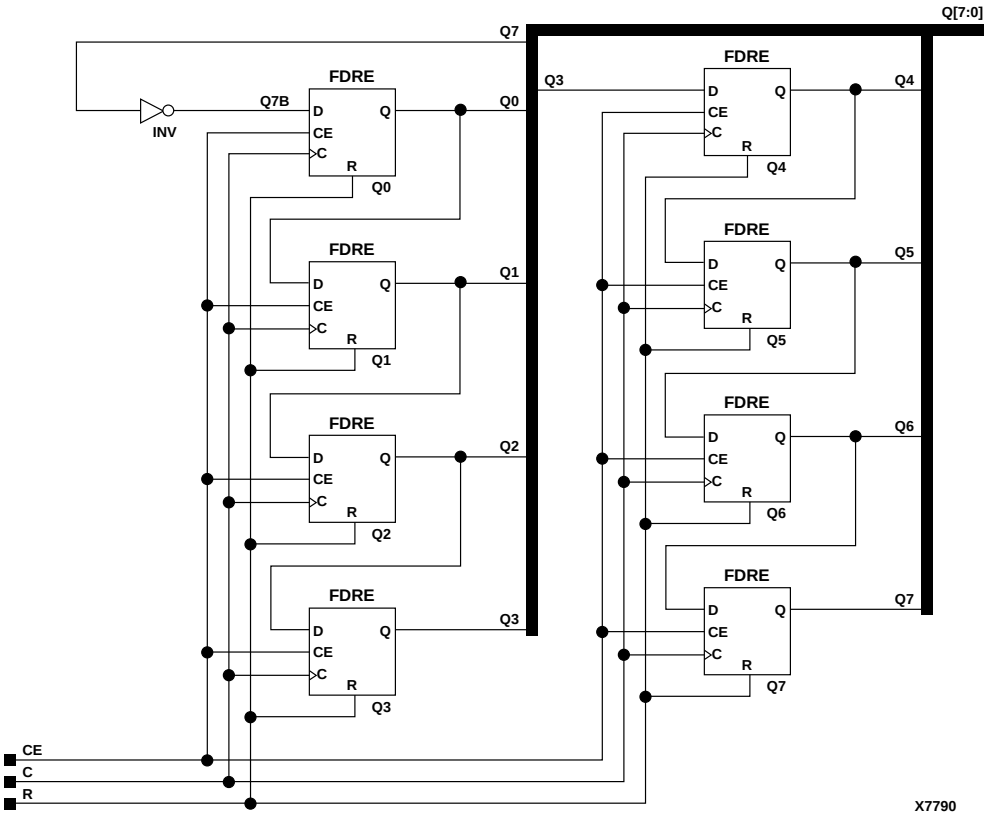
Inputs			Outputs				
R	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	↑	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q4}$	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

CJ8RE Truth Table

Inputs			Outputs	
R	CE	C	Q0	Q1 – Q7
1	X	↑	0	0
0	0	X	No Chg	No Chg
0	1	↑	$\overline{q7}$	q0 – q6

q = state of referenced output one setup time prior to active clock transition



CJ8RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cj4re is

begin

process (C)
begin
 if (C'event and C = '1') then
  if (R = '1') then
   Q <= (others => '0');
  elsif (CE = '1') then
   Q(0) <= not Q(WIDTH-1);
   Q(WIDTH-1 downto 1) <= Q(WIDTH-2 downto 0);
  end if;
 end if;
end process;

end Behavioral;
```

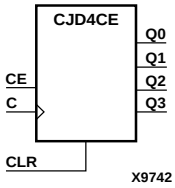
Verilog Inference Code

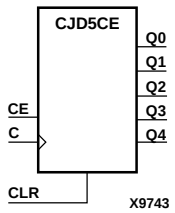
```
always @ (posedge C)
  begin
   if (R)
    Q <= 0;
   else
    begin
     if (CE)
      begin
       Q[0] <= !Q[WIDTH-1];
       Q[WIDTH-1:1] <= Q[WIDTH-2:0];
      end
    end
  end
```

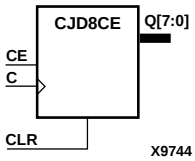
CJD4CE, CJD5CE, CJD8CE

4-, 5-, 8-Bit Dual Edge Triggered Johnson Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







CJD4CE, CJD5CE, and CJD8CE are dual edge triggered clearable Johnson / shift counters. The asynchronous clear (CLR) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero, independent of clock (C) transitions. The counter increments (shifts Q0 to Q1, Q1 to Q2,and so forth) when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock transition. Clock transitions are ignored when CE is Low.

For CJD4CE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operations. For CJD5CE, the Q4 output is inverted and fed back to input Q0. For CJD8CE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

CJD4CE Truth Table

Inputs			Outputs			
CLR	CE	C	Q0	Q1	Q2	Q3
1	X	X	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	!q3	q0	q1	q2
0	1	↓	!q3	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

CJD5CE Truth Table

Inputs			Outputs				
CLR	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	X	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	!q4	q0	q1	q2	q3
0	1	↓	!q4	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

CJD8CE Truth Table

Inputs			Outputs	
CLR	CE	C	Q0	Q1 – Q7
1	X	X	0	0
0	0	X	No Chg	No Chg
0	1	↑	!q7	q0 – q6
0	1	↓	!q7	q0 – q6

q = state of referenced output one setup time prior to active clock transition

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cjd4ce is

begin

process (C, CLR)
begin
 if (CLR = '1') then
  Q <= (others => '0');
 elsif (C'event) then
  if (CE = '1') then
   Q(0) <= not Q(WIDTH-1);
   Q(WIDTH-1 downto 1) <= Q(WIDTH-2 downto 0);
  end if;
 end if;
end process;

end Behavioral;
```

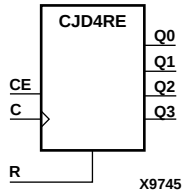
Verilog Inference Code

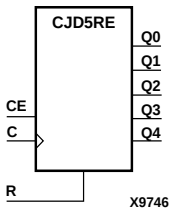
```
always @ (posedge C or negedge C or posedge CLR)
  begin
   if (CLR)
    Q_INT <= 0;
   else
    begin
     if (CE)
      begin
       Q_INT[0] <= !Q_INT[WIDTH-1];
       Q_INT[WIDTH-1:1] <= Q_INT[WIDTH-2:0];
      end
    end
  end

 always @ (Q_INT)
  begin
   Q <= Q_INT;
  end
```

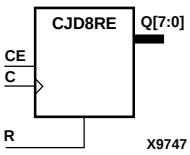
CJD4RE, CJD5RE, CJD8RE

4-, 5-, 8-Bit Dual Edge Triggered Johnson Counters with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







CJD4RE, CJD5RE, and CJD8RE are resettable dual edge triggered Johnson / shift counters. The synchronous reset (R) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero during the Low-to-High and High-to-Low clock (C) transition. The counter increments (shifts Q0 to Q1, Q1 to Q2, and so forth) when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock transition. Clock transitions are ignored when CE is Low.

For CJD4RE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operations. For CJD5RE, the Q4 output is inverted and fed back to input Q0. For CJD8RE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

CJD4RE Truth Table

Inputs			Outputs			
R	CE	C	Q0	Q1	Q2	Q3
1	X	↑	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q3}$	q0	q1	q2
0	1	↓	$\overline{q3}$	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

CJD5RE Truth Table

Inputs			Outputs				
R	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	↑	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q4}$	q0	q1	q2	q3
0	1	↓	$\overline{q4}$	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

CJD8RE Truth Table

Inputs			Outputs	
R	CE	C	Q0	Q1 – Q7
1	X	↑	0	0
1	X	↓	0	0
0	0	X	No Chg	No Chg
0	1	↑	$\overline{q7}$	q0 – q6
0	1	↓	$\overline{q7}$	q0 – q6

q = state of referenced output one setup time prior to active clock transition

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of cjd4re is

begin

process (C)
begin
 if (C'event) then
  if (R = '1') then
   Q <= (others => '0');
  elsif (CE = '1') then
   Q(0) <= not Q(WIDTH-1);
   Q(WIDTH-1 downto 1) <= Q(WIDTH-2 downto 0);
  end if;
 end if;
end process;

end Behavioral;
```

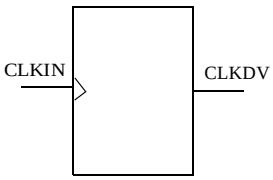
Verilog Inference Code

```
always @ (posedge C or negedge C)
  begin
   if (R)
    Q <= 0;
   else
    begin
     if (CE)
      begin
       Q[0] <= !Q[WIDTH-1];
       Q[WIDTH-1:1] <= Q[WIDTH-2:0];
      end
    end
  end
```

CLK_DIV2,4,6,8,10,12,14,16

Global Clock Divider

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Primitive



The CLK_DIV2,4,6,8,10,12,14,16 Global Clock Dividers divide a user-provided external clock signal gclk<2> by 2, 4, 6, 8, 10, 12, 14, and 16, respectively. Only one clock divider may be used per design. The global clock divider is available on the XC2C128, XC2C256, XC2C384, and XC2C512 CoolRunner-II devices, but not the XC2C32 orXC2C64. The CLKIN input can only be connected to the device gclk<2> pin. The duty cycle of the CLKDV output is 50-50.

The CLKDV output is reset low by power-on reset circuitry.

Usage

This design element is supported for schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for CLK_DIVn should be placed
-- after architecture statement but before begin keyword

component CLK_DIVn
   port (CLKDV: out STD_ULOGIC;
         CLKIN: in STD_ULOGIC);
end component;

-- Component Instantiation for CLK_DIVn should be placed
-- in architecture after the begin keyword

CLK_DIVn_INSTANCE_NAME : CLK_DIVn
      port map  (CLKDV => user_CLKDV,
                 CLKIN => user_CLKIN);
```

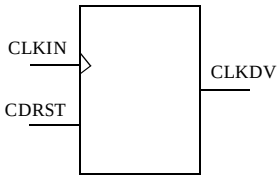
Verilog Instantiation Template

```
CLK_DIVn CLK_DIVn_instance_name(.CLKDV (user_CLKDV),
                                .CLKIN (user_CLKIN));
```

CLK_DIV2,4,6,8,10,12,14,16R

Global Clock Divider with Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Primitive



The CLK_DIV2,4,6,8,10,12,14,16R Global Clock Dividers with Synchronous Reset divide a user-provided external clock signal gclk<2> by 2, 4, 6, 8, 10, 12, 14, and 16, respectively. Only one clock divider may be used per design. The global clock divider is available on the XC2C128, XC2C256, XC2C384, and XC2C512 CoolRunner-II devices, but not the XC2C32 or XC2C64. The CLKIN and CDRST inputs can only be connected to the device gclk<2> and CDRST pins. The duty cycle of the CLKDV output is 50-50.

The CDRST input is an active High synchronous reset. If CDRST is input High when the CLKDV output is High, the CLKDV output remains High to complete the last clock pulse, and then goes Low.

The CLKDV output is reset low by power-on reset circuitry.

Usage

For HDL, these design elements are supported for instantiation but not inference.

VHDL Instantiation Template

```
-- Component Declaration for CLK_DIVnR should be placed
-- after architecture statement but before begin keyword

component CLK_DIVnR
   port (CLDV : out STD_ULOGIC;
         CDRST: in STD_ULOGIC;
         CLKIN: in STD_ULOGIC);
end component;

-- Component Instantiation for CLK_DIVnR should be placed
-- in architecture after the begin keyword

CLK_DIVnR_INSTANCE_NAME : CLK_DIVnR
      port map  (CLKDV => user_CLKDV,
                 CDRST => user_CDRST,
                 CLKIN => user_CLKIN);
```

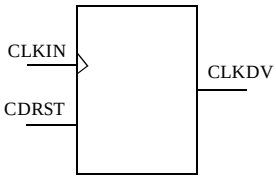
Verilog Instantiation Template

```
CLK_DIVnR CLK_DIVnR_instance_name(.CLKDV (user_CLKDV),
                                  .CDRST (user_CDRST),
                                  .CLKIN (user_CLKIN));
```

CLK_DIV2,4,6,8,10,12,14,16RSD

Global Clock Divider with Synchronous Reset and Start Delay

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Primitive



The CLK_DIV2,4,6,8,10,12,14,16 Global Clock Dividers with Synchronous Reset and Start Delay divide a user-provided external clock signal gclk<2> by 2, 4, 6, 8, 10, 12, 14, and 16, respectively. Only one clock divider may be used per design. The global clock divider is available on the XC2C128, XC2C256, XC2C384, and XC2C512 CoolRunner-II devices, but not the XC2C32 or XC2C64. The CLKIN and CDRST inputs can only be connected to the device gclk<2> and CDRST pins. The duty cycle of the CLKDV output is 50-50.

The CDRST input is an active High synchronous reset. If CDRST is input High when the CLKDV output is High, the CLKDV output remains High to complete the last clock pulse, and then goes Low.

The start delay function delays the start of the CLKDV output by (n + 1) clocks, where n is the divisor for the clock divider.

The CLKDV output is reset low by power-on reset circuitry.

Usage

For HDL, these design elements are supported for instantiation but not inference.

VHDL Instantiation Template

```
-- Component Declaration for CLK_DIVnRSD should be placed
-- after architecture statement but before begin keyword

component CLK_DIVnRSD
      port(CLDV : out STD_ULOGIC;
         CDRST: in STD_ULOGIC;
         CLKIN: in STD_ULOGIC);
end component;

-- Component Instantiation for CLK_DIVnRSD should be placed
-- in architecture after the begin keyword

CLK_DIVnRSD_INSTANCE_NAME : CLK_DIVnRSD
      port map  (CLKDV => user_CLKDV,
                 CDRST => user_CDRST,
                 CLKIN => user_CLKIN);
```

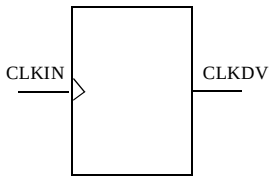
Verilog Instantiation Template

```
CLK_DIVnRSD CLK_DIVnRSD_instance_name(.CLKDV (user_CLKDV),
                                      .CDRST (user_CDRST),
                                      .CLKIN (user_CLKIN));
```

CLK_DIV2,4,6,8,10,12,14,16SD

Global Clock Divider with Start Delay

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Primitive



The CLK_DIV2,4,6,8,10,12,14,16SD Global Clock Dividers with Start Delay divide a user-provided external clock signal gclk<2> by 2, 4, 6, 8, 10, 12, 14, and 16, respectively. Only one clock divider may be used per design. The global clock divider is available on the XC2C128, XC2C256, XC2C384, and XC2C512 CoolRunner-II devices, but not the XC2C32 or XC2C64. The CLKIN input can only be connected to the device gclk<2> pin. The duty cycle of the CLKDV output is 50-50.

The start delay function delays the CLKDV output (n + 1) clocks, where n is the divisor for the clock divider.

The CLKDV output is reset low by power-on reset circuitry.

Usage

This design element is supported for schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for CLK_DIVnSD should be placed
-- after architecture statement but before begin keyword

component CLK_DIVnSD
   port (CLKDV: out STD_ULOGIC;
         CLKIN: in STD_ULOGIC);
end component;


-- Component Instantiation for CLK_DIVnSD should be placed
-- in architecture after the begin keyword

CLK_DIVnSD_INSTANCE_NAME : CLK_DIVnSD
      port map  (CLKDV => user_CLKDV,
                 CLKIN => user_CLKIN);
```

Verilog Instantiation Template

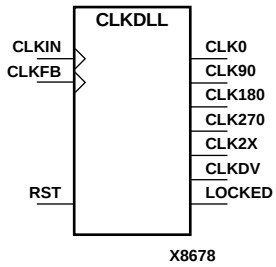
```
CLK_DIVnSD CLK_DIVnSD_instance_name(.CLKDV (user_CLKDV),
                                    .CLKIN (user_CLKIN));
```

CLKDLL

Clock Delay Locked Loop

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive*	Primitive*	N / A	N / A	N / A	N / A

*Use CLKDLLE for Spartan-IIE and Virtex-E



CLKDLL is a clock delay locked loop used to minimize clock skew. CLKDLL synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specific range of each other (see *The Programmable Logic Data Book* for the most current value).

The frequency of the clock signal at the CLKIN input must be in a specific range depending on speed grade (see *The Programmable Logic Data Book* for the most current values). The CLKIN pin must be driven by an IBUFG or a BUFG.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG connected to the CLKFB input of the CLKDLL must be sourced from either the CLK0 or CLK2X outputs of the same CLKDLL. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Either the CLK0 or CLK2X output can be used but not both. The CLK0 or CLK2X must be connected to the input of OBUF, an output buffer.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted outputs (CLK90, CLK180, and CLK270) is the same as that of the CLK0 output. The duty cycle of the CLK2X and CLKDV outputs is always 50-50. The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

The master reset input (RST) resets CLKDLL to its initial (power-on) state. The signal at the RST input is asynchronous and must be held High for just 2ns.

CLKDLL Outputs

Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLKIN frequency, shifted 180^{o} with regards to CLK0
CLK270	Clock at 1x CLKIN frequency, shifted 270^{o} with regards to CLK0
CLK2X	Clock at 2x CLKIN frequency, in phase with CLK0
CLK90	Clock at 1x CLKIN frequency, shifted 90^{o} with regards to CLK0
CLKDV	Clock at (1 / n)x CLKIN frequency, n=CLKDV_DIVIDE value. CLKDV is in phase with CLK0.
LOCKED	CLKDLL locked

Note See the "PERIOD Specifications on CLKDLLs and DCM" section of the "Xilinx Constraints P" chapter in the *Constraints Guide* for additional information on using the TNM, TNM_NET, and PERIOD attributes with CLKDLL components.

Usage

This component is generally instantiated in the code as it can not be easily inferred in synthesis tools. Some synthesis tools may allow inference via an attribute. Refer to your synthesis tool's documentation if that is desirable. Generally, global buffers (IBUFG, BUFG) are instantiated with the CLKDLL component to construct the proper clocking circuit. Refer to the XAPP 132 application note, "Using the Virtex Delay-Locked Loop" and the *Xilinx Databook* for more information on using the CLKDLL component.

VHDL Instantiation Template

```
-- Component Declaration for CLKDLL should be placed
-- after architecture statement but before begin keyword

component CLKDLL
  -- synthesis translate_off
  generic (CLKDV_DIVIDE :real := 2.0; -- (1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 8.0, 16.0)
           DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
           FACTORY_JF : bit_vector :=X"C080";
           STARTUP_WAIT :boolean := FALSE); -- (TRUE, FALSE)
  -- synthesis translate_on
  port (CLK0  : out STD_ULOGIC;
        CLK180  : out STD_ULOGIC;
        CLK270  : out STD_ULOGIC;
        CLK2X   : out STD_ULOGIC;
        CLK90   : out STD_ULOGIC;
        CLKDV   : out STD_ULOGIC;
        LOCKED  : out STD_ULOGIC;
        CLKFB : in STD_ULOGIC;
        CLKIN : in STD_ULOGIC;
        RST   : in STD_ULOGIC);
end component;

-- Component Attribute specification for CLKDLL
-- should be placed after architecture declaration but
```

```
-- before the begin keyword

attribute CLKDV_DIVIDE : real;
attribute DUTY_CYCLE_CORRECTION : boolean;
attribute FACTORY_JF : bit_vector;
attribute STARTUP_WAIT : boolean;

attribute CLKDV_DIVIDE of CLKDLL_instance_name: label is 2.0;
-- 1.5,2,2.5,3,4, 5, 8, 16 are valid for CLKDV_DIVIDE
attribute DUTY_CYCLE_CORRECTION of CLKDLL_instance_name: label is "TRUE";
-- TRUE, FALSE are valid for DUTY_CYCLE_CORRECTION
attribute FACTORY_JF of CLKDLL_instance_name label is X"C080";
attribute STARTUP_WAIT of CLKDLL_instance_name: label is "FALSE"; -- (TRUE,FALSE)


-- Component Instantiation for CLKDLL should be placed
-- in architecture after the begin keyword

CLKDLL_INSTANCE_NAME : CLKDLL
  -- synthesis translate_off
  generic map(CLKDV_DIVIDE => real_value, -- (1.5,2,2.5,3,4,5,8,16)
              DUTY_CYCLE_CORRECTION => boolean_value, -- (TRUE, FALSE)
              FACTORY_JF => "hex_value";
              STARTUP_WAIT  => boolean_value); -- (TRUE, FALSE)
              -- synthesis translate_on
  port map  (CLK0   => user_CLK0,
             CLK180 => user_CLK180,
             CLK270 => user_CLK270,
             CLK2X  => user_CLK2X,
             CLK90  => user_CLK90,
             CLKDV=> user_CLKDV,
             LOCKED => user_LOCKED,
             CLKFB  => user_CLKFB,
             CLKIN0 => user_CLKIN,
             RST    => user_RST);
```

Verilog Instantiation Template

```
CLKDLL CLKDLL_instance_name (.CLK0 (user_CLK0),
                             .CLK180 (user_CLK180),
                             .CLK270 (user_CLK270),
                             .CLK2X (user_CLK2X),
                             .CLK90 (user_CLK90),
                             .CLKDV (user_CLKDV),
                             .LOCKED (user_LOCKED),
                             .CLKFB (user_CLKFB),
                             .CLKIN (user_CLKIN),
                             .RST (user_RST));

defparam CLKDLL_instance_name.CLKDV_DIVIDE = integer_value; //(1.5,2,2.5,3,4,5,8,16)
defparam CLKDLL_instance_name.DUTY_CYCLE_CORRECTION = boolean_value;// (TRUE, FALSE)
defparam CLKDLL_instance_name.FACTORY_JF = "hex_value";
defparam CLKDLL_instance_name.STARTUP_WAIT = boolean_value; // (TRUE, FALSE)
```

Note Additional syntax may be necessary in order to pass the CLKDLL attributes via the synthesis tool and the above defparam statements may need to be isolated from the synthesis tool with translate_off / translate_on directives. Please refer to your synthesis tool documentation for more information on Verilog attribute passing to ensure you properly pass these attributes to the synthesis tool or else you may pass these attributes to the UCF file.

Commonly Used Constraints

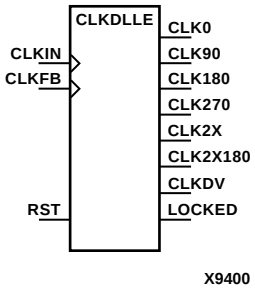
STARTUP_WAIT, DUTY_CYCLE_CORRECTION, CLKDV_DIVIDE, FACTORY_JF, LOC

CLKDLLE

Virtex-E Clock Delay Locked Loop

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive*	Primitive*	N / A	N / A	N / A	N / A

* Supported for Spartan-IIE and Virtex-E devices only



CLKDLLE is a clock delay locked loop used to minimize clock skew for Virtex-E devices. CLKDLLE synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specific range of each other (see *The Programmable Logic Data Book* for the most current value).

The frequency of the clock signal at the CLKIN input must be in a specific range depending on speed grade (see *The Programmable Logic Data Book* for the most current values). The CLKIN pin must be driven by an IBUFG or a BUFG.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG input can only be connected to the CLK0 or CLK2X output of CLKDLLE. The BUFG connected to the CLKFB input of the CLKDLLE must be sourced from either the CLK0 or CLK2X outputs of the same CLKDLLE. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Either the CLK0 or CLK2X output can be used but not both. The CLK0 or CLK2X must be connected to the input of OBUF, an output buffer.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted outputs (CLK90, CLK180, and CLK270) is the same as that of the CLK0 output. The duty cycle of the CLK2X, CLK2X180, and CLKDV outputs is always 50-50. The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

The master reset input (RST) resets CLKDLLE to its initial (power-on) state. The signal at the RST input is asynchronous and must be held High for just 2ns.

CLKDLLE Outputs

Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLK0 frequency, shifted 180^{o} with regards to CLK0
CLK270	Clock at 1x CLK0 frequency, shifted 270^{o} with regards to CLK0
CLK2X	Clock at 2x CLK0 frequency, in phase with CLK0
CLK2X180	Clock at 1x CLK2X frequency shifted 180^{o} with regards to CLK2X
CLK90	Clock at 1x CLK0 frequency, shifted 90^{o} with regards to CLK0
CLKDV	Clock at (1 / n) x CLK0 frequency, where n=CLKDV_DIVIDE value. CLKDV is in phase with CLK0.
LOCKED	CLKDLLE locked. CLKIN and CLKFB synchronized.

Usage

This component is generally instantiated in the code as it cannot be easily inferred in synthesis tools. Some synthesis tools may allow inference via an attribute. Refer to your synthesis tool documentation if that is desirable. Generally, global buffers (IBUFG, BUFG) are instantiated with the CLKDLLE component to construct the proper clocking circuit. Refer to the XAPP 132 application note, "Using the Virtex Delay-Locked Loop" and the *Xilinx Databook* for more information on using the CLKDLLE component.

VHDL Instantiation Template

```
-- Component Declaration for CLKDLLE should be placed
-- after architecture statement but before begin keyword

component CLKDLLE
  -- synthesis translate_off
  generic (CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0,
           5.5, 6.0, 6.5, 7.5, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0)
           DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
           FACTORY_JF : bit_vector : X"C080";
           STARTUP_WAIT :boolean := FALSE); -- (TRUE, FALSE)
  -- synthesis translate_on
  port (CLK0      : out STD_ULOGIC;
        CLK180    : out STD_ULOGIC;
        CLK270    : out STD_ULOGIC;
        CLK2X     : out STD_ULOGIC;
        CLK2X180  : out STD_ULOGIC;
        CLK90     : out STD_ULOGIC;
        CLKDV     : out STD_ULOGIC;
        LOCKED    : out STD_ULOGIC;
        CLKFB     : in STD_ULOGIC;
        CLKIN     : in STD_ULOGIC;
        RST       : in STD_ULOGIC);
end component;

-- Component Attribute specification for CLKDLLE
-- should be placed after architecture declaration but
```

```
-- before the begin keyword

attribute CLKDV_DIVIDE : real;
attribute DUTY_CYCLE_CORRECTION : boolean;
attribute FACTORY_JF : bit_vector;
attribute STARTUP_WAIT : boolean;

attribute CLKDV_DIVIDE of CLKDLLE_instance_name: label is 2.0;
-- (1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.5, 8.0, 9.0, 10.0, 11.0,
-- 12.0, 13.0, 14.0, 15.0, 16.0) are valid for CLKDV_DIVIDE
attribute DUTY_CYCLE_CORRECTION of CLKDLLE_instance_name: label is TRUE;
-- (TRUE, FALSE) are valid for DUTY_CYCLE_CORRECTION
attribute FACTORY_JF of CLKDLLE_instance_name : label is X"C080";
attribute STARTUP_WAIT of CLKDLLE_instance_name: label is FALSE; -- (TRUE,FALSE)


-- Component Instantiation for CLKDLLE should be placed
-- in architecture after the begin keyword

CLKDLLE_INSTANCE_NAME : CLKDLLE
  -- synthesis translate_off
  generic map (CLKDV_DIVIDE=> real_value, -- (1.5,2,2.5,3,4,5,8,16)
               DUTY_CYCLE_CORRECTION => boolean_value, -- (TRUE, FALSE)
               FACTORY_JF => "hex_value",
               STARTUP_WAIT=> boolean_value); -- (TRUE, FALSE)
  -- synthesis translate_on
  port map  (CLK0   => user_CLK0,
             CLK180 => user_CLK180,
             CLK270 => user_CLK270,
             CLK2X  => user_CLK2X,
             CLK2X180=> user_CLK2X,
             CLK90  => user_CLK90,
             CLKDV  => user_CLKDV,
             LOCKED => user_LOCKED,
             CLKFB  => user_CLKFB,
             CLKIN0 => user_CLKIN,
             RST    => user_RST);
```

Verilog Instantiation Template

```
CLKDLLE CLKDLLE_instance_name (.CLK0 (user_CLK0),
                               .CLK180 (user_CLK180),
                               .CLK270 (user_CLK270),
                               .CLK2X (user_CLK2X),
                               .CLK2X180 (user_CLK2X180),
                               .CLK90 (user_CLK90),
                               .CLKDV (user_CLKDV),
                               .LOCKED (user_LOCKED),
                               .CLKFB (user_CLKFB),
                               .CLKIN (user_CLKIN),
                               .RST (user_RST));

defparam CLKDLLE_instance_name.CLKDV_DIVIDE = integer_value;
// 1.5,2,2.5,3,4,5,8,16 are valid for CLKDV_DIVIDE
```

```
defparam CLKDLLE_instance_name.DUTY_CYCLE_CORRECTION = boolean_value;// (TRUE,FALSE)
defparam CLKDLLE_instance_name.FACTORY_JF = "hex_value";
defparam CLKDLLE_instance_name.STARTUP_WAIT = boolean_value; // (TRUE, FALSE)
```

Note Additional syntax may be necessary in order to pass the CLKDLLE attributes via the synthesis tool and the above defparam statements may need to be isolated from the synthesis tool with translate_off / translate_on directives. Please refer to your synthesis tool documentation for more information on Verilog attribute passing to ensure you properly pass these attributes to the synthesis tool or else you may pass these attributes to the UCF file.

Commonly Used Constraints

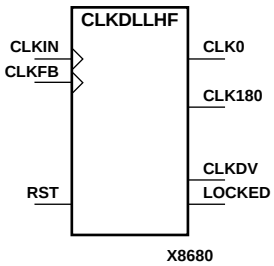
STARTUP_WAIT, DUTY_CYCLE_CORRECTION, CLKDV_DIVIDE, FACTORY_JF, LOC

CLKDLLHF

High Frequency Clock Delay Locked Loop

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive*	N/A	N/A	N/A	N/A

*Use CLKDLLHF for the Virtex-E DLL in HF mode. In LF mode, both the separate CLKDLLE and CLKDLL primitive can be used.



CLKDLLHF is a high frequency clock delay locked loop used to minimize clock skew. CLKDLLHF synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specific range of each other (see *The Programmable Logic Data Book* for the most current value).

The frequency of the clock signal at the CLKIN input must be in a specific range depending on speed grade (see *The Programmable Logic Data Book* for the most current values). The CLKIN pin must be driven by an IBUFG or a BUFG.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG input can only be connected to the CLK0 output of CLKDLLHF. The BUFG connected to the CLKFB input of the CLKDLLHF must be sourced from the CLK0 output of the same CLKDLLHF. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Only the CLK0 output can be used. CLK0 must be connected to the input of OBUF, an output buffer.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted output (CLK180) is the same as that of the CLK0 output. The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

The master reset input (RST) resets CLKDLLHF to its initial (power-on) state. The signal at the RST input is asynchronous and must be held High for just 2ns.

CLKDLLHF Outputs

Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLKIN frequency, shifted 180^{o} with regards to CLK0
CLKDV	Clock at (1/n)x CLKIN frequency, n=CLKDV_DIVIDE value. CLKDV is in phase with CLK0.
LOCKED	CLKDLLHF locked

Note See the "PERIOD Specifications on CLKDLLs and DCM" section of the "Xilinx Constraints P" chapter in the *Constraints Guide* for additional information on using the TNM, TNM_NET, and PERIOD attributes with CLKDLLHF components.

Usage

This component is generally instantiated in the code as it cannot be easily inferred in synthesis tools. Some synthesis tools may allow inference via an attribute. Refer to your synthesis tool documentation if that is desirable. Generally, global buffers (IBUFG, BUFG) are instantiated with the CLKDLLHF component to construct the proper clocking circuit. Refer to the XAPP 132 application note, "Using the Virtex Delay-Locked Loop" and the *Xilinx Databook* for more information on using the CLKDLLHF component.

VHDL Instantiation Template

```
-- Component Declaration for CLKDLLHF should be placed
-- after architecture statement but before begin keyword

component CLKDLLHF
  -- synthesis translate_off
  generic (CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 8.0, 16.0)
          DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
          FACTORY_JF : bit_vector : X"C080";
          STARTUP_WAIT :boolean := FALSE); -- (TRUE, FALSE)
  -- synthesis translate_on
  port (CLK0    : out STD_ULOGIC;
        CLK180  : out STD_ULOGIC;
        CLKDV   : out STD_ULOGIC;
        LOCKED  : out STD_ULOGIC;
        CLKFB   : in STD_ULOGIC;
        CLKIN   : in STD_ULOGIC;
        RST     : in STD_ULOGIC);
end component;

-- Component Attribute specification for CLKDLLHF
-- should be placed after architecture declaration but
-- before the begin keyword

attribute CLKDV_DIVIDE : real;
attribute DUTY_CYCLE_CORRECTION : boolean;
attribute FACTORY_JF : bit_vector;
attribute STARTUP_WAIT : boolean;


attribute CLKDV_DIVIDE of CLKDLLHF_instance_name: label is 2.0;
-- (1.5,2,2.5,3,4, 5, 8, 16) are valid for CLKDV_DIVIDE
attribute DUTY_CYCLE_CORRECTION of CLKDLLHF_instance_name: label is TRUE;
-- (TRUE, FALSE) are valid for DUTY_CYCLE_CORRECTION
attribute FACTORY_JF of CLKDLLHF_instance_name label is X"C080";
attribute STARTUP_WAIT of CLKDLLHF_instance_name: label is FALSE; -- (TRUE,FALSE)

-- Component Instantiation for CLKDLLHF should be placed
-- in architecture after the begin keyword

CLKDLLHF_INSTANCE_NAME : CLKDLLHF
  -- synthesis translate_off
  generic map(CLKDV_DIVIDE => real_value, -- (1.5,2,2.5,3,4,5,8,16)
```

```
          DUTY_CYCLE_CORRECTION => boolean_value, -- (TRUE, FALSE)
          FACTORY_JF => "hex_value",
          STARTUP_WAIT => boolean_value); -- (TRUE, FALSE)
  -- synthesis translate_on
  port map  (CLK0     => user_CLK0,
             CLK180   => user_CLK180,
             CLKDV    => user_CLKDV,
             LOCKED   => user_LOCKED,
             CLKFB    => user_CLKFB,
             CLKIN    => user_CLKIN,
             RST      => user_RST);
```

Verilog Instantiation Template

```
CLKDLLHF CLKDLLHF_instance_name (.CLK0 (user_CLK0),
                                 .CLK180 (user_CLK180),
                                 .CLKDV (user_CLKDV),
                                 .LOCKED (user_LOCKED),
                                 .CLKFB (user_CLKFB),
                                 .CLKIN (user_CLKIN),
                                 .RST (user_RST));

defparam CLKDLLHF_instance_name.CLKDV_DIVIDE = integer_value;
// 1.5,2,2.5,3,4,5,8,16 are valid for CLKDV_DIVIDE
defparam CLKDLLHF_instance_name.DUTY_CYCLE_CORRECTION = boolean_value;//
(TRUE,FALSE)
defparam CLKDLLHF_instance_name.FACTORY_JF = "hex_value";
defparam CLKDLLHF_instance_name.STARTUP_WAIT = boolean_value; // (TRUE, FALSE)
```

Note Additional syntax may be necessary in order to pass the CLKDLLHF attributes via the synthesis tool and the above defparam statements may need to be isolated from the synthesis tool with translate_off / translate_on directives. Please refer to your synthesis tool documentation for more information on Verilog attribute passing to ensure that you properly pass these attributes to the synthesis tool or else you may pass these attributes to the UCF file.

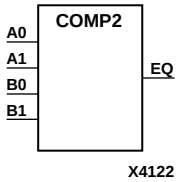
Commonly Used Constraints

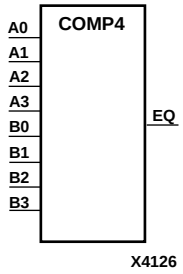
STARTUP_WAIT, DUTY_CYCLE_CORRECTION, CLKDV_DIVIDE, FACTORY_JF, LOC

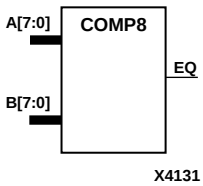
COMP2, 4, 8, 16

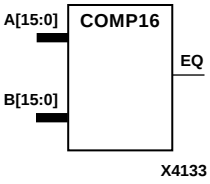
2-, 4-, 8-, 16-Bit Identity Comparators

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



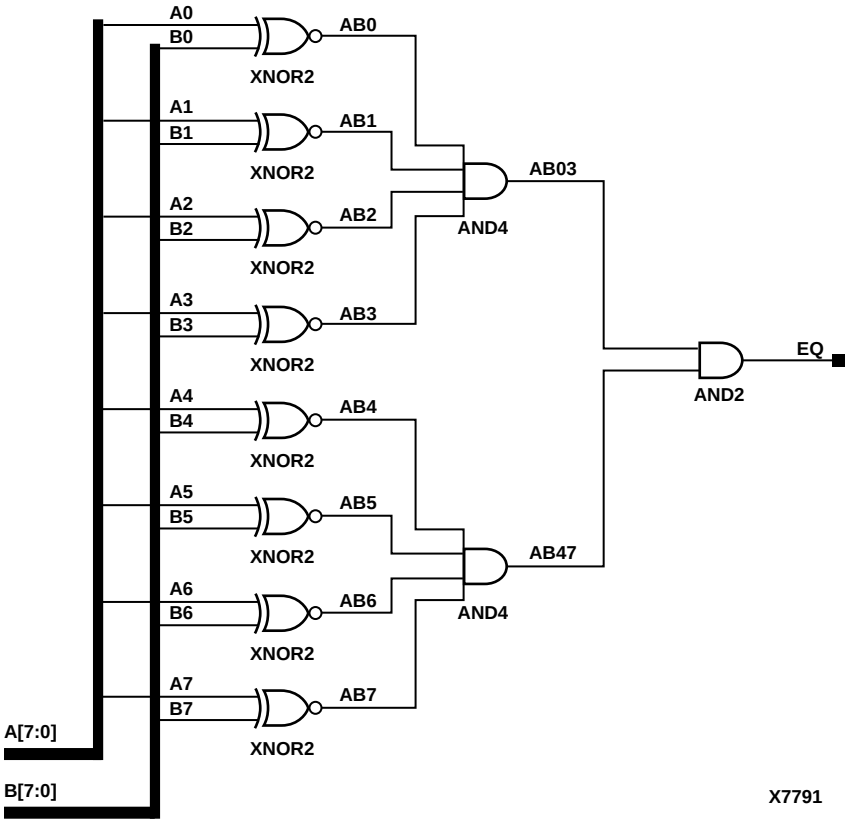






COMP2, COMP4, COMP8, and COMP16 are, respectively, 2-, 4-, 8-, and 16-bit identity comparators. The equal output (EQ) of the COMP2 2-bit, identity comparator is High when the two words A1 – A0 and B1 – B0 are equal. EQ is high for COMP4 when A3 – A0 and B3 – B0 are equal; for COMP8, when A7 – A0 and B7 – B0 are equal; and for COMP16, when A15 – A0 and B15 – B0 are equal.

Equality is determined by a bit comparison of the two words. When any two of the corresponding bits from each word are not the same, the EQ output is Low.



COMP8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture behavioral of comp2 is

begin
 process (A, B)
 begin
  If (A=B) then
   EQ <= ’1’;
  else
   EQ <= ’0’;
  end if;
  end process;
end behavioral;
```

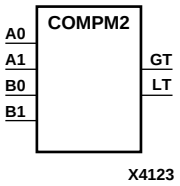
Verilog Inference Code

```
always @ (A or B)
  begin
   if (A == B)
   EQ <= 1;
  else
   EQ <= 0;
  end
```

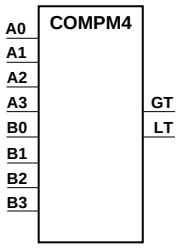
COMPM2, 4, 8, 16

2-, 4-, 8-, 16-Bit Magnitude Comparators

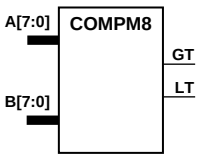
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



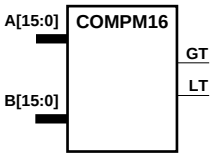
X4123



X4127



X4132



X4134

COMPM2, COMPM4, COMPM8, and COMPM16 are, respectively, 2-, 4-, 8-, and 16-bit magnitude comparators that compare two positive binary-weighted words.

COMPM2 compares A1 – A0 and B1 – B0, where A1 and B1 are the most significant bits. COMPM4 compares A3 – A0 and B3 – B0, where A3 and B3 are the most significant bits. COMPM8 compares A7 – A0 and B7 – B0, where A7 and B7 are the most significant bits. COMPM16 compares A15 – A0 and B15 – B0, where A15 and B15 are the most significant bits.

The greater-than output (GT) is High when A>B, and the less-than output (LT) is High when A<B. When the two words are equal, both GT and LT are Low. Equality can be measured with this macro by comparing both outputs with a NOR gate.

COMPM2 Truth Table

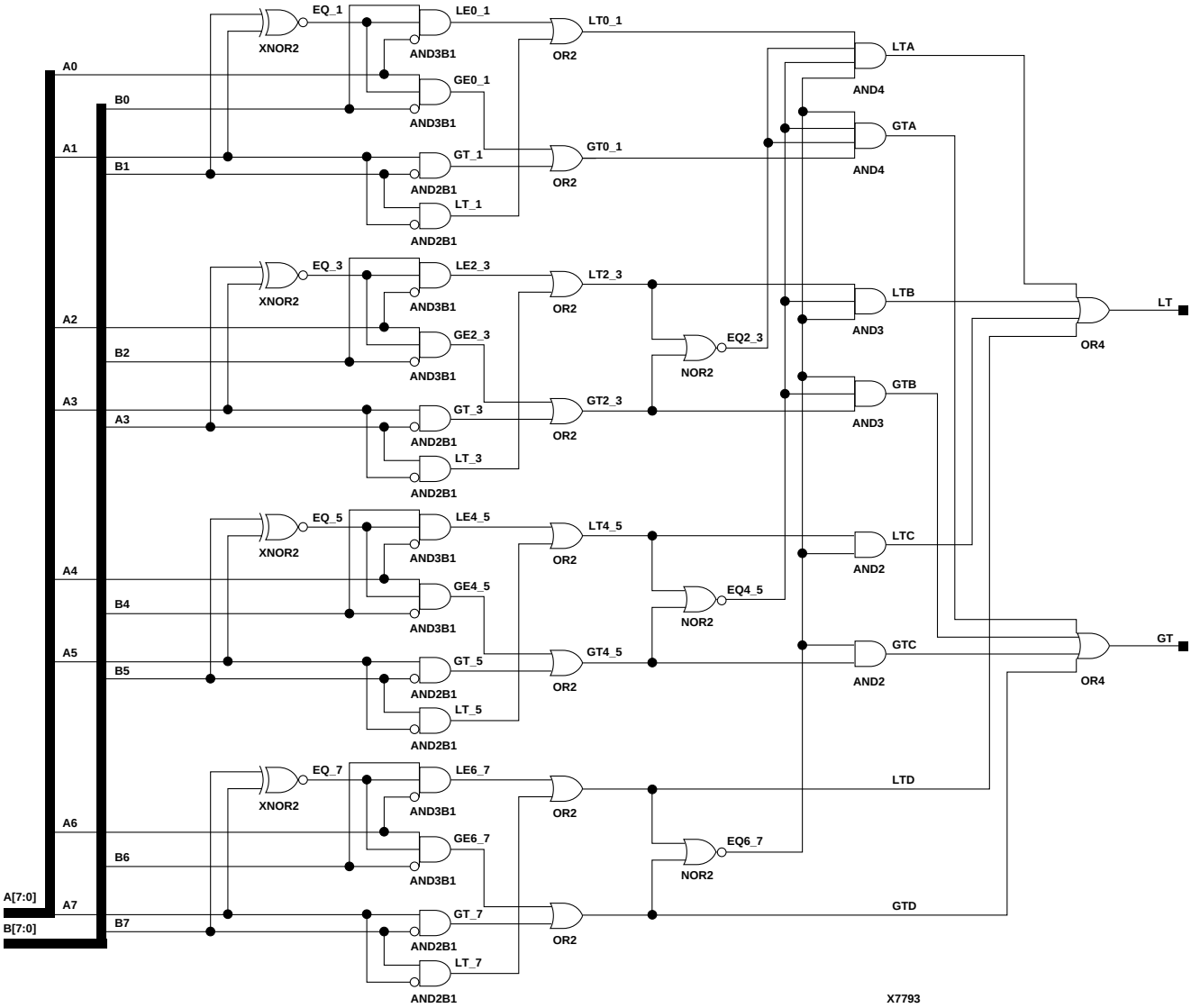
Inputs				Outputs	
A1	B1	A0	B0	GT	LT
0	0	0	0	0	0
0	0	1	0	1	0
0	0	0	1	0	1
0	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	0
1	1	0	1	0	1
1	1	1	1	0	0
1	0	X	X	1	0
0	1	X	X	0	1

COMPM4 Truth Table

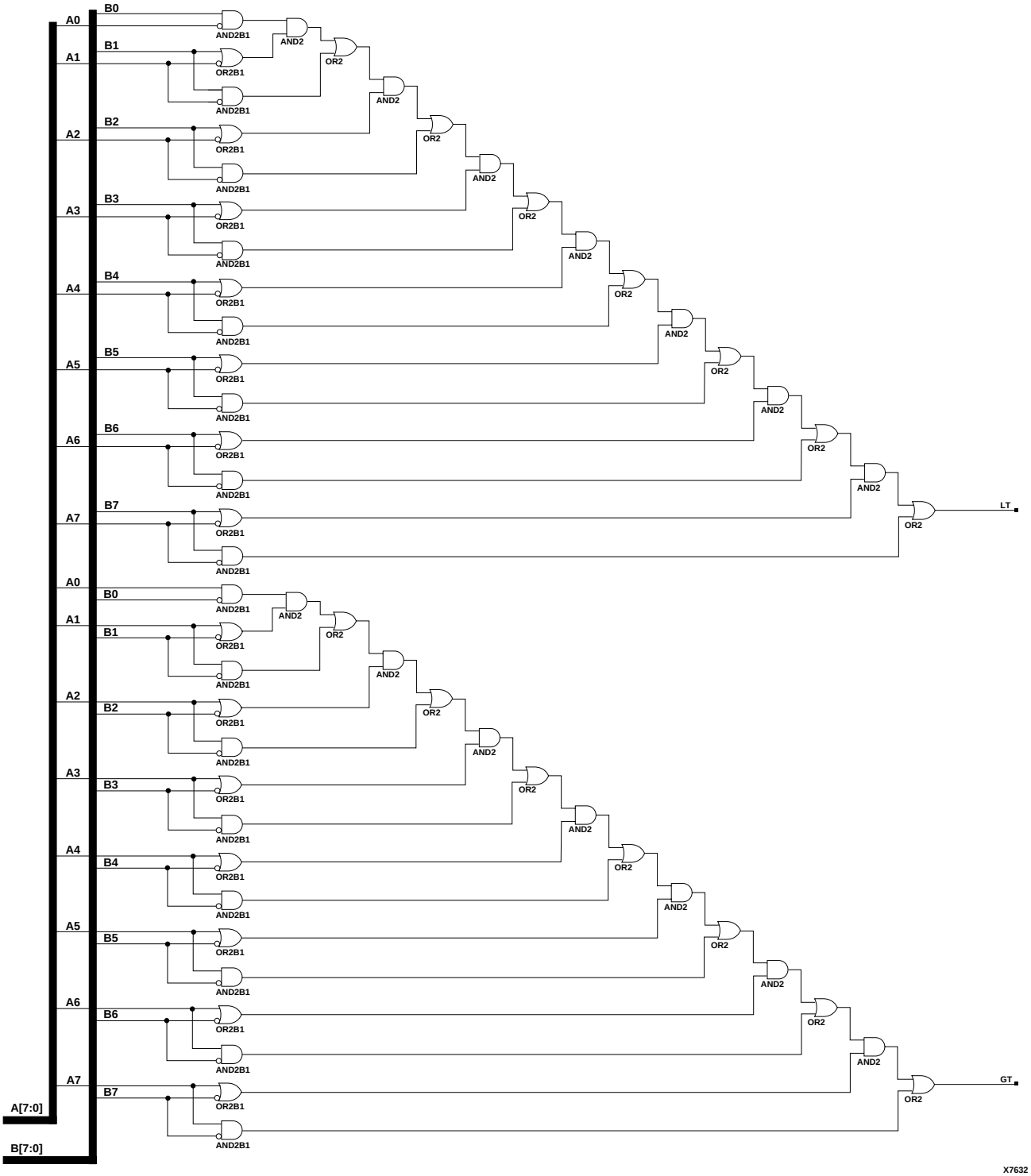
Inputs				Outputs	
A3, B3	**A2, B2**	**A1, B1**	**A0, B0**	**GT**	**LT**
A3>B3	X	X	X	1	0
A3<B3	X	X	X	0	1
A3=B3	A2>B2	X	X	1	0
A3=B3	A2<B2	X	X	0	1
A3=B3	A2=B2	A1>B1	X	1	0
A3=B3	A2=B2	A1<B1	X	0	1
A3=B3	A2=A2	A1=B1	A0>B0	1	0
A3=B3	A2=B2	A1=B1	A0<B0	0	1
A3=B3	A2=B2	A1=B1	A0=B0	0	0

COMPM8 Truth Table (also representative of COMPM16)

Inputs								Outputs	
A7, B7	**A6, B6**	**A5, B5**	**A4, B4**	**A3, B3**	**A2, B2**	**A1, B1**	**A0, B0**	**GT**	**LT**
A7>B7	X	X	X	X	X	X	X	1	0
A7<B7	X	X	X	X	X	X	X	0	1
A7=B7	A6>B6	X	X	X	X	X	X	1	0
A7=B7	A6<B6	X	X	X	X	X	X	0	1
A7=B7	A6=B6	A5>B5	X	X	X	X	X	1	0
A7=B7	A6=B6	A5<B5	X	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4>B4	X	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4<B4	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3>B3	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3<B3	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2>B2	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2<B2	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1>B1	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1<B1	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0>B0	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0<B0	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0=B0	0	0



COMPM8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



COMPM8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are supported for inference rather than instantiation.

VHDL Inference Code

```
architecture Behavioral of compm2 is

begin
 process (A,B)
 begin
  if (A>B) then
   GT <= '1';
   LT <= '0';
  elsif (A<B) then
   GT <= '0';
   LT <= '1';
  else
   GT <= '0';
   LT <= '0';
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

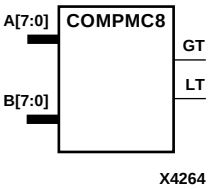
always @ (A or B)

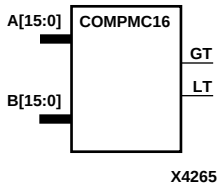
```
  begin
   if (A > B)
  begin
   GT <= 1;
   LT <= 0;
  end
  else if (A < B)
  begin
   GT <= 0;
   LT <= 1;
  end
  else
  begin
   GT <= 0;
   LT <= 0;
  end
  end
```

COMPMC8, 16

8-, 16-Bit Magnitude Comparators

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A





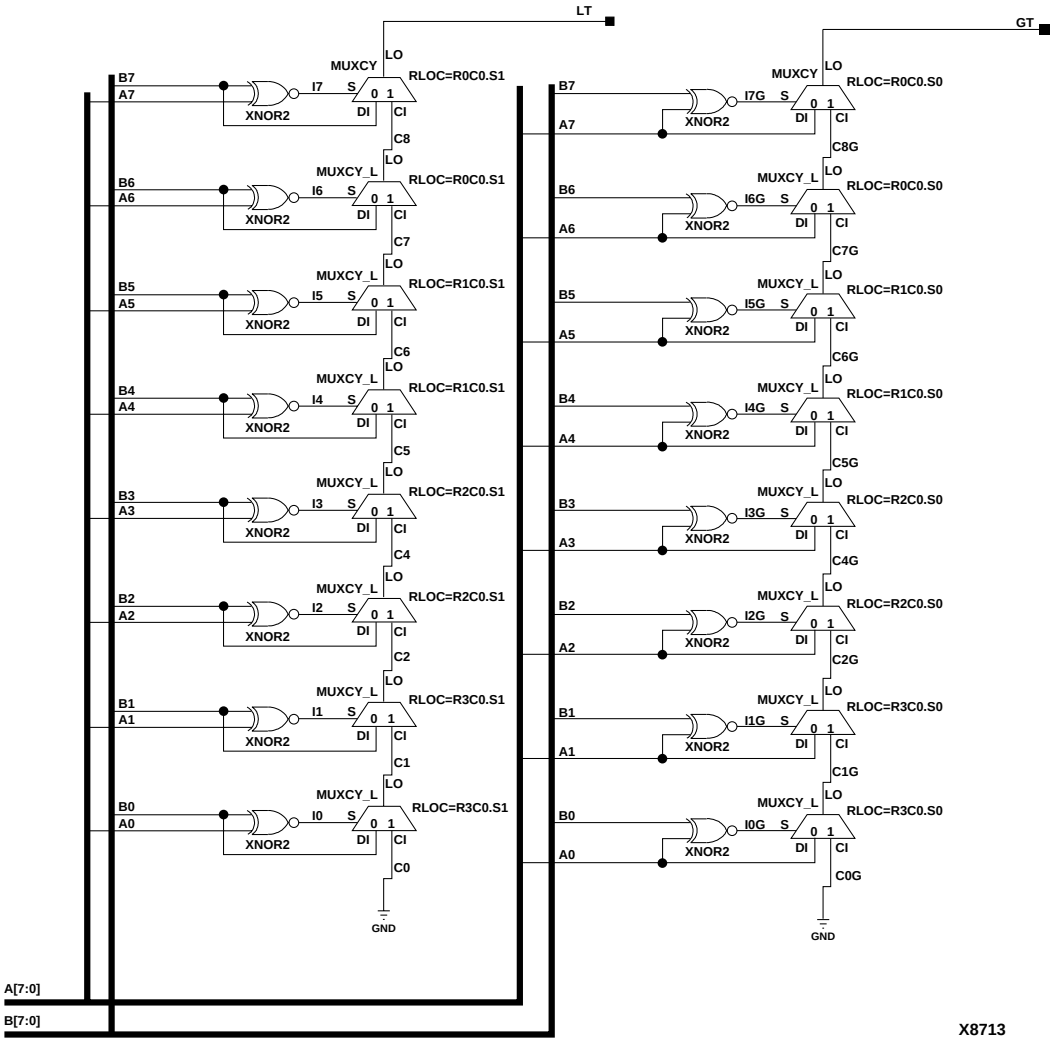
COMPMC8 is an 8-bit, magnitude comparator that compares two positive binary-weighted words A7 – A0 and B7 – B0, where A7 and B7 are the most significant bits. COMPMC16 is a 16-bit, magnitude comparator that compares two positive binary-weighted words A15 – A0 and B15 – B0, where A15 and B15 are the most significant bits.

These comparators are implemented using carry logic with relative location constraints to ensure efficient logic placement.

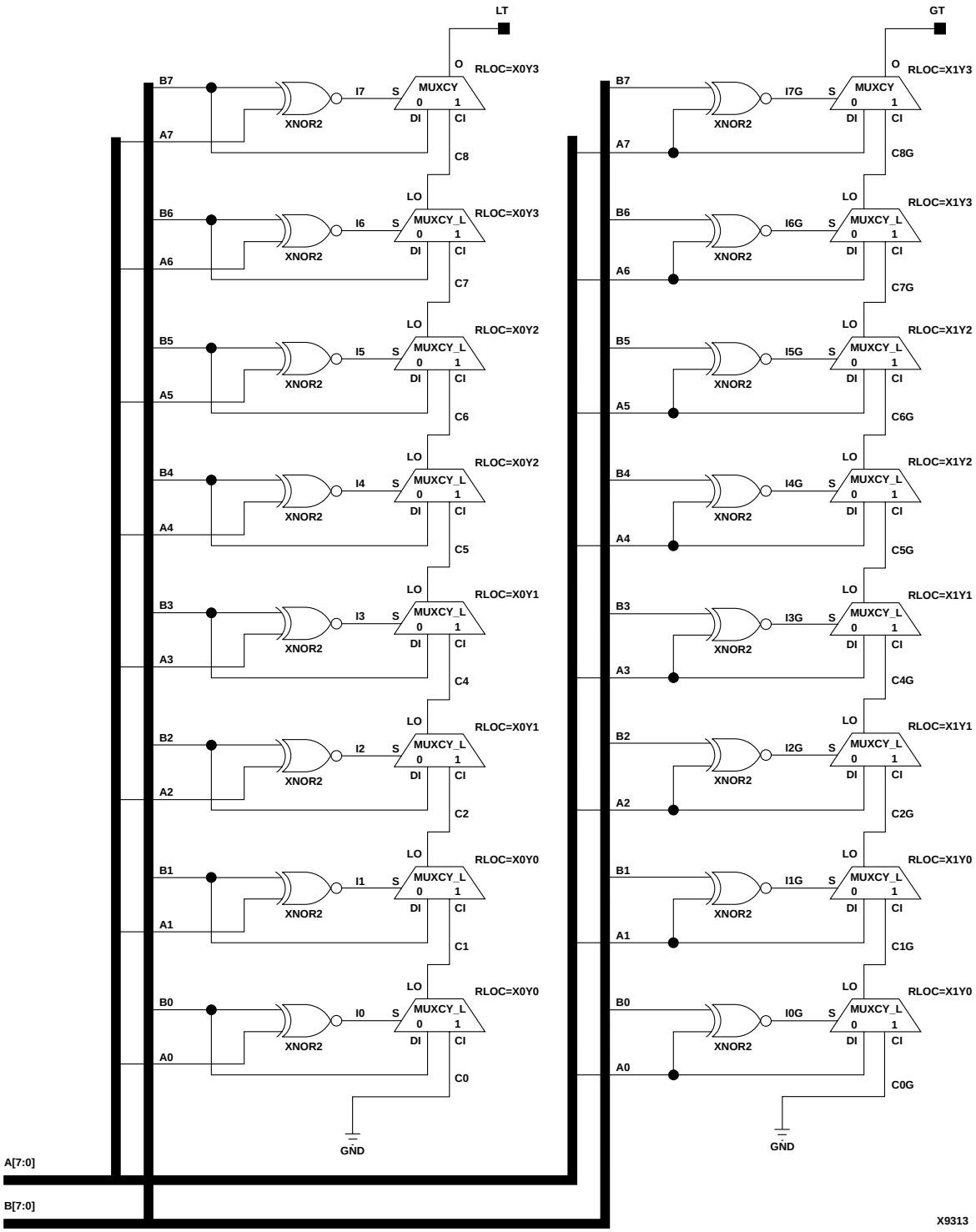
The greater-than output (GT) is High when A>B, and the less-than output (LT) is High when A<B. When the two words are equal, both GT and LT are Low. Equality can be flagged with this macro by connecting both outputs to a NOR gate.

COMPMC8 Truth Table (also representative of COMPMC16)

Inputs								Outputs	
A7, B7	A6, B6	A5, B5	A4, B4	A3, B3	A2, B2	A1, B1	A0, B0	GT	LT
A7>B7	X	X	X	X	X	X	X	1	0
A7<B7	X	X	X	X	X	X	X	0	1
A7=B7	A6>B6	X	X	X	X	X	X	1	0
A7=B7	A6<B6	X	X	X	X	X	X	0	1
A7=B7	A6=B6	A5>B5	X	X	X	X	X	1	0
A7=B7	A6=B6	A5<B5	X	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4>B4	X	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4<B4	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3>B3	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3<B3	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2>B2	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2<B2	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1>B1	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1<B1	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0>B0	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0<B0	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0=B0	0	0



COMPMC8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



COMPMC8 Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are supported for inference rather than instantiation.

VHDL Inference Code

```
architecture Behavioral of compmc8 is

begin
 process (A,B)
 begin
  if (A>B) then
   GT <= '1';
   LT <= '0';
  elsif (A<B) then
   GT <= '0';
   LT <= '1';
  else
   GT <= '0';
   LT <= '0';
  end if;
 end process;
end Behavioral;
```

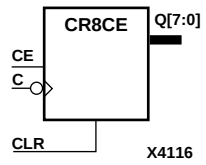
Verilog Inference Code

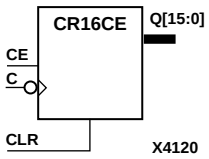
```
always @(A or B)
  begin
   if (A > B)
  begin
   GT <= 1;
   LT <= 0;
  end
  else if (A < B)
  begin
   GT <= 0;
   LT <= 1;
  end
  else
  begin
   GT <= 0;
   LT <= 0;
  end
  end
```

CR8CE, CR16CE

8-, 16-Bit Negative-Edge Binary Ripple Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro





CR8CE and CR16CE are 8-bit and 16-bit, cascadable, clearable, binary, ripple counters. The asynchronous clear (CLR), when High, overrides all other inputs and causes the Q outputs to go to logic level zero. The counter increments when the clock enable input (CE) is High during the High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low.

Larger counters can be created by connecting the last Q output (Q7 for CR8CE, Q15 for CR16CE) of the first stage to the clock input of the next stage. CLR and CE inputs are connected in parallel. The clock period is not affected by the overall length of a ripple counter. The overall clock-to-output propagation is $n(t_{C-Q})$, where n is the number of stages and the time t_{C-Q} is the C-to-Qz propagation delay of each stage.

The counter is asynchronously cleared, output Low, when power is applied.

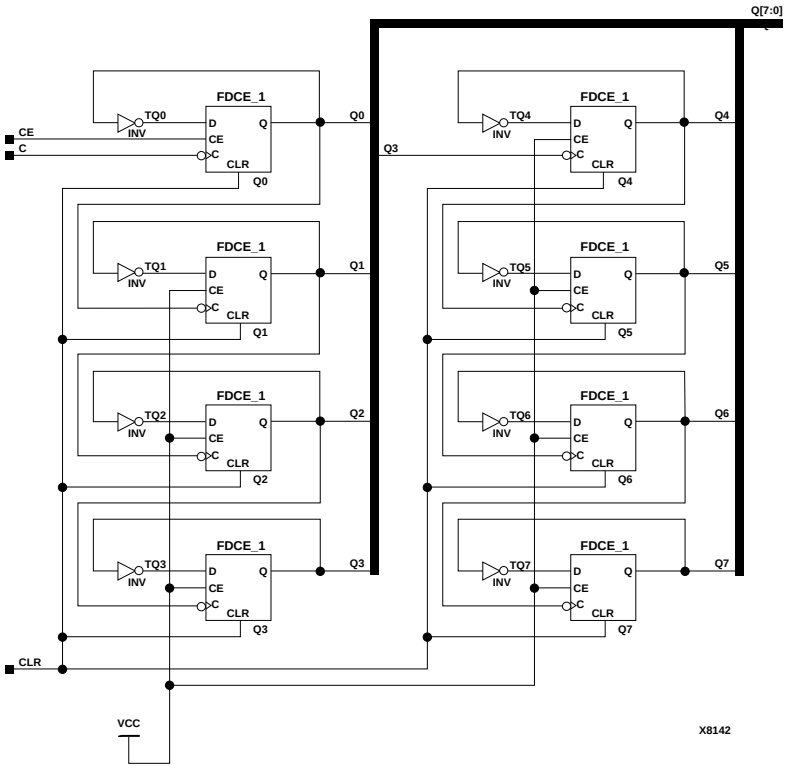
For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

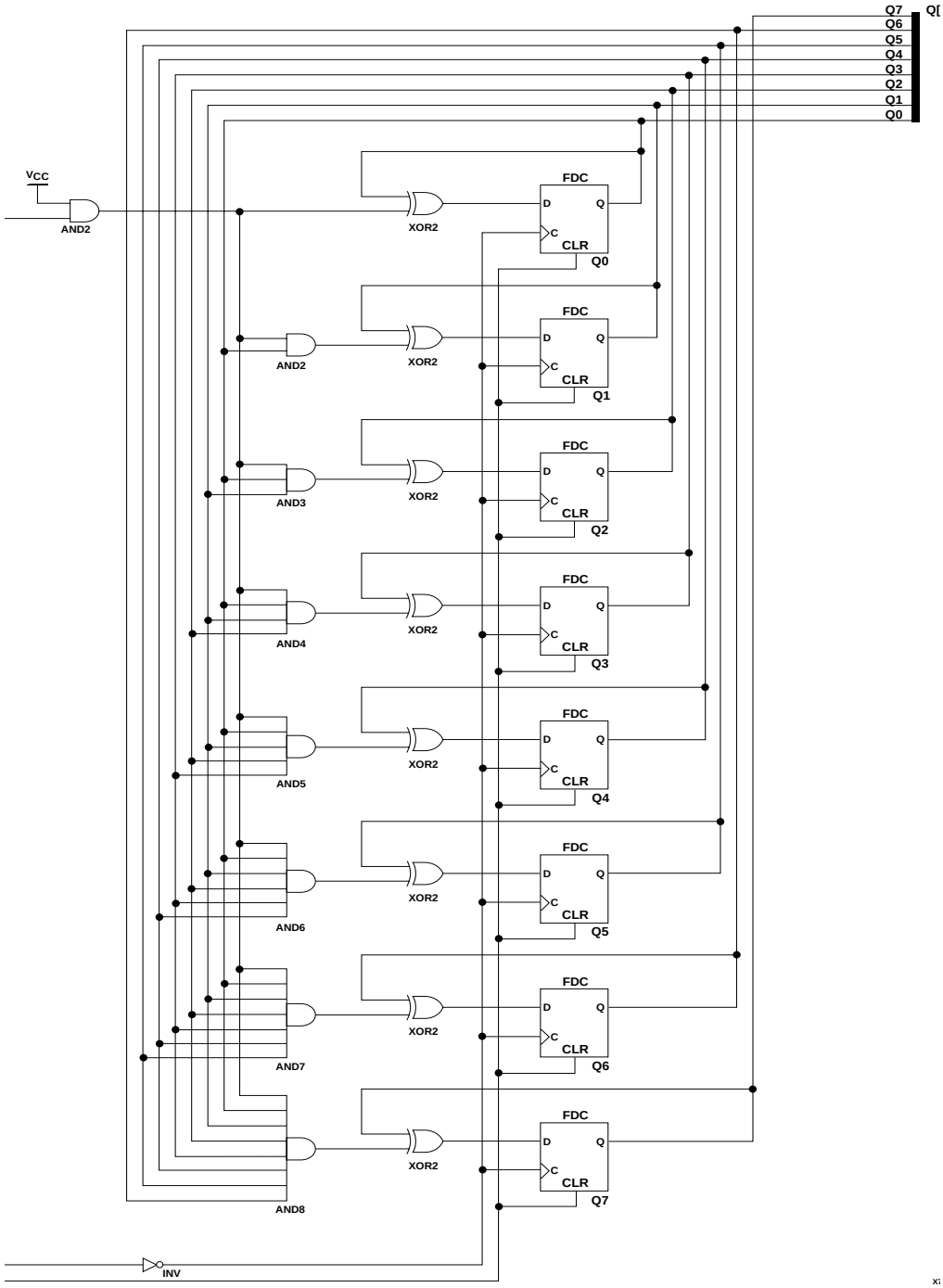
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	CE	C	Qz – Q0
1	X	X	0
0	0	X	No Chg
0	1	↓	Inc

z = 7 for CR8CE; z = 15 for CR16CE.



CR8CE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



CR8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cr8ce is

begin

process(C, CLR)
begin
        if (CLR='1') then
                Q <= (others => '0');
        elsif (C'event and C='0') then
                if (CE='1') then
                        Q <= Q + 1;
                end if;
        end if;
end process;

end Behavioral;
```

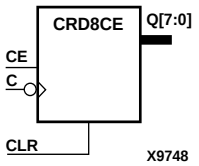
Verilog Inference Code

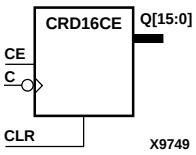
```
always @(posedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (CE)
                Q <= Q + 1;
end
```

CRD8CE, CRD16CE

8-, 16-Bit Dual-Edge Triggered Binary Ripple Counters with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro





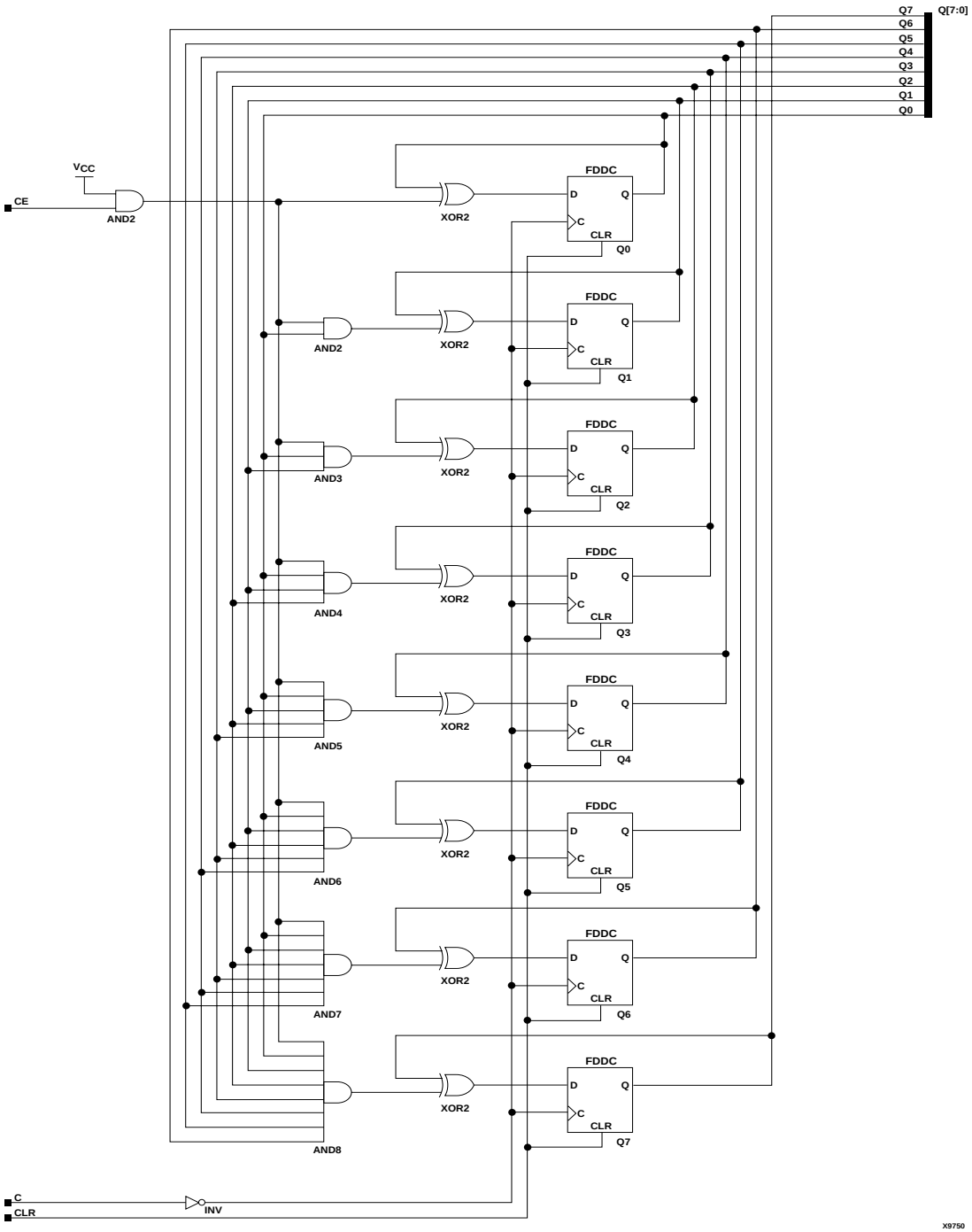
CRD8CE and CRD16CE are dual edge triggered 8-bit and 16-bit, cascadable, clearable, binary, ripple counters. The asynchronous clear (CLR), when High, overrides all other inputs and causes the Q outputs to go to logic level zero. The counter increments when the clock enable input (CE) is High during the High-to-Low and Low-to-High clock (C) transitions. The counter ignores clock transitions when CE is Low.

Larger counters can be created by connecting the last Q output (Q7 for CRD8CE, Q15 for CRD16CE) of the first stage to the clock input of the next stage. CLR and CE inputs are connected in parallel. The clock period is not affected by the overall length of a ripple counter. The overall clock-to-output propagation is $n(t_{C\text{-}Q})$, where n is the number of stages and the time $t_{C\text{-}Q}$ is the C-to-Qz propagation delay of each stage.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
CLR	CE	C	Qz – Q0
1	X	X	0
0	0	X	No Chg
0	1	↑	Inc
0	1	↓	Inc

z = 7 for CR8CE; z = 15 for CR16CE.



CRD8CE Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of crd8ce is

begin

process(C, CLR)
begin
        if (CLR='1') then
                Q <= (others => '0');
        elsif (C'event) then
                if (CE='1') then
                        Q <= Q + 1;
                end if;
        end if;
end process;

end Behavioral;
```

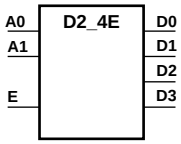
Verilog Inference Code

```
always @(posedge C or negedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (CE)
                Q <= Q + 1;
end
```

D2_4E

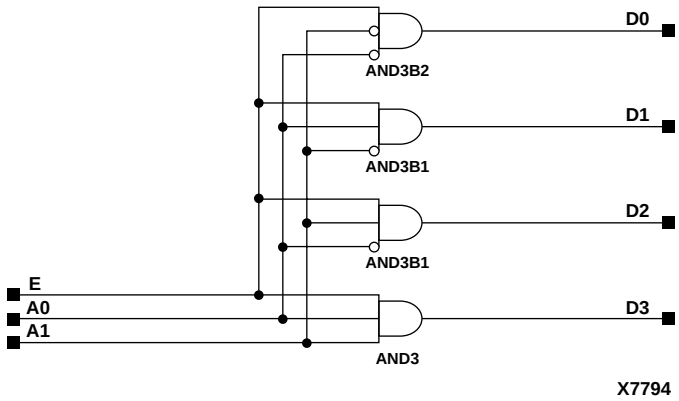
2- to 4-Line Decoder/Demultiplexer with Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



When the enable (E) input of the D2_4E decoder / demultiplexer is High, one of four active-High outputs (D3 – D0) is selected with a 2-bit binary address (A1 – A0) input. The non-selected outputs are Low. Also, when the E input is Low, all outputs are Low. In demultiplexer applications, the E input is the data input.

Inputs			Outputs			
A1	A0	E	D3	D2	D1	D0
X	X	0	0	0	0	0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0



D2_4E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of d2_4e is

begin

process (A, E)
begin
 if (E='0') then
  D <= "0000";
 else
  case A is
       when "00" =>
          D <= "0001";
       when "01" =>
          D <= "0010";
       when "10" =>
          D <= "0100";
       when "11" =>
          D <= "1000";
   when others =>
    D <= "0000";
  end case;
 end if;
end process;

end Behavioral;
```

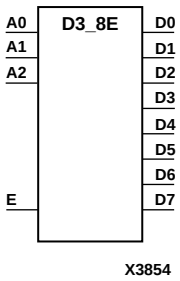
Verilog Inference Code

```
always @ (A or E)
begin
 if (!E)
  D <= 4'b0000;
 else
 begin
  case (A)
     2'b00 : D <= 4'b0001;
     2'b01 : D <= 4'b0010;
     2'b10 : D <= 4'b0100;
     2'b11 : D <= 4'b1000;
  endcase
 end
end
```

D3_8E

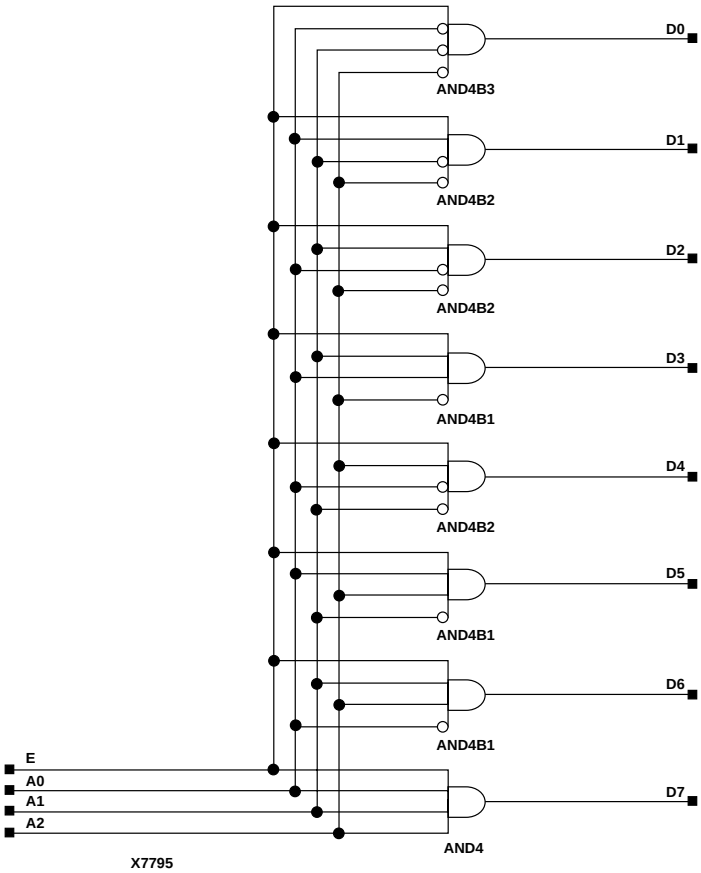
3- to 8-Line Decoder/Demultiplexer with Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



When the enable (E) input of the D3_8E decoder / demultiplexer is High, one of eight active-High outputs (D7 – D0) is selected with a 3-bit binary address (A2 – A0) input. The non-selected outputs are Low. Also, when the E input is Low, all outputs are Low. In demultiplexer applications, the E input is the data input.

Inputs				Outputs							
A2	A1	A0	E	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	0	0	1	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	1	1	0	0	1	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



D3_8E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of d3_8e is

begin

process (A, E)
begin
 if (E='0') then
  D <= "00000000";
 else
  case A is
       when "000" =>
           D <= "00000001";
       when "001" =>
           D <= "00000010";
       when "010" =>
           D <= "00000100";
       when "011" =>
```

```
            D <= "00001000";
        when "100" =>
            D <= "00010000";
        when "101" =>
            D <= "00100000";
        when "110" =>
            D <= "01000000";
        when "111" =>
            D <= "10000000";
   when others =>
    D <= "00000000";
   end case;
 end if;
end process;

end Behavioral;
```

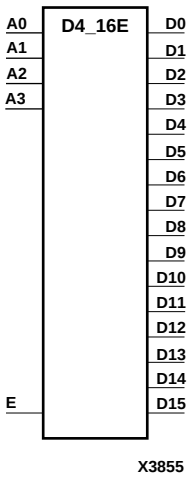
Verilog Inference Code

```
always @ (A or E)
begin
 if (!E)
  D <= 8'b00000000;
 else
 begin
  case (A)
     3'b000 : D <= 8'b00000001;
     3'b001 : D <= 8'b00000010;
     3'b010 : D <= 8'b00000100;
     3'b011 : D <= 8'b00001000;
     3'b100 : D <= 8'b00010000;
     3'b101 : D <= 8'b00100000;
     3'b110 : D <= 8'b01000000;
     3'b111 : D <= 8'b10000000;
  endcase
 end
end
```

D4_16E

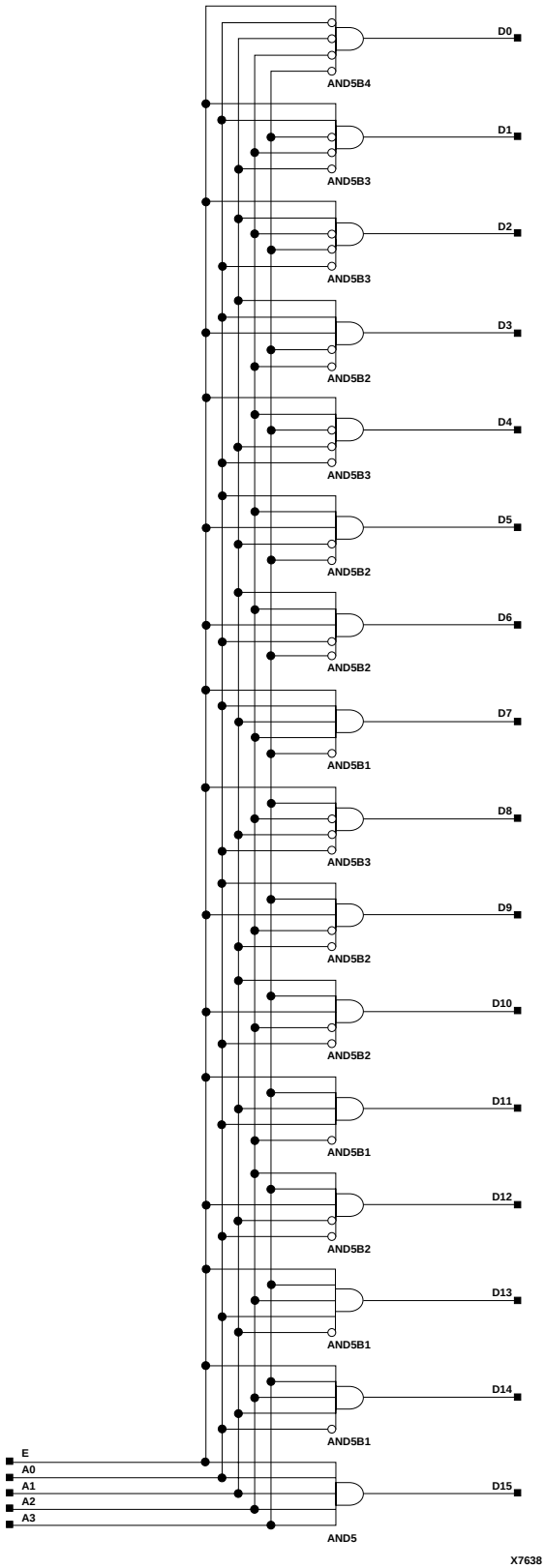
4- to 16-Line Decoder/Demultiplexer with Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



When the enable (E) input of the D4_16E decoder / demultiplexer is High, one of 16 active-High outputs (D15 – D0) is selected with a 4-bit binary address (A3 – A0) input. The non-selected outputs are Low. Also, when the E input is Low, all outputs are Low. In demultiplexer applications, the E input is the data input.

See the "D3_8E" section for a representative truth table derivation.



D4_16E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of d4_16e is

begin
process (A, E)
begin
 if (E='0') then
  D <= "0000000000000000";
 else
  case A is
       when "0000" =>
           D <= "0000000000000001";
       when "0001" =>
           D <= "0000000000000010";
       when "0010" =>
           D <= "0000000000000100";
       when "0011" =>
           D <= "0000000000001000";
       when "0100" =>
           D <= "0000000000010000";
       when "0101" =>
           D <= "0000000000100000";
       when "0110" =>
           D <= "0000000001000000";
       when "0111" =>
           D <= "0000000010000000";
       when "1000" =>
           D <= "0000000100000000";
       when "1001" =>
           D <= "0000001000000000";
       when "1010" =>
           D <= "0000010000000000";
       when "1011" =>
           D <= "0000100000000000";
       when "1100" =>
           D <= "0001000000000000";
       when "1101" =>
           D <= "0010000000000000";
       when "1110" =>
           D <= "0100000000000000";
       when "1111" =>
           D <= "1000000000000000";
  when others =>
   D <= "0000000000000000";
  end case;
 end if;
end process;

end Behavioral;
```

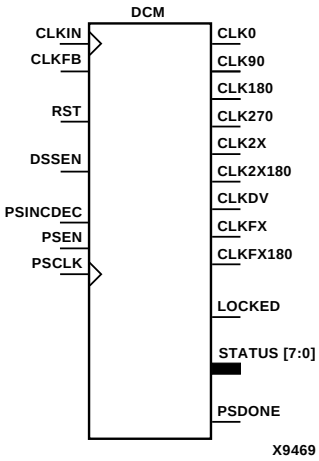
Verilog Inference Code

```
always @ (A or E)
begin
 if (!E)
  D <= 16'b0000000000000000;
 else
 begin
  case (A)
     4'b0000 : D <= 16'b0000000000000001;
     4'b0001 : D <= 16'b0000000000000010;
     4'b0010 : D <= 16'b0000000000000100;
     4'b0011 : D <= 16'b0000000000001000;
     4'b0100 : D <= 16'b0000000000010000;
     4'b0101 : D <= 16'b0000000000100000;
     4'b0110 : D <= 16'b0000000001000000;
     4'b0111 : D <= 16'b0000000010000000;
     4'b1000 : D <= 16'b0000000100000000;
     4'b1001 : D <= 16'b0000001000000000;
     4'b1010 : D <= 16'b0000010000000000;
     4'b1011 : D <= 16'b0000100000000000;
     4'b1100 : D <= 16'b0001000000000000;
     4'b1101 : D <= 16'b0010000000000000;
     4'b1110 : D <= 16'b0100000000000000;
     4'b1111 : D <= 16'b1000000000000000;
  endcase
 end
end
```

DCM

Digital Clock Manager

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A



DCM is a digital clock manager that provides multiple functions. It can implement a clock delay locked loop, a digital frequency synthesizer, digital phase shifter, and a digital spread spectrum.

Note All unused inputs must be driven Low. The program will automatically tie the inputs Low if they are unused.

Clock Delay Locked Loop (DLL)

DCM includes a clock delay locked loop used to minimize clock skew for Virtex-II and Virtex-II Pro devices. DCM synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specified time (ps) of each other. See *The Programmable Logic Data Book* for the specified time value.

DCM supports two frequency modes for the DLL. By default, the DLL_FREQUENCY_MODE attribute is set to Low and the frequency of the clock signal at the CLKIN input must be in the Low (DLL_CLKIN_MIN_LF to DLL_CLKIN_MAX_LF) frequency range (MHz). See the *The Programmable Logic Data Book* for the current DLL_CLKIN_MIN_LF to DLL_CLKIN_MAX_LF frequency range values. In Low frequency mode, the CLK0, CLK90, CLK180, CLK270, CLK2X, CLKDV, and CLK2X180 outputs are available. When the DLL_FREQUENCY_MODE attribute is set to High, the frequency of the clock signal at the CLKIN input must be in the High (DLL_CLKIN_MIN_HF to DLL_CLKIN_MAX_HF) frequency range (MHz). See the *The Programmable Logic Data Book* for the current DLL_CLKIN_MIN_HF to DLL_CLKIN_MAX_HF frequency range values. In High frequency mode, only the CLK0, CLK180, and CLKDV outputs are available.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG connected to the CLKFB input of the DCM must be sourced from either the CLK0 or CLK2X outputs of the same DCM. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Either the CLK0 or CLK2X output can be used but not both. The CLK0 or CLK2X must be connected to the input of OBUF, an output buffer. The CLK_FEEDBACK attribute controls whether the CLK0 output, the default, or the CLK2X output is the source of the CLKFB input.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted outputs (CLK90, CLK180, and CLK270) is the same as that of the CLK0 output. The duty cycle of the CLK2X, CLK2X180, and CLKDV outputs is 50-50 unless CLKDV_DIVIDE is a non-integer and the DLL_FREQUENCY_MODE is High (see the "CLKDV_DIVIDE" section of the *Constraints Guide* for details). The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

DCM Clock Delay Lock Loop Outputs

Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLK0 frequency, shifted 180^{o} with regards to CLK0
CLK270*	Clock at 1x CLK0 frequency, shifted 270^{o} with regards to CLK0
CLK2X*	Clock at 2x CLK0 frequency, in phase with CLK0
CLK2X180*	Clock at 2x CLK0 frequency shifted 180^{o} with regards to CLK2X
CLK90*	Clock at 1x CLK0 frequency, shifted 90^{o} with regards to CLK0
CLKDV	Clock at (1 / n) x CLK0 frequency, where n=CLKDV_DIVIDE value. CLKDV is in phase with CLK0.
LOCKED	All enabled DCM features locked.

* The CLK90, CLK270, CLK2X, and CLK2X180 outputs are *not* available if the DLL_FREQUENCY_MODE is set to High.

Digital Frequency Synthesizer (DFS)

The CLKFX and CLKFX180 outputs in conjunction with the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes provide a frequency synthesizer that can be any multiple or division of CLKIN. CLKFX and CLKIN are in phase every CLKFX_MULTIPLY cycles of CLKFX and every CLKFX_DIVIDE cycles of CLKIN when a feedback is provided to the CLKFB input of the DLL. The frequency of CLKFX is defined by the following equation.

$$Frequency_{CLKFX} = (CLKFX\_MULTIPLY\_value / CLKFX\_DIVIDE\_value) * Frequency_{CLKIN}$$

Both the CLKFX or CLKFX180 output can be used simultaneously.

CLKFX180 is 1x the CLKFX frequency, shifted 180^{o} with regards to CLKFX. CLKFX and CLKFX180 always have a 50 / 50 duty cycle.

The DFS_FREQUENCY_MODE attribute specifies the allowable input clock and output clock frequency ranges.

The CLK_FEEDBACK attribute set to NONE will cause the DCM to be in the Digital Frequency Synthesizer mode. The CLKFX and CLKFX180 will be generated without phase correction with respect to CLKIN.

See *The Programmable Logic Data Book* for the allowable frequency range of CLKFX.

Digital Phase Shifter (DPS)

The phase shift (skew) between the rising edges of CLKIN and CLKFB may be configured as a fraction of the CLKIN period with the PHASE_SHIFT attribute. This allows the phase shift to remain constant as ambient conditions change. The CLKOUT_PHASE_SHIFT attribute controls the use of the PHASE_SHIFT value. By default, the CLKOUT_PHASE_SHIFT attribute is set to NONE and the PHASE_SHIFT attribute has no effect.

Note By creating skew between CLKIN and CLKFB, all DCM output clocks are phase shifted by the amount of the skew.

When the CLKOUT_PHASE_SHIFT attribute is set to FIXED, the skew set by the PHASE_SHIFT attribute is used at configuration for the rising edges of CLKIN and CLKFB. The skew remains constant.

When the CLKOUT_PHASE_SHIFT attribute is set to VARIABLE, the skew set at configuration is used as a starting point and the skew value can be changed dynamically during operation using the PS* signals. This digital phase shifter feature is controlled by a synchronous interface. The inputs PSEN (phase shift enable) and PSINCDEC (phase shift increment / decrement) are set up to the rising edge of PSCLK (phase shift clock). The PSDONE (phase shift done) output is clocked with the rising edge of PSCLK (the phase shift clock). PSDONE must be connected to implement the complete synchronous interface. The rising-edge skew between CLKIN and CLKFB may be dynamically adjusted after the LOCKED output goes High. The PHASE_SHIFT attribute value specifies the initial phase shift amount when the device is configured. Then the PHASE_SHIFT value is changed one unit when PSEN is activated for one period of PSCLK. The PHASE_SHIFT value is incremented when PSINCDEC is High and decremented when PSINCDEC is Low during the period that PSEN is High. When the DCM completes an increment or decrement operation, the PSDONE output goes High for a single PSCLK cycle to indicate the operation is complete. At this point the next change may be made. When RST (reset) is High, the PHASE_SHIFT attribute value is reset to the skew value set at configuration.

Note If CLKOUT_PHASE_SHIFT is FIXED or NONE, the PSEN, PSINCDEC, and PSCLK inputs must be tied to GND. The program will automatically tie the inputs to GND if they are not connected by the user.

Digital Spread Spectrum (DSS)

The digital spread spectrum function of DCM broadens the frequency spectrum of the output clocks to help meet FCC requirements. Spread spectrum clocking is a simple method to lower electromagnetic interference (EMI). A digital system can create a severe EMI spike at the clock frequency. Spread spectrum clocking speeds up and slows down the clock within a few percent of the target frequency, thus flattening out the EMI peak by spreading it across a range of frequencies.

When the DSS_MODE attribute is set to NONE, the DSSEN input has no effect. When DSS_MODE is set to a value other than NONE and DSSEN is High, the output clock frequency is modulated by the amount of spread specified by the DSS_MODE value.

For Virtex-II and Virtex-II Pro, the DSS feature of the DCM has limited EMI reduction capability.

For maximum EMI reduction an IP core available from Xilinx is recommended.

Additional Status Bits

The STATUS output bits return the following information.

DCM Additional Status Bits

Bit	Description
0	Phase Shift Overflow* 1 = |PHASE_SHIFT| > 255
1	DLL CLKIN stopped** 1 = CLKIN stopped toggling
2	DLL CLKFX stopped 1 = CLKFX stopped toggling
3	N/A
4	N/A
5	N/A
6	N/A
7	N/A

* Phase Shift Overflow will also go high if the end of the phase shift delay line is reached (see the product data sheet for the most current value of the maximum shifting delay).

** If only the DFS outputs are used (CLKFX & CLKFX180), this status bit will not go high if CLKIN stops.

LOCKED

When LOCKED is high, all enabled signals are locked.

RST

The master reset input (RST) resets DCM to its initial (power-on) state. The signal at the RST input is asynchronous and must be held High for 2ns.

Usage

This component is generally instantiated in the code as it cannot be easily inferred in synthesis tools. Some synthesis tools may allow inference via an attribute. Refer to your synthesis tool documentation if that is desirable.

VHDL Instantiation Template

```
-- Component Declaration for DCM should be placed
-- after architecture statement but before begin keyword

component DCM
  -- synthesis translate_off
  generic (CLK_FEEDBACK :string := "1X";
           CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 8.0, 16.0)
           CLKFX_DIVIDE : integer :=1; -- (1 to 4096)
           CLKFX_MULTIPLY : integer := 4; -- (1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 5.5,
            -- 6.0, 6.5, 7.0, 7.5, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0)
           CLKIN_DIVIDE_BY_2 : boolean := FALSE; -- (TRUE, FALSE)
           CLKOUT_PHASE_SHIFT: string := "NONE";
           DESKEW_ADJUST: string := "SYSTEM_SYNCHRONOUS";
           DFS_FREQUENCY_MODE: string := "LOW";
           DLL_FREQUENCY_MODE: string := "LOW";
           DSS_MODE: string := "NONE";
           DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
           FACTORY_JF : bit_vector := X"C080";
           PHASE_SHIFT: real := 0;
           STARTUP_WAIT :boolean := FALSE); -- (TRUE, FALSE)
  -- synthesis translate_on
  port (CLK0      : out STD_ULOGIC;
        CLK180    : out STD_ULOGIC;
        CLK270    : out STD_ULOGIC;
        CLK2X     : out STD_ULOGIC;
        CLK2X180  : out STD_ULOGIC;
        CLK90     : out STD_ULOGIC;
        CLKDV     : out STD_ULOGIC;
        CLKFX     : out STD_ULOGIC;
        CLKFX180  : out STD_ULOGIC;
        LOCKED    : out STD_ULOGIC;
        PSDONE    : out STD_ULOGIC;
        STATUS    : out STD_ULOGIC;
        CLKFB     : in STD_ULOGIC;
        CLKIN     : in STD_ULOGIC;
        DSSEN   : in STD_ULOGIC;
        PSCLK   : in STD_ULOGIC;
        PSEN    : in STD_ULOGIC;
        PSINCDEC: in STD_ULOGIC;
        RST     : in STD_ULOGIC);
end component;

-- Component Attribute specification for DCM
-- should be placed after architecture declaration but
-- before the begin keyword

attribute CLK_FEEDBACK :  string;
attribute CLKDV_DIVIDE : real;
attribute CLKFX_DIVIDE : integer;
attribute CLKFX_MULTIPLY: integer;
attribute CLKIN_DIVIDE_BY_2:  string;
attribute CLKOUT_PHASE_SHIFT: string;
```

```
attribute DESKEW_ADJUST : string;
attribute DFS_FREQUENCY_MODE: string;
attribute DLL_FREQUENCY_MODE: string;
attribute DSS_MODE: string;
attribute DUTY_CYCLE_CORRECTION : string;
attribute FACTORY_JF : bit_vector;
attribute PHASE_SHIFT: real;
attribute STARTUP_WAIT : boolean;


attribute CLK_FEEDBACK of DCM_instance_name: label is "1X";
attribute CLKDV_DIVIDE of DCM_instance_name: label is 2.0;
-- (1.5,2,2.5,3,4, 5, 8, 16) are valid for CLKDV_DIVIDE
attribute CLKFX_DIVIDE of DCM_instance_name: label is 1;
attribute CLKFX_MULTIPLY of DCM_instance_name: label is 4;
attribute CLKIN_DIVIDE_BY_2 of DCM_instance_name: label is "FALSE";
attribute CLKOUT_PHASE_SHIFT of DCM_instance_name: label is "NONE";
attribute DESKEW_ADJUST of DCM_instance_name : label is "SYSTEM_SYNCHRONOUS";
attribute DFS_FREQUENCY_MODE of DCM_instance_name: label is "LOW";
attribute DLL_FREQUENCY_MODE of DCM_instance_name: label is "LOW";
attribute DSS_MODE of DCM_instance_name: label is "NONE";
attribute DUTY_CYCLE_CORRECTION of DCM_instance_name: label is TRUE;
-- (TRUE, FALSE) are valid for DUTY_CYCLE_CORRECTION
attribute FACTORY_JF of DCM_instance_name : label is X"C080";
attribute PHASE_SHIFT of DCM_instance_name: label is 0;
attribute STARTUP_WAIT of DCM_instance_name: label is FALSE; -- (TRUE,FALSE)


-- Component Instantiation for DCM should be placed
-- in architecture after the begin keyword

DCM_INSTANCE_NAME : DCM
  -- synthesis translate_off
  generic map(CLK_FEEDBACK => "string_value",
              CLKDV_DIVIDE  => real_value, -- (1.5,2,2.5,3,4,5,8,16)
              CLKFX_DIVIDE => integer_value,
              CLKFX_MULTIPLY => integer_value,
              CLKIN_DIVIDE_BY_2 => boolean_value, -- (TRUE, FALSE)
              CLKOUT_PHASE_SHIFT => "string_value",
              DESKEW_ADJUST => "string_value",
              DFS_FREQUENCY_MODE => "string_value",
              DLL_FREQUENCY_MODE => "string_value",
              DSS_MODE=> "string_value",
              DUTY_CYCLE_CORRECTION => boolean_value, -- (TRUE, FALSE)
              FACTORY_JF=> "bit_vector_value",
              PHASE_SHIFT=> integer_value,
              STARTUP_WAIT=> boolean); -- (TRUE, FALSE)
  -- synthesis translate_on
  port map  (CLK0     => user_CLK0,
             CLK180   => user_CLK180,
             CLK270   => user_CLK270,
             CLK2X    => user_CLK2X,
             CLK2X180 => user_CLK2X180,
             CLK90    => user_CLK90,
```

```
              CLKDV    => user_CLKDV,
              CLKFX    => user_CLKFX,
              CLKFX180 => user_CLKFX180,
              LOCKED   => user_LOCKED)
              PSDONE   => user_PSDONE,
              STATUS   => user_STATUS,
              CLKFB    => user_CLKFB,
              CLKIN    => user_CLKIN,
              DSSEN    => user_DSSEN,
              PSCLK    => user_PSCLK,
              PSEN     => user_PSEN,
              PSINCDEC => user_PSINCDEC,
              RST      => user_RST);
```

Verilog Instantiation Template

```
DCM DCM_instance_name (.CLK0 (user_CLK0),
                       .CLK180 (user_CLK180),
                       .CLK270 (user_CLK270),
                       .CLK2X (user_CLK2X),
                       .CLK2X180(user_CLK2X180),
                       .CLK90 (user_CLK90),
                       .CLKDV (user_CLKDV),
                       .CLKFX (user_CLKFX),
                       .CLKFX180(user_CLKFX180),
                       .LOCKED (user_LOCKED),
                       .PSDONE(user_PSDONE),
                       .STATUS(user_STATUS),
                       .CLKFB (user_CLKFB),
                       .CLKIN (user_CLKIN),
                       .DSSEN(user_DSSEN),
                       .PSCLK (user_PSCLK),
                       .PSEN  (user_PSEN),
                       .PSINCDEC(user_PSINCDEC),
                       .RST (user_RST));

defparam DCM_instance_name.CLK_FEEDBACK => "string_value";
defparam DCM_instance_name.CLKDV_DIVIDE = integer_value; //(1.5,2,2.5,3,4,5,8,16)
defparam DCM_instance_name.CLKFX_DIVIDE => integer_value;
defparam DCM_instance_name.CLKFX_MULTIPLY => integer_value;
defparam DCM_instance_name.CLKIN_DIVIDE_BY_2 => boolean_value; // (TRUE, FALSE)
defparam DCM_instance_name.CLKOUT_PHASE_SHIFT => "string_value";
defparam DCM_instance_name.DESKEW_ADJUST => "string_value";
defparam DCM_instance_name.DFS_FREQUENCY_MODE => "string_value";
defparam DCM_instance_name.DLL_FREQUENCY_MODE => "string_value";
defparam DCM_instance_name.DSS_MODE=> "string_value";
defparam DCM_instance_name.DUTY_CYCLE_CORRECTION = "string_value";// (TRUE, FALSE)
defparam DCM_instance_name.FACTORY_JF=> "bit_vector_value";
defparam DCM_instance_name.PHASE_SHIFT=> integer_value;
defparam DCM_instance_name.STARTUP_WAIT = boolean_value; // (TRUE, FALSE)
```

Note Additional syntax may be necessary in order to pass the DCM attributes via the synthesis tool and the above defparam statements may need to be isolated from the synthesis tool with translate_off / translate_on directives. Please refer to your synthesis tool documentation for more information on Verilog attribute passing to ensure that you properly pass these attributes to the synthesis tool; or else you may pass these attributes to the UCF file.

Commonly Used Constraints

CLKDV_DIVIDE

CLK_FEEDBACK

CLKFX_DIVIDE

CLKFX_MULTIPLY

CLKIN_DIVIDE_BY_2

CLKOUT_PHASE_SHIFT

DUTY_CYCLE_CORRECTION

DFS_FREQUENCY_MODE

DLL_FREQUENCY_MODE

LOC

PHASE_SHIFT

STARTUP_WAIT

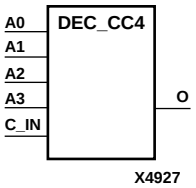
FACTORY_JF

DESKEW_ADJUST

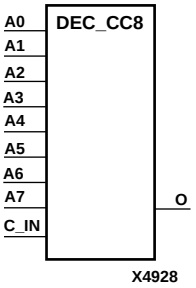
DEC_CC4, 8, 16

4-, 8-, 16-Bit Active Low Decoders

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A

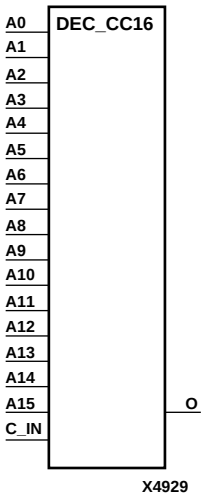


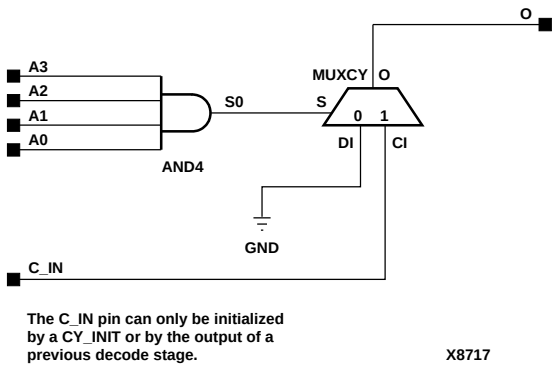
These decoders are used to build wide-decoder functions. They are implemented by cascading CY_MUX elements driven by lookup tables (LUTs). The C_IN pin can only be driven by the output (O) of a previous decode stage. When one or more of the inputs (A) are Low, the output is Low. When all the inputs are High and the C_IN input is High, the output is High. You can decode patterns by adding inverters to inputs.



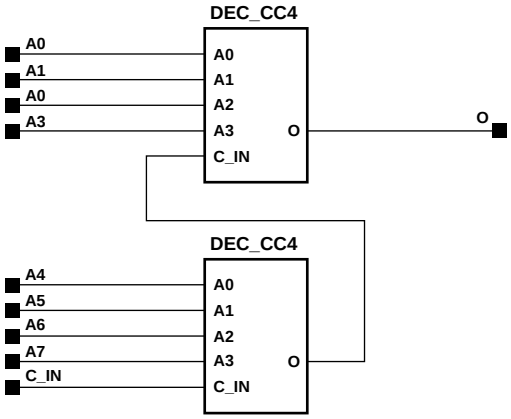
Inputs					Outputs
A0	A1	...	Az	C_IN	O
1	1	1	1	1	1
X	X	X	X	0	0
0	X	X	X	X	0
X	0	X	X	X	0
X	X	X	0	X	0

z = 3 for DEC_CC4; z = 7 for DEC_CC8; z = 15 for DEC_CC16





DEC_CC4 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



DEC_CC8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

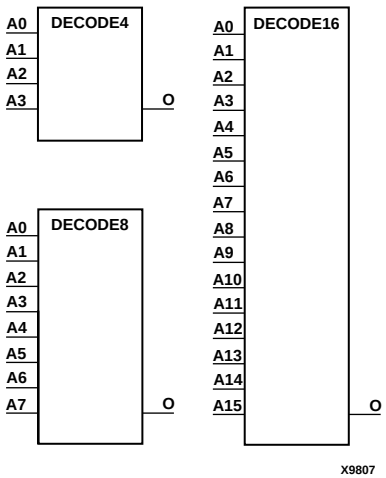
Usage

DEC_CC4 cannot be directly inferred or instantiated. The proper way to use a DEC_CC4 is to infer the primitive components that make up the DEC_CC4.

DECODE4, 8, 16

4-, 8-, 16-Bit Active-Low Decoders

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



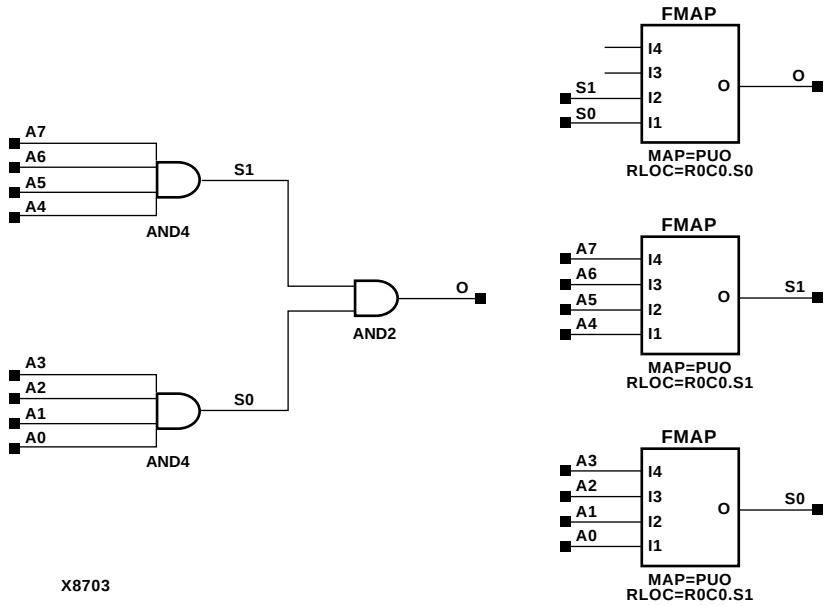
DECODE Representations

In Spartan-II, , Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, decoders are implemented using combinations of LUTs and MUXCYs.

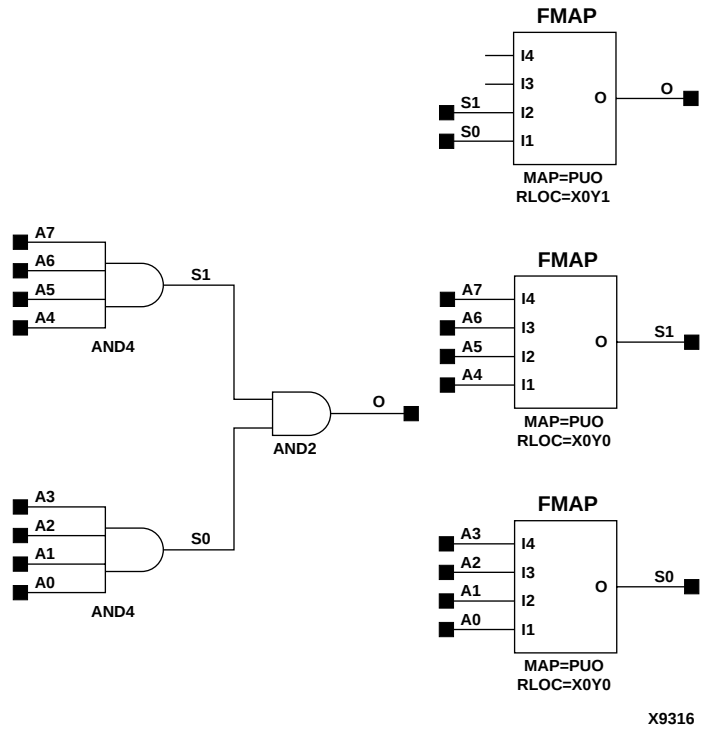
Inputs				Outputs*
A0	A1	...	Az	O
1	1	1	1	1
0	X	X	X	0
X	0	X	X	0
X	X	X	0	0

z = 3 for DECODE4, z = 7 for DECODE8; z = 15 for DECODE16

*A pull-up resistor must be connected to the output to establish High-level drive current.



DECODE8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



DECODE8 Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferrred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of decode4 is

begin

process (A)
begin
        case A is
                when "1111" => O <= ’1’;
                when others => O <= ’0’;
        end case;
end process;

end Behavioral;
```

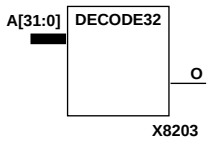
Verilog Inference Code

```
always @(A)
begin
        case (A)
                4’h0 : O <= 1;
                default : O <= 0;
        endcase
end
```

DECODE32, 64

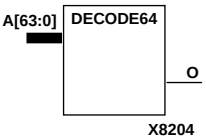
32- and 64-Bit Active-Low Decoders

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



DECODE32 and DECODE64 are 32- and 64-bit active-low decoders. These decoders are implemented using combinations of LUTs and MUXCYs.

See the DECODE4, 8, 16 section for a representative schematic.



Inputs				Outputs
A0	A1	...	Az	O
1	1	1	1	1
0	X	X	X	0
X	0	X	X	0
X	X	X	0	0

z = 31 for DECODE32
z = 63 for DECODE64

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of decode32 is

begin

process (A)
begin
        case A is
                when x"11111111" => O <= '1';
                when others => O <= '0';
        end case;
end process;

end Behavioral;
```

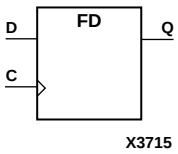
Verilog Inference Code

```
always @(A)
begin
        case (A)
                32’h00000000 : O <= 1;
                default : O <= 0;
        endcase
end
```

FD

D Flip-Flop

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FD is a single D-type flip-flop with data input (D) and data output (Q). The data on the D inputs is loaded into the flip-flop during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

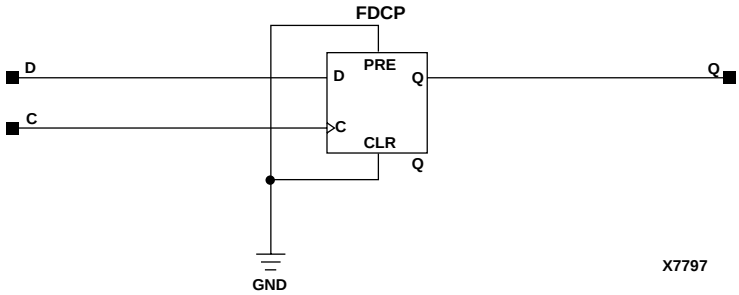
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

See the "FD4, 8, 16" section for information on multiple D flip-flops for XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1



FD Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fd is
begin
 process (C)
 begin
  if C’event and C=’1’ then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C) begin
   Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FD should be placed
-- after architecture statement but before begin keyword

component FD
   -- synthesis translate_off
   generic (
        INIT  : bit := ’1’);
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FD
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FD_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FD should be placed
-- in architecture after the begin keyword

FD_INSTANCE_NAME : FD
   -- synthesis translate_off
   generic  (
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 D => user_D);
```

Verilog Instantiation Template

```
FD FD_instance_name (.Q (user_Q),
                     .C (user_C),
                     .D (user_D));

defparam FD_instance_name.INIT = bit_value;
```

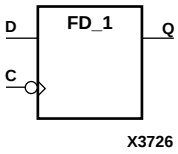
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FD_1

D Flip-Flop with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FD_1 is a single D-type flip-flop with data input (D) and data output (Q). The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fd_1 is
begin
 process (C)
 begin
  if C'event and C='0' then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C) begin
   Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FD_1 should be placed
-- after architecture statement but before begin keyword

component FD_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FD_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FD_1_instance_name : label is "0";
-- values can be (0 or 1)


begin

FD_1_INSTANCE_NAME : FD_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value); -- INIT value can be '0' or '1'
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 D => user_D);
end Behavioral
```

Verilog Instantiation Template

```
FD_1 FD_1_instance_name (.Q (user_Q),
                         .C (user_C),
                         .D (user_D));

defparam FD_1_instance_name.INIT = bit_value;
```

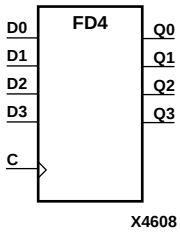
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FD4, 8, 16

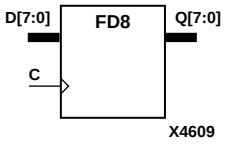
Multiple D Flip-Flops

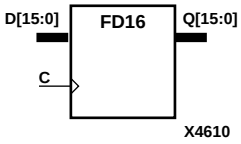
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



FD4, FD8, FD16 are multiple D-type flip-flops with data inputs (D) and data outputs (Q). FD4, FD8, and FD16 are, respectively, 4-bit, 8-bit, and 16-bit registers, each with a common clock (C). The data on the D inputs is loaded into the flip-flop during the Low-to-High clock (C) transition.

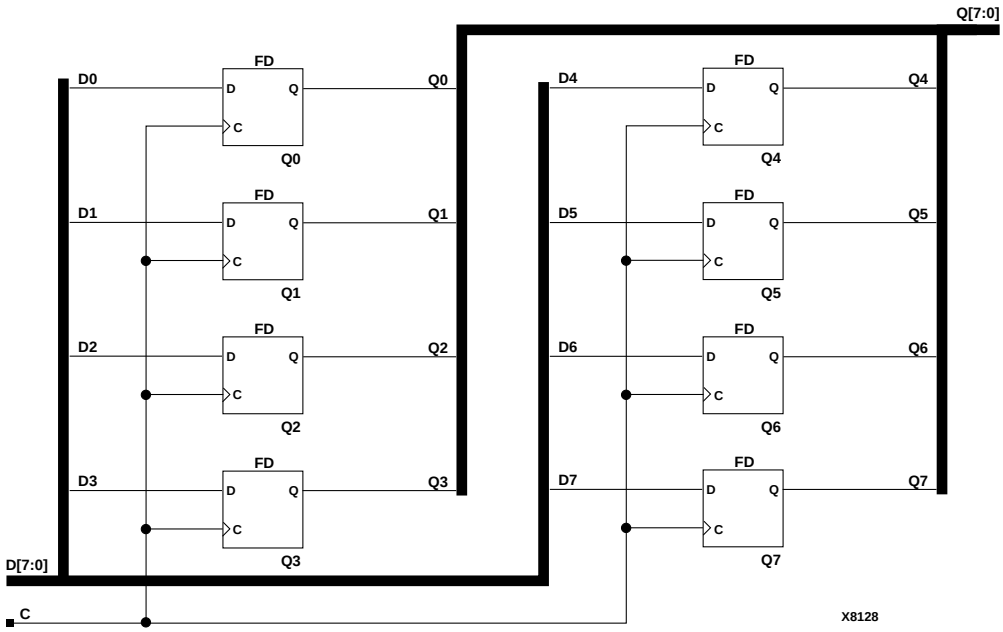
The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.





Inputs		Outputs
Dz – D0	C	Qz – Q0
0	↑	0
1	↑	1

z = 3 for FD4; z = 7 for FD8; z = 15 for FD16



FD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fd4 is
begin
 process (C)
 begin
  if C’event and C=’1’ then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

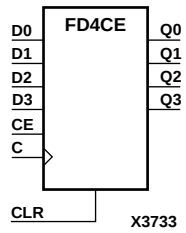
Verilog Inference Code

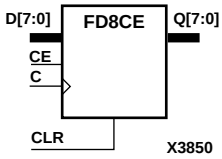
```
always @ (posedge C)
 begin
  Q<=D;
 end
```

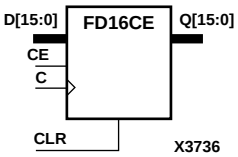
FD4CE, FD8CE, FD16CE

4-, 8-, 16-Bit Data Registers with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







FD4CE, FD8CE, and FD16CE are, respectively, 4-, 8-, and 16-bit data registers with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q) during the Low-to-High clock (C) transition. When CLR is High, it overrides all other inputs and resets the data outputs (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flops are asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

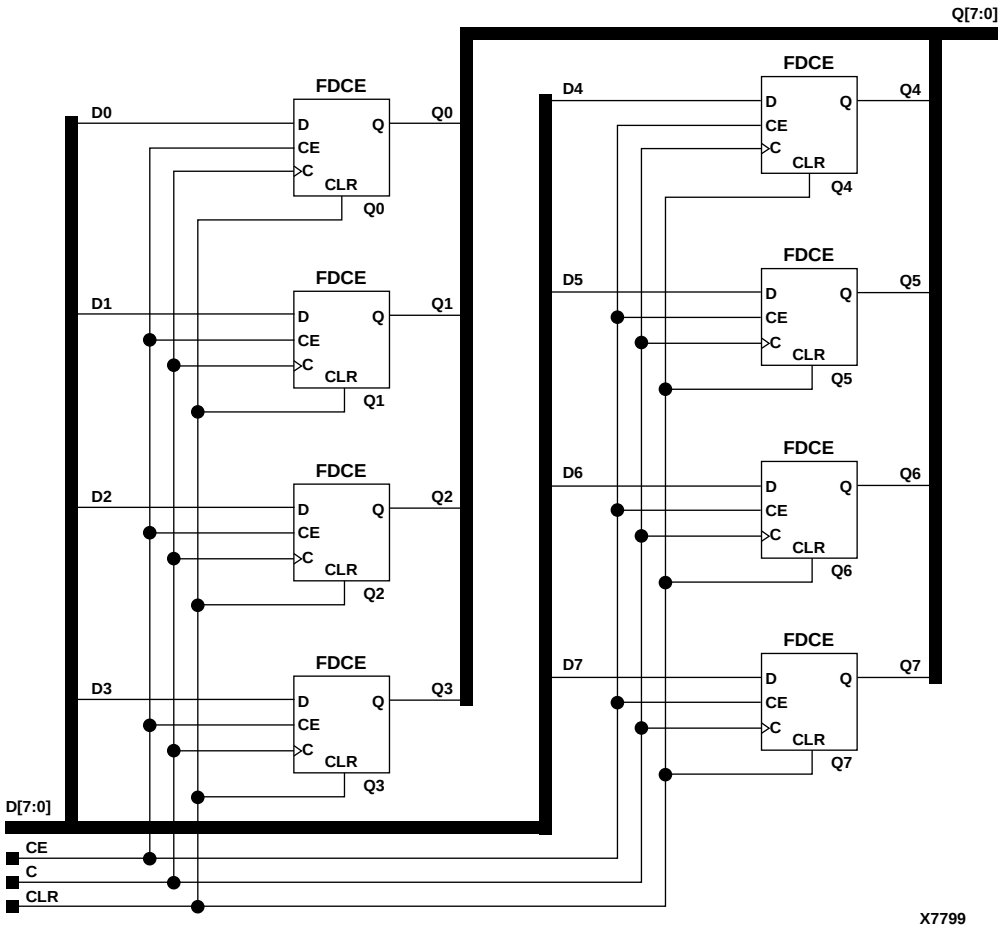
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	CE	Dz – D0	C	Qz – Q0
1	X	X	X	0
0	0	X	X	No Chg
0	1	Dn	↑	dn

z = 3 for FD4CE; z = 7 for FD8CE; z = 15 for FD16CE.

dn = state of corresponding input (Dn) one setup time prior to active clock transition



FD8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fd4ce is
begin
 process (C, CLR) begin
  if (CLR = '1') then
   Q <= "0000";
  elsif (C' event and C = '1') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

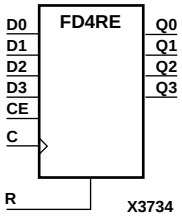
Verilog Inference Code

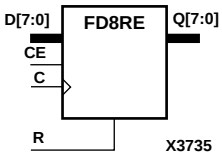
```
always @ (posedge C or posedge CLR)
 begin
  if (CLR)
   Q<=4'b0000;
  else
   begin
    if (CE)
     Q<=D;
   end
 end
```

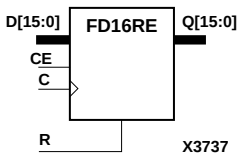
FD4RE, FD8RE, FD16RE

4-, 8-, 16-Bit Data Registers with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







FD4RE, FD8RE, and FD16RE are, respectively, 4-, 8-, and 16-bit data registers. When the clock enable (CE) input is High, and the synchronous reset (R) input is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q0) during the Low-to-High clock (C) transition. When R is High, it overrides all other inputs and resets the data outputs (Q) Low on the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

The flip-flops are asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

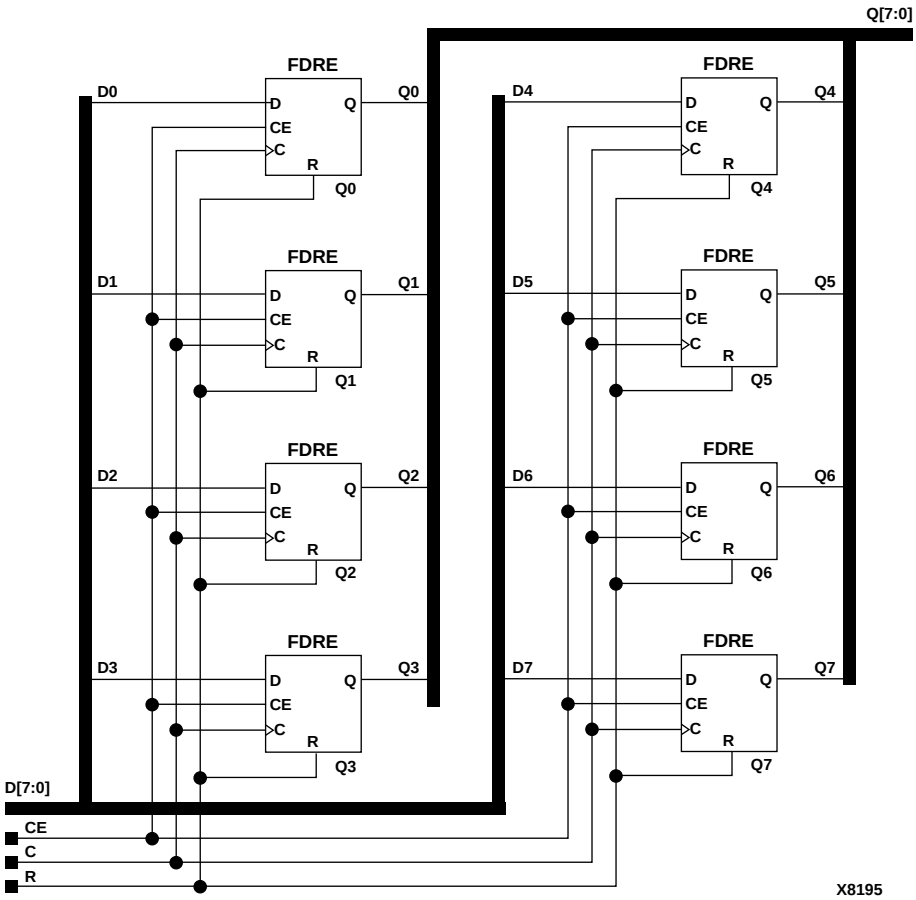
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	CE	Dz – D0	C	Qz – Q0
1	X	X	↑	0
0	0	X	X	No Chg
0	1	Dn	↑	dn

z = 3 for FD4RE; z = 7 for FD8RE; z = 15 for FD16RE

dn = state of referenced input (Dn) one setup time prior to active clock transition



FD8RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fd4re is
begin
 process (C) begin
  if (C’ event and C = ’1’) then
   if (R = ’1’) then
    Q <= "0000";
   elsif (CE = ’1’) then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

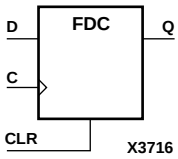
Verilog Inference Code

```
always @ (posedge C)
 begin
  if (R)
   Q<=4'b0000;
  else
   begin
    if (CE)
     Q<=D;
   end
 end
```

FDC

D Flip-Flop with Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FDC is a single D-type flip-flop with data (D) and asynchronous clear (CLR) inputs and data output (Q). The asynchronous CLR, when High, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop when CLR is Low on the Low-to-High clock transition.

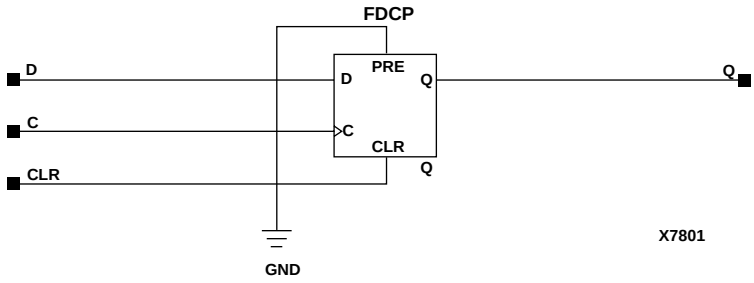
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	D	C	Q
1	X	X	0
0	1	↑	1
0	0	↑	0



FDC Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdc is
begin
 process (C, CLR) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (C' event and C = '1') then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or posedge CLR) begin
      if (CLR)
        Q <= 0;
      else
        Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDC should be placed
-- after architecture statement but before begin keyword

component FDC
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDC
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDC_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDC should be placed
-- in architecture after the begin keyword
```

```
FDC_INSTANCE_NAME : FDC
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CLR=> user_CLR,
                 D  => user_D);
```

Verilog Instantiation Template

```
FDC FDC_instance_name (.Q (user_Q),
                       .C (user_C),
                       .CLR (user_CLR),
                       .D (user_D));

defparam FDC_instance_name.INIT = bit_value;
```

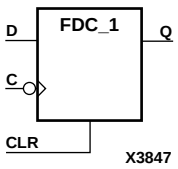
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDC_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDC_1 is a single D-type flip-flop with data input (D), asynchronous clear input (CLR), and data output (Q). The asynchronous CLR, when active, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	D	C	Q
1	X	X	0
0	1	↓	1
0	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdc_1 is
begin
 process (C, CLR) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (C' event and C = '0') then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @(negedge C or posedge CLR) begin
   if (CLR)
      Q <= 0;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDC_1 should be placed
-- after architecture statement but before begin keyword

component FDC_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDC_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDC_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDC_1 should be placed
-- in architecture after the begin keyword

FDC_1_INSTANCE_NAME : FDC_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CLR=> user_CLR,
                 D  => user_D);
```

Verilog Instantiation Template

```
FDC_1 FDC_1_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CLR (user_CLR),
                           .D (user_D));

defparam FDC_1_instance_name.INIT = bit_value;
```

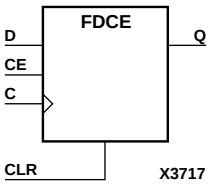
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCE

D Flip-Flop with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive



FDCE is a single D-type flip-flop with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data input (D) of FDCE is transferred to the corresponding data output (Q) during the Low-to-High clock (C) transition. When CLR is High, it overrides all other inputs and resets the data output (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For XC9500XL and XC9500XV devices, logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDCE and FDPE flip-flops primitives may take advantage of the clock-enable p-term.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdce is
begin
 process (C, CLR) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (C' event and C = '1') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge CLR or posedge C) begin
   if (CLR)
      Q <= 0;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDCE should be placed
-- after architecture statement but before begin keyword

component FDCE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDCE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDCE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDCE should be placed
-- in architecture after the begin keyword

FDCE_INSTANCE_NAME : FDCE
   -- synthesis translate_off
```

```
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 CLR=> user_CLR,
                 D  => user_D);
```

Verilog Instantiation Template

```
FDCE FDCE_instance_name (.Q (user_Q),
                         .C (user_C),
                         .CE (user_CE),
                         .CLR (user_CLR),
                         .D (user_D));

defparam FDCE_instance_name.INIT = bit_value;
```

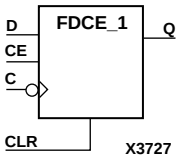
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDCE_1 is a single D-type flip-flop with data (D), clock enable (CE), asynchronous clear (CLR) inputs, and data output (Q). The asynchronous CLR input, when High, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop when CLR is Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	↓	No Chg
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdce_1 is
begin
 process (C, CLR) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (C' event and C = '0') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge CLR or negedge C) begin
   if (CLR)
      Q <= 0;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDCE_1 should be placed
-- after architecture statement but before begin keyword

component FDCE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDCE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDCE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDCE_1 should be placed
-- in architecture after the begin keyword

FDCE_1_INSTANCE_NAME : FDCE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 CLR=> user_CLR,
                 D  => user_D);
```

Verilog Instantiation Template

```
FDCE_1 FDCE_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .CLR (user_CLR),
                             .D (user_D));

defparam FDCE_1_instance_name.INIT = bit_value;
```

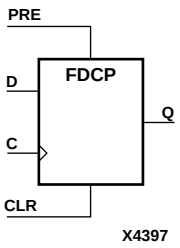
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCP

D Flip-Flop Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive



FDCP is a single D-type flip-flop with data (D), asynchronous preset (PRE) and clear (CLR) inputs, and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low on the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	PRE	D	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	↑	0
0	0	1	↑	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdcp is
begin
 process (C, CLR, PRE) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (PRE = '1') then
   Q <= '1';
  elsif (C' event and C = '1') then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge CLR or posedge PRE or posedge C) begin
   if (CLR)
      Q <= 0;
   else if (PRE)
      Q <= 1;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDCP should be placed
-- after architecture statement but before begin keyword

component FDCP
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDCP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDCP_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDCP should be placed
-- in architecture after the begin keyword

FDCP_INSTANCE_NAME : FDCP
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CLR=> user_CLR,
                 D  => user_D,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
FDCP FDCP_instance_name (.Q (user_Q),
                         .C (user_C),
                         .CLR (user_CLR),
                         .D (user_D),
                         .PRE (user_PRE));

defparam FDCP_instance_name.INIT = bit_value;
```

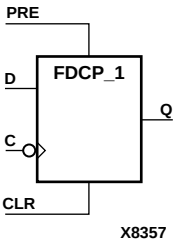
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCP_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDCP_1 is a single D-type flip-flop with data (D), asynchronous preset (PRE) and clear (CLR) inputs, and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low on the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	PRE	D	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	↓	0
0	0	1	↓	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdcp_1 is
begin
 process (C, CLR, PRE) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (PRE = '1') then
   Q <= '1';
  elsif (C' event and C = '0') then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge CLR or posedge PRE or negedge C) begin
   if (CLR)
      Q <= 0;
   else if (PRE)
      Q <= 1;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDCP_1 should be placed
-- after architecture statement but before begin keyword

component FDCP_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDCP_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDCP_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDCP_1 should be placed
-- in architecture after the begin keyword

FDCP_1_INSTANCE_NAME : FDCP_1
   -- synthesis translate_off
   generic map (
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CLR=> user_CLR,
                 D  => user_D,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
FDCP_1 FDCP_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CLR (user_CLR),
                             .D (user_D),
                             .PRE (user_PRE));

defparam FDCP_1_instance_name.INIT = bit_value;
```

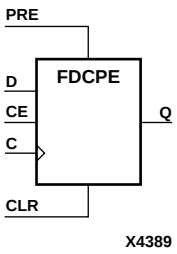
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCPE

D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Primitive



FDCPE is a single D-type flip-flop with data (D), clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High clock (C) transition. When CE is Low, the clock transitions are ignored.

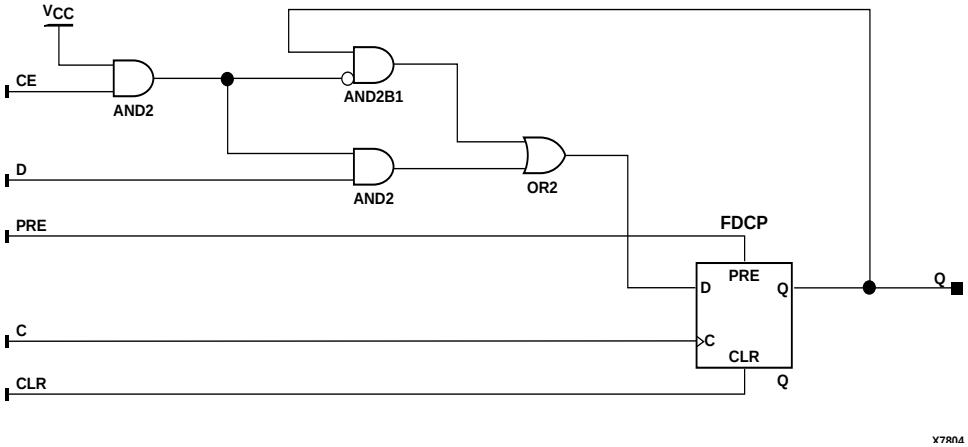
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	↑	0
0	0	1	1	↑	1



FDCPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdcpe is
begin
 process (C, CLR, PRE) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (PRE = '1') then
   Q <= '1';
  elsif (C' event and C = '1') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge CLR or posedge PRE or posedge C) begin
   if (CLR)
      Q <= 0;
   else if (PRE)
      Q <= 1;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDCPE should be placed
-- after architecture statement but before begin keyword

component FDCPE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDCPE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDCPE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDCPE should be placed
-- in architecture after the begin keyword

FDCPE_INSTANCE_NAME : FDCPE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 CLR=> user_CLR,
                 D  => user_D,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
FDCPE FDCPE_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CE (user_CE),
                           .CLR (user_CLR),
                           .D (user_D),
                           .PRE (user_PRE));

defparam FDCPE_instance_name.INIT = bit_value;
```

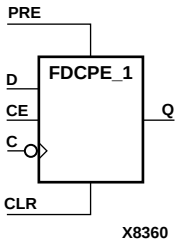
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCPE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDCPE_1 is a single D-type flip-flop with data (D), clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	↓	0
0	0	1	1	↓	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdcpe_1 is
begin
 process (C, CLR, PRE) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (PRE = '1') then
   Q <= '1';
  elsif (C' event and C = '0') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge CLR or posedge PRE or negedge C) begin
   if (CLR)
      Q <= 0;
   else if (PRE)
      Q <= 1;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDCPE_1 should be placed
-- after architecture statement but before begin keyword

component FDCPE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDCPE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDCPE_1_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDCPE_1 should be placed
-- in architecture after the begin keyword

FDCPE_1_INSTANCE_NAME : FDCPE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 CLR=> user_CLR,
                 D  => user_D,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
FDCPE_1 FDCPE_1_instance_name (.Q (user_Q),
                               .C (user_C),
                               .CE (user_CE),
                               .CLR (user_CLR),
                               .D (user_D),
                               .PRE (user_PRE));

defparam FDCPE_1_instance_name.INIT = bit_value;
```

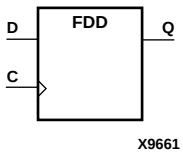
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDD

Dual Edge Triggered D Flip-Flop

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro

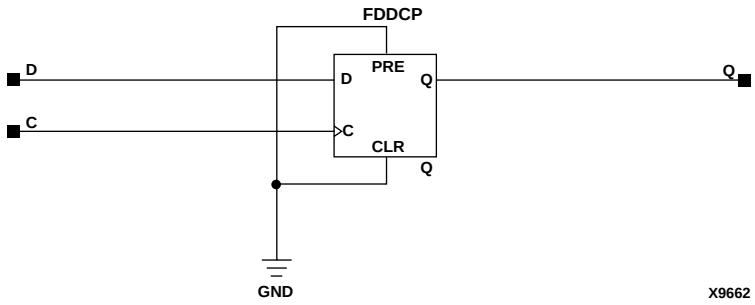


FDD is a single dual edge triggered D-type flip-flop with data input (D) and data output (Q). The data on the D input is loaded into the flip-flop during the Low-to-High and the High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

See the FDD4,8,16 section for information on multiple D flip-flops for the XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1
0	↓	0
1	↓	1



FDD Implementation CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdd is
begin
 process (C)
 begin
  if C'event then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or negedge C) begin
   Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDD should be placed
-- after architecture statement but before begin keyword

component FDD
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDD
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDD_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDD should be placed
-- in architecture after the begin keyword

FDD_INSTANCE_NAME : FDD
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 D => user_D);
```

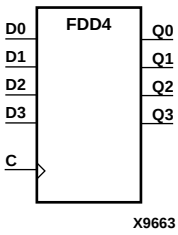
Verilog Instantiation Template

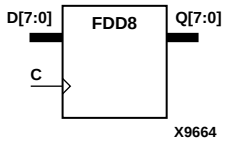
```
FDD FDD_instance_name(.Q (user_Q),
                      .C (user_C),
                      .D (user_D));

defparam FDD_instance_name.INIT = bit_value;
```

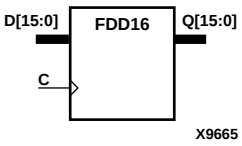
FDD4,8,16

Multiple Dual Edge Triggered D Flip-Flops

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro





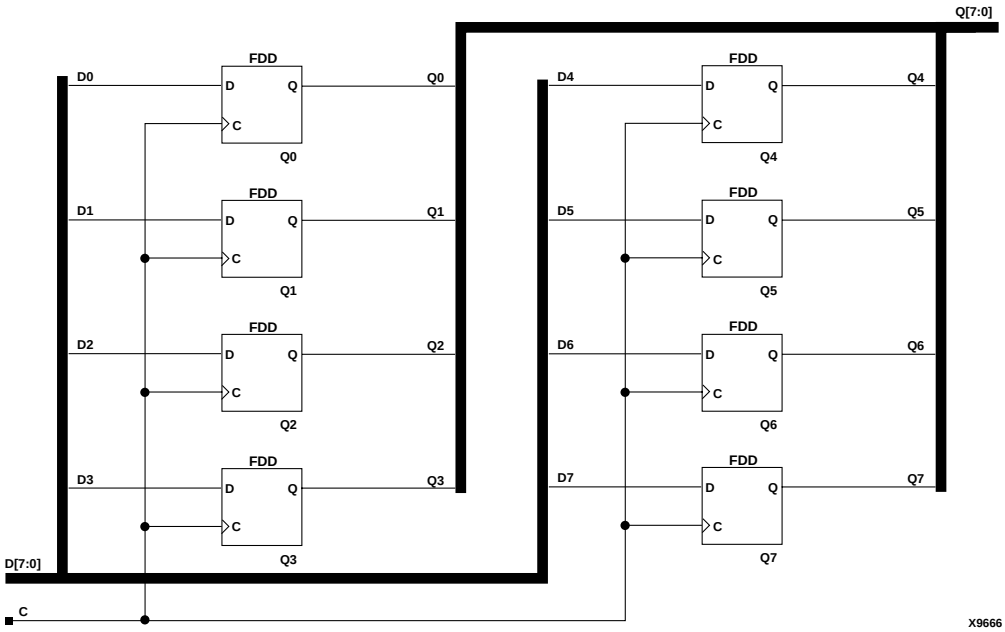


FDD4, FDD8, FDD16 are multiple dual edge triggered D-type flip-flops with data inputs (D) and data outputs (Q). FDD4, FDD8, and FDD16 are, respectively, 4-bit, 8-bit, and 16-bit registers, each with a common clock (C). The data on the D inputs is loaded into the flip-flop during the Low-to-High and High-to-Low clock (C) transitions.

The flip-flops are asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs		Outputs
Dz – D0	C	Qz – Q0
0	↑	0
1	↑	1
0	↓	0
1	↓	1

z = 3 for FDD4; z = 7 for FDD8; z = 15 for FDD16



FDD8 Implementation CoolRunner-II

Usage

For HDL, these deign elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fdd4 is
begin
 process (C)
 begin
  if C’event then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

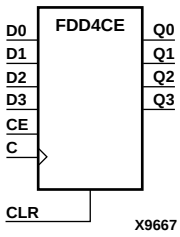
Verilog Inference Code

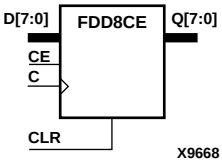
```
always @ (posedge C or negedge C) begin
   Q[3:0] <= D[3:0] ;
end
```

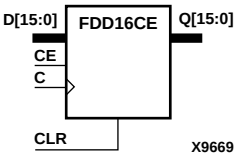
FDD4CE, FDD8CE, FDD16CE

4-, 8-, 16-Bit Dual Edge Triggered Data Registers with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







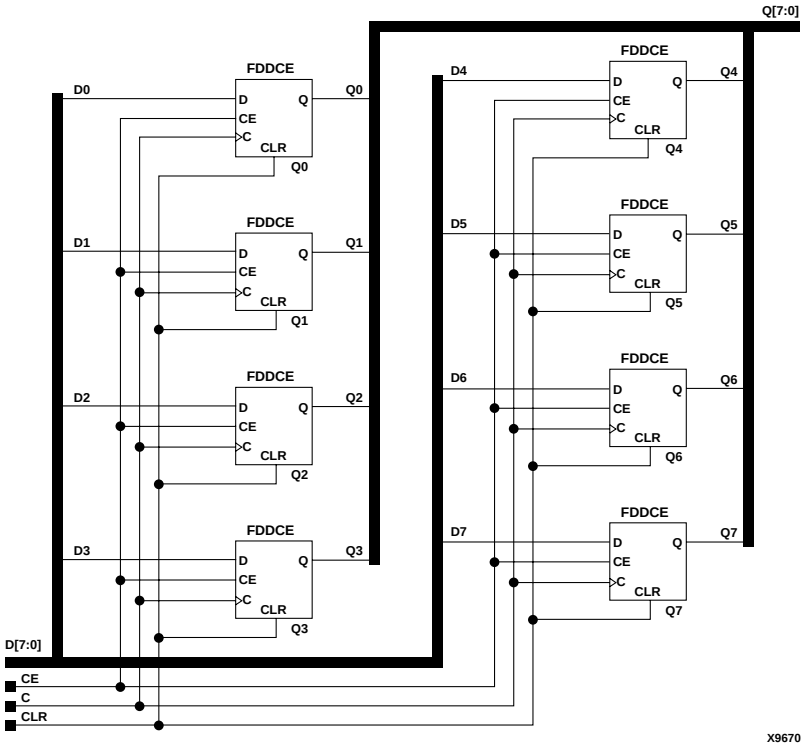
FDD4CE, FDD8CE, and FDD16CE are, respectively, 4-, 8-, and 16-bit data registers with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q) during the Low-to-High and High-to-Low clock (C) transitions. When CLR is High, it overrides all other inputs and resets the data outputs (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flops are asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	CE	Dz – D0	C	Qz – Q0
1	X	X	X	0
0	0	X	X	No Chg
0	1	Dn	↑	dn
0	1	Dn	↓	dn

z = 3 for FDD4CE; z = 7 for FDD8CE; z = 15 for FDD16CE.

dn = state of corresponding input (Dn) one setup time prior to active clock transitions



FDD8CE Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fdd4ce is
begin
 process (C, CLR) begin
  if (CLR = '1') then
   Q <= "0000";
  elsif (C' event) then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

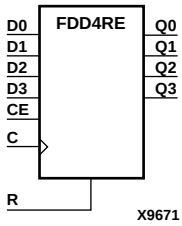
Verilog Inference Code

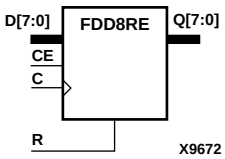
```
always @(posedge C or negedge C or posedge CLR)
begin
   if (CLR)
      Q <= 4'b0;
   else if (CE)
      Q[3:0] <= D[3:0] ;
end
```

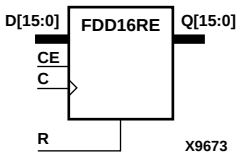
FDD4RE, FDD8RE, FDD16RE

4-, 8-, 16-Bit Dual Edge Triggered Data Registers with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







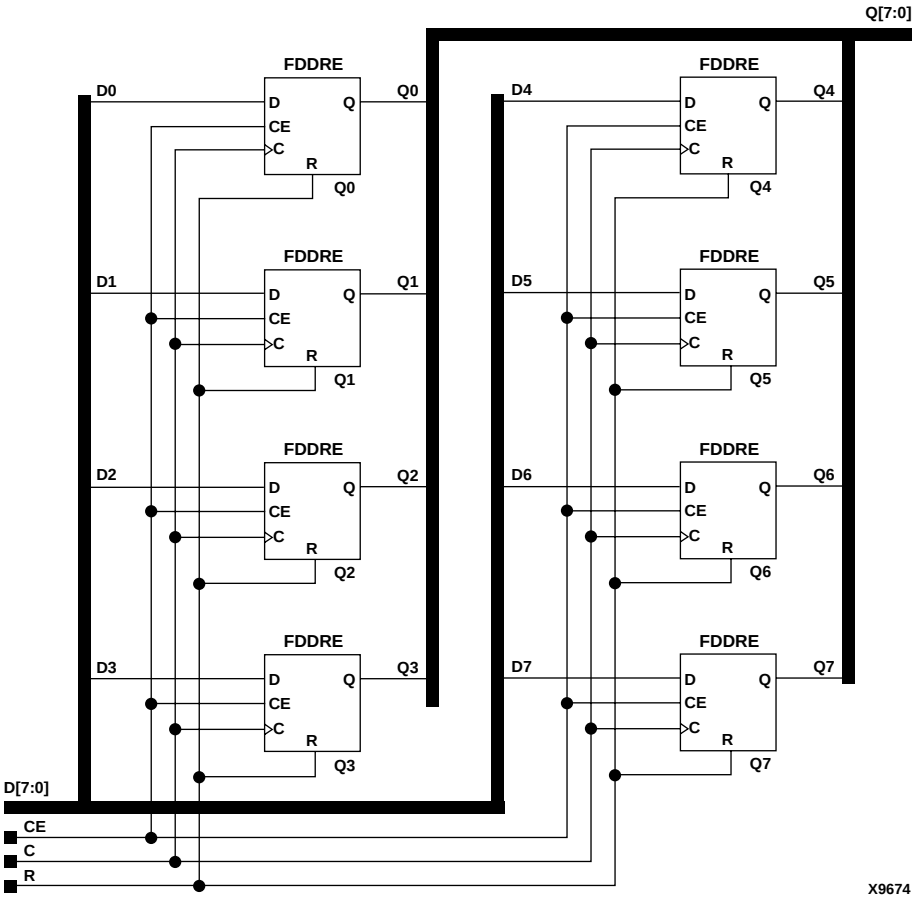
FDD4RE, FDD8RE, and FDD16RE are, respectively, 4-, 8-, and 16-bit data registers. When the clock enable (CE) input is High, and the synchronous reset (R) input is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q0) during the Low-to-High or High-to-Low clock (C) transition. When R is High, it overrides all other inputs and resets the data outputs (Q) Low on the Low-to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flops are asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
R	CE	Dz – D0	C	Qz – Q0
1	X	X	↑	0
1	X	X	↓	0
0	0	X	X	No Chg
0	1	Dn	↑	dn
0	1	Dn	↓	dn

z = 3 for FDD4RE; z = 7 for FDD8RE; z = 15 for FDD16RE

dn = state of referenced input (Dn) one setup time prior to active clock transitions



FDD8RE Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fdd4re is
begin
 process (C) begin
  if (C’ event) then
   if (CE = ’1’) then
    if (R = ’1’) then
     Q <= "0000";
    else
     Q <= D;
    end if;
   end if;
  end if;
 end process;
end Behavioral;
```

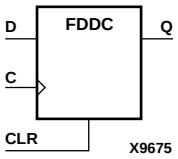
Verilog Inference Code

```
always @(posedge C or negedge C) begin
   if (R)
      Q <= 4'b0;
   else if (CE)
      Q[3:0] <= D[3:0] ;

end
```

FDDC

D Dual Edge Triggered Flip-Flop with Asynchronous Clear

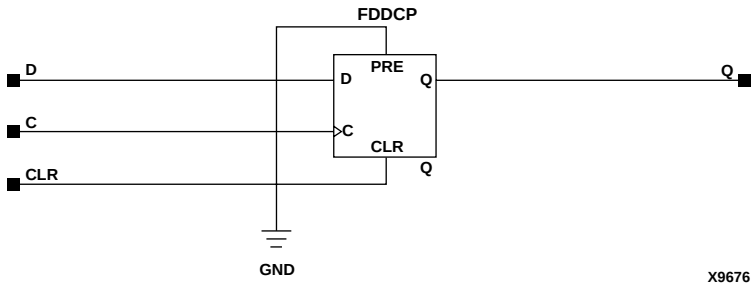
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDC is a single dual edge triggered D-type flip-flop with data (D) and asynchronous clear (CLR) inputs and data output (Q). The asynchronous CLR, when High, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop when CLR is Low on the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
CLR	D	C	Q
1	X	X	0
0	1	↑	1
0	1	↓	1
0	0	↑	0
0	0	↓	0



FDDC Implementation CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fddc is
begin
 process (C,CLR) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (C' event) then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or negedge C or posedge CLR)
begin
   if (CLR)
      Q <= 1'b0;
   else
      Q <= D ;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDDC should be placed
-- after architecture statement but before begin keyword

component FDDC
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDC
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDC_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDDC should be placed
-- in architecture after the begin keyword

FDDC_INSTANCE_NAME : FDDC
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
```

```
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CLR=> user_CLR,
                 D  => user_D);
```

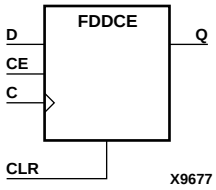
Verilog Instantiation Template

```
FDDC FDDC_instance_name(.Q (user_Q),
                        .C (user_C),
                        .CLR (user_CLR),
                        .D (user_D));

defparam FDDC_instance_name.INIT = bit_value;
```

FDDCE

Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Primitive



FDDCE is a single dual edge triggered D-type flip-flop with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data input (D) of FDDCE is transferred to the corresponding data output (Q) during the Low-to-High and High-to-Low clock (C) transitions. When CLR is High, it overrides all other inputs and resets the data output (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDDCE and FDDPE flip-flops primitives may take advantage of the clock-enable p-term.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fddce is
begin
 process (C, CLR) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (C' event) then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or negedge C or posedge CLR)
begin
   if (CLR)
      Q <= 1'b0;
   else
   if (CE)
      Q <= D ;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDDCE should be placed
-- after architecture statement but before begin keyword

component FDDCE
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDCE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for FDDCE should be placed
-- in architecture after the begin keyword

FDDCE_INSTANCE_NAME : FDDCE
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 CLR=> user_CLR,
                 D  => user_D);
```

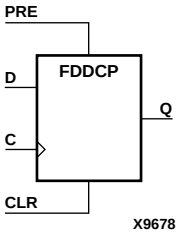
Verilog Instantiation Template

```
FDDCE FDDCE_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CE (user_CE),
                           .CLR (user_CLR),
                           .D (user_D));
```

FDDCP

Dual Edge Triggered D Flip-Flop Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Primitive



FDDCP is a single dual edge triggered D-type flip-flop with data (D), asynchronous preset (PRE) and clear (CLR) inputs, and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low on the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	PRE	D	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	↑	0
0	0	1	↑	1
0	0	0	↓	0
0	0	1	↓	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fddcp is
begin
 process (C, CLR, PRE) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (PRE = '1') then
   Q <= '1';
  elsif (C' event) then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or negedge C or posedge CLR or posedge PRE)
begin
   if (CLR)
      Q <= 1'b0;
   else
   if (PRE)
      Q <= 1'b1;
   else
      Q <= D ;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDDCP should be placed
-- after architecture statement but before begin keyword

component FDDCP
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDCP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDCP_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDDCP should be placed
-- in architecture after the begin keyword

FDDCP_INSTANCE_NAME : FDDCP
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CLR=> user_CLR,
                 D  => user_D,
                 PRE => user_PRE);
```

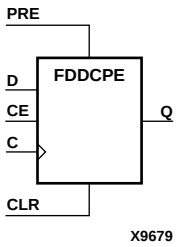
Verilog Instantiation Template

```
FDDCP FDDCP_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CLR (user_CLR),
                           .D (user_D),
                           .PRE (user_PRE));

defparam FDDCP_instance_name.INIT = bit_value;
```

FDDCPE

Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

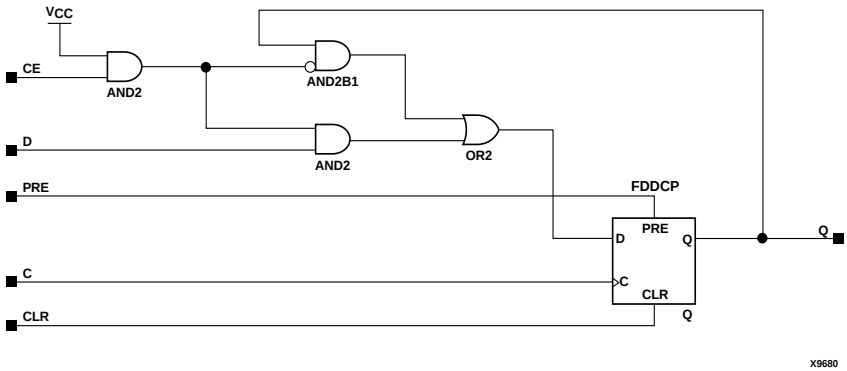
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDCPE is a single dual edge triggered D-type flip-flop with data (D), clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High and High-to-Low clock (C) transitions. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	↑	0
0	0	1	1	↑	1
0	0	1	0	↓	0
0	0	1	1	↓	1



FDDCPE Implementation CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fddcpe is
begin
 process (C, CLR, PRE) begin
  if (CLR = '1') then
   Q <= '0';
  elsif (PRE = '1') then
   Q <= '1';
  elsif (C' event) then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or negedge C or posedge CLR or posedge PRE)
begin
   if (CLR)
      Q <= 1'b0;
   else
   if (PRE)
      Q <= 1'b1;
   else
   if (CE)
      Q <= D ;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDDCPE should be placed
-- after architecture statement but before begin keyword

component FDDCPE
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDCPE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Enter values here


-- Component Instantiation for FDDCPE should be placed
-- in architecture after the begin keyword

FDDCPE_INSTANCE_NAME : FDDCPE
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 CLR=> user_CLR,
                 D  => user_D,
                 PRE => user_PRE);
```

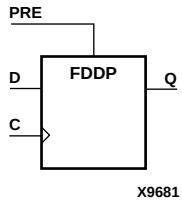
Verilog Instantiation Template

```
FDDCPE FDDCPE_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .CLR (user_CLR),
                             .D (user_D),
                             .PRE (user_PRE));
```

FDDP

Dual Edge Triggered D Flip-Flop with Asynchronous Preset

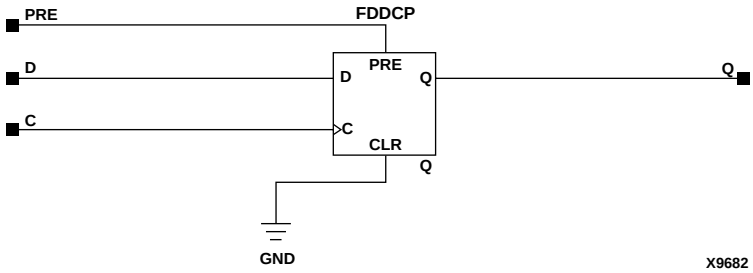
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDP is a single dual edge triggered D-type flip-flop with data (D) and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and presets the Q output High. The data on the D input is loaded into the flip-flop when PRE is Low on the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
PRE	C	D	Q
1	X	X	1
0	↑	1	1
0	↑	0	0
0	↓	1	1
0	↓	0	0



FDDP Implementation CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of fddp is
begin
 process (C,PRE) begin
  if (PRE = '1') then
   Q <= '1';
  elsif (C' event) then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or negedge C or posedge PRE)
begin
   if (PRE)
      Q <= 1'b1;
   else
      Q <= D ;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDDP should be placed
-- after architecture statement but before begin keyword

component FDDP
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDP_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDDP should be placed
-- in architecture after the begin keyword

FDDP_INSTANCE_NAME : FDDP
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
```

```
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 D => user_D,
                 PRE => user_PRE);
```

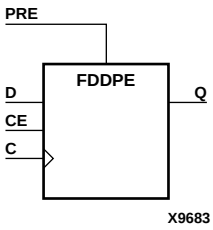
Verilog Instantiation Template

```
FDDP FDDP_instance_name(.Q (user_Q),
                        .C (user_C),
                        .D (user_D),
                        .PRE (user_PRE));

defparam FDDP_instance_name.INIT = bit_value;
```

FDDPE

Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Primitive



FDDPE is a single dual edge triggered D-type flip-flop with data (D), clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and sets the Q output High. Data on the D input is loaded into the flip-flop when PRE is Low and CE is High on the Low-to-High and High-to-Low clock (C) transitions. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDDCE and FDDPE flip-flops primitives may take advantage of the clock-enable p-term.

Inputs				Outputs
PRE	CE	D	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	↑	0
0	1	1	↑	1
0	1	0	↓	0
0	1	1	↓	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fddpe is
begin
 process (C,PRE) begin
  if (PRE = '1') then
   Q <= '1';
  elsif (C' event) then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or negedge C or posedge PRE)
begin
   if (PRE)
      Q <= 1'b1;
   else
   if (CE)
      Q <= D ;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDDPE should be placed
-- after architecture statement but before begin keyword

component FDDPE
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDPE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for FDDPE should be placed
-- in architecture after the begin keyword

FDDPE_INSTANCE_NAME : FDDPE
      port map  (Q => user_Q,
                 C => user_C,
                 CE => user_CE,
                 D => user_D,
                 PRE => user_PRE);
```

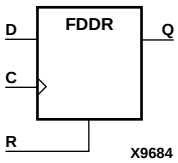
Verilog Instantiation Template

```
FDDPE FDDPE_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CE (user_CE),
                           .D (user_D),
                           .PRE (user_PRE));

defparam FDDPE_instance_name.INIT = bit_value;
```

FDDR

Dual Edge Triggered D Flip-Flop with Synchronous Reset

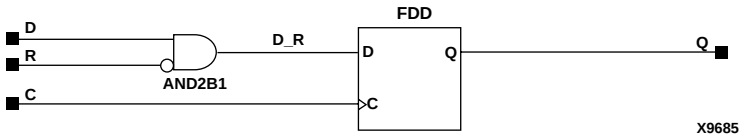
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDR is a single dual edge triggered D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High and High-to-Low clock (C) transitions. The data on the D input is loaded into the flip-flop when R is Low during the Low-to-High or High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
R	D	C	Q
1	X	↑	0
1	X	↓	0
0	1	↑	1
0	0	↑	0
0	1	↓	1
0	0	↓	0



FDDR Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fddr is
begin
 process (C) begin
  if (C’ event) then
   if (R = ’1’) then
    Q <= ’0’;
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

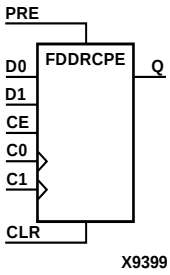
Verilog Inference Code

```
always @ (posedge C or negedge C)
begin
   if (R)
      Q <= 1’b0;
   else
      Q <= D ;
end
```

FDDRCPE

Dual Data Rate D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



FDDRCPE is a Dual Data Rate (DDR) D flip-flop with two separate clocks (C0 and C1) phase shifted 180 degrees that allow selection of two separate data inputs (D0 and D1). It also has clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C1 clock transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Use the INIT attribute to initialize FDDRCPE during configuration.

Inputs							Outputs
C0	C1	CE	D0	D1	CLR	PRE	Q
X	X	X	X	X	1	0	0
X	X	X	X	X	0	1	1
X	X	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for FDDRCPE should be placed
-- after architecture statement but before begin keyword

component FDDRCPE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C0   : in STD_ULOGIC;
         C1   : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D0   : in STD_ULOGIC;
         D1   : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDRCPE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDRCPE_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDDRCPE should be placed
-- in architecture after the begin keyword

FDDRCPE_INSTANCE_NAME : FDDRCPE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C0 => user_C0,
                 C1 => user_C1,
                 CE => user_CE,
                 CLR=> user_CLR,
                 D0 => user_D0,
                 D1 => user_D1,
                 PRE => user_PRE);
```

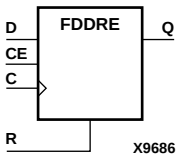
Verilog Instantiation Template

```
FDDRCPE FDDRCPE_instance_name (.Q (user_Q),
                               .C0 (user_C0),
                               .C1 (user_C1),
                               .CE (user_CE),
                               .CLR (user_CLR),
                               .D0 (user_D0),
                               .D1 (user_D1),
                               .PRE (user_PRE));

defparam FDDRCPE_instance_name.INIT = bit_value;
```

FDDRE

Dual Edge Triggered D Flip-Flop with Clock Enable and Synchronous Reset

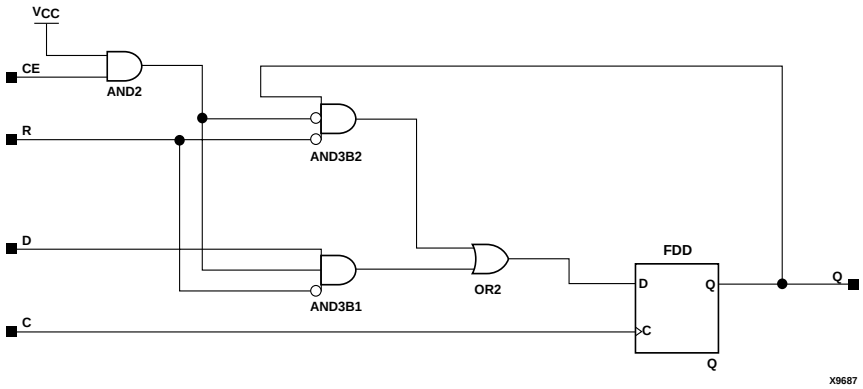
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDRE is a single dual edge triggered D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High or High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low and CE is High during the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
R	CE	D	C	Q
1	X	X	↑	0
1	X	X	↓	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0
0	1	1	↓	1
0	1	0	↓	0



FDDRE Implementation CoolRunner-II

Usage

For HDL, this design element can be inferred but not instantiated.

VHDL Inference Code

```
architecture Behavioral of fddre is
begin
    process (C) begin
        if (C' event) then
            if (R = '1') then
                Q <= '0';
            elsif (CE = '1') then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

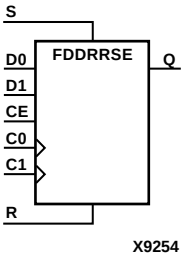
Verilog Inference Code

```
always @ (posedge C or negedge C)
begin
   if (R)
      Q <= 1'b0;
   else
   if (CE)
      Q <= D ;
end
```

FDDRRSE

Dual Data Rate D Flip-Flop with Clock Enable and Synchronous Reset and Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



FDDRRSE is a Dual Data Rate (DDR) D flip-flop with two separate clocks (C0 and C1) phase shifted 180 degrees that allow selection of two separate data inputs (D0 and D1). It also has synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during any Low-to-High clock transition (C0 or C1). (Reset has precedence over Set.) When the S input is High and R is Low, the flip-flop is set, output High, during a Low-to-High clock transition (C0 or C1). Data on the D0 input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High C1 clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Use the INIT attribute to initialize FDDRRSE during configuration.

Inputs							Outputs
C0	C1	CE	D0	D1	R	S	Q
↑	X	X	X	X	1	0	0
↑	X	X	X	X	0	1	1
↑	X	X	X	X	1	1	0
X	↑	X	X	X	1	0	0
X	↑	X	X	X	0	1	1
X	↑	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for FDDRRSE should be placed
-- after architecture statement but before begin keyword

component FDDRRSE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C0   : in STD_ULOGIC;
         C1   : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D0   : in STD_ULOGIC;
         D1   : in STD_ULOGIC;
         R    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDRRSE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDRRSE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDDRRSE should be placed
-- in architecture after the begin keyword

FDDRRSE_INSTANCE_NAME : FDDRRSE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C0 => user_C0,
                 C1 => user_C1,
                 CE => user_CE,
                 D0 => user_D0,
                 D1 => user_D1,
                 R  => user_R,
                 S  => user_S);
```

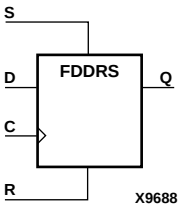
Verilog Instantiation Template

```
FDDRRSE FDDRRSE_instance_name (.Q (user_Q),
                               .C0 (user_C0),
                               .C1 (user_C1),
                               .CE (user_CE),
                               .D0 (user_D0),
                               .D1 (user_D1),
                               .R (user_R),
                               .S (user_S));

defparam FDDRRSE_instance_name.INIT = bit_value;
```

FDDRS

Dual Edge Triggered D Flip-Flop with Synchronous Reset and Set

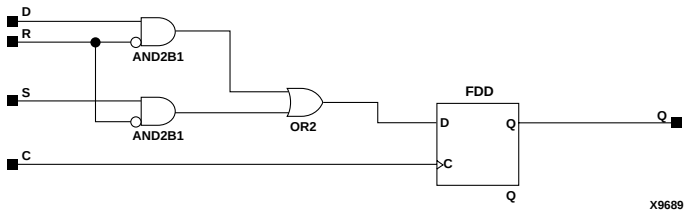
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDRS is a single dual edge triggered D-type flip-flop with data (D), synchronous set (S), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High or High-to-Low clock (C) transitions. (Reset has precedence over Set.) When S is High and R is Low, the flip-flop is set, output High, during the Low-to-High or High-to-Low clock transition. When R and S are Low, data on the (D) input is loaded into the flip-flop during the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
R	S	D	C	Q
1	X	X	↑	0
1	X	X	↓	0
0	1	X	↑	1
0	1	X	↓	1
0	0	1	↑	1
0	0	1	↓	1
0	0	0	↑	0
0	0	0	↓	0



FDDRS Implementation CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fddrs is
begin
 process (C) begin
  if (C’ event) then
   if (R = ’1’) then
      Q <= ’0’;
   elsif (S = ’1’) then
    Q <= ’1’;
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @(posedge C or negedge C)
begin
   if (R)
      Q <= 1’b0;
   else
   if (S)
      Q <= 1’b1;
   else
      Q <= D ;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDDRS should be placed
-- after architecture statement but before begin keyword

component FDDRS
   -- synthesis translate_off
   generic (
        INIT  : bit := ’1’);
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDRS
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDRS_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDDRS should be placed
-- in architecture after the begin keyword

FDDRS_INSTANCE_NAME : FDDRS
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 D  => user_D,
                 R  => user_R,
                 S  => user_S);
```

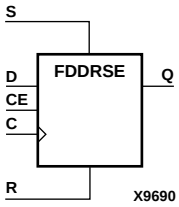
Verilog Instantiation Template

```
FDDRS FDDRS_instance_name (.Q (user_Q),
                           .C (user_C),
                           .D (user_D),
                           .R (user_R),
                           .S (user_S));

defparam FDDRS_instance_name.INIT = bit_value;
```

FDDRSE

Dual Edge Triggered D Flip-Flop with Synchronous Reset and Set and Clock Enable

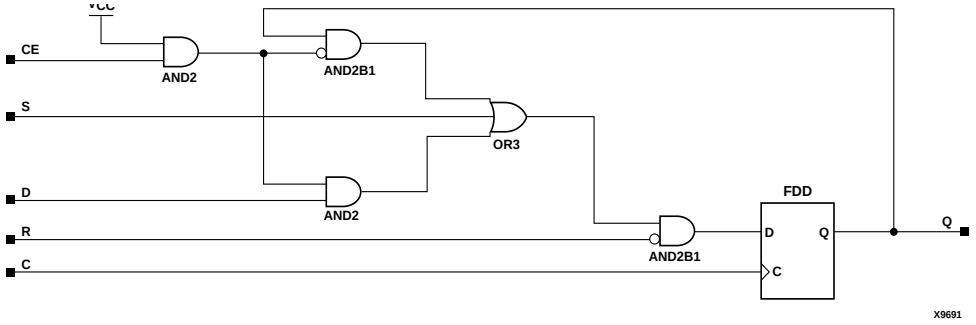
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDRSE is a single dual edge triggered D-type flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High or High-to-Low clock transitions. (Reset has precedence over Set.) When the set (S) input is High and R is Low, the flip-flop is set, output High, during the Low-to-High or High-to-Low clock (C) transition. Data on the D input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
R	S	CE	D	C	Q
1	X	X	X	↑	0
1	X	X	X	↓	0
0	1	X	X	↑	1
0	1	X	X	↓	1
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0
0	0	1	1	↓	1
0	0	1	0	↓	0



FDDRSE Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fddrse is
begin
    process (C) begin
        if (C’ event) then
            if (R = ’1’) then
                Q <= ’0’;
            elsif (S = ’1’) then
                Q <= ’1’;
            elsif (CE = ’1’) then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

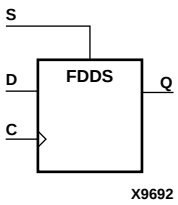
Verilog Inference Code

```
always @ (posedge C or negedge C)
begin
   if (R)
      Q <= 1'b0;
   else
   if (S)
      Q <= 1'b1;
   else
   if (CE)
   Q <= D ;
end
```

FDDS

Dual Edge Triggered D Flip-Flop with Synchronous Set

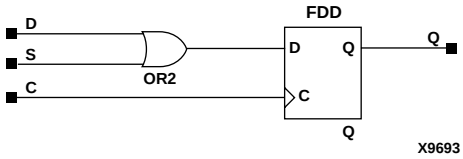
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDS is a single dual edge triggered D-type flip-flop with data (D) and synchronous set (S) inputs and data output (Q). The synchronous set input, when High, sets the Q output High on the Low-to-High or High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low during the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
S	D	C	Q
1	X	↑	1
1	X	↓	1
0	1	↑	1
0	0	↑	0
0	1	↓	1
0	0	↓	0



FDDS Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fdds is
begin
 process (C) begin
  if (C’ event) then
   if (S = ’1’) then
    Q <= ’1’;
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

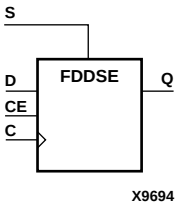
Verilog Inference Code

```
always @ (posedge C or negedge C)
begin
   if (S)
      Q <= 1’b1;
   else
      Q <= D ;
end
```

FDDSE

D Flip-Flop with Clock Enable and Synchronous Set

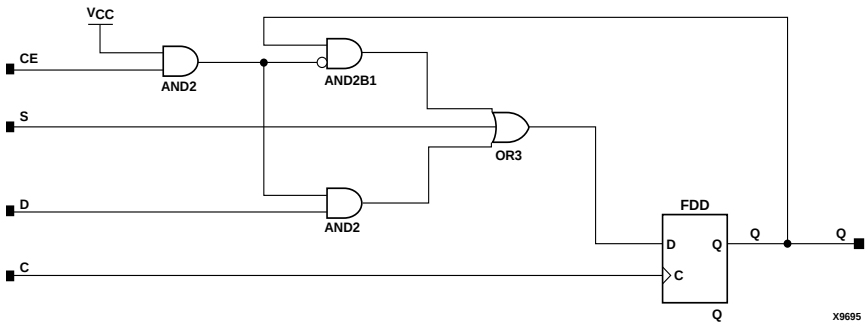
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDSE is a single dual edge triggered D-type flip-flop with data (D), clock enable (CE), and synchronous set (S) inputs and data output (Q). The synchronous set (S) input, when High, overrides the clock enable (CE) input and sets the Q output High during the Low-to-High or High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low and CE is High during the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↑	1
1	X	X	↓	1
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0
0	1	1	↓	1
0	1	0	↓	0



FDDSE Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fddse is
begin
    process (C) begin
        if (C’ event) then
            if (S = ’1’) then
                Q <= ’1’;
            elsif (CE = ’1’) then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

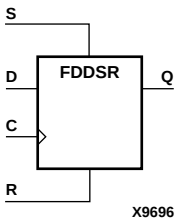
Verilog Inference Code

```
always @ (posedge C or negedge C)
begin
   if (S)
      Q <= 1'b1;
   else
   if (CE)
      Q <= D ;
end
```

FDDSR

Dual Edge Triggered D Flip-Flop with Synchronous Set and Reset

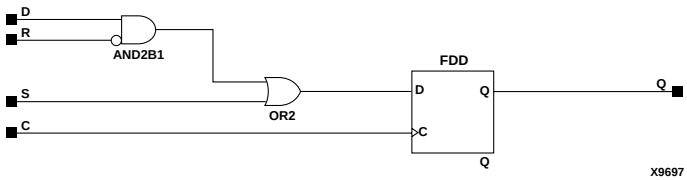
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDSR is a single dual edge triggered D-type flip-flop with data (D), synchronous reset (R) and synchronous set (S) inputs and data output (Q). When the set (S) input is High, it overrides all other inputs and sets the Q output High during the Low-to-High or High-to-Low clock transition. (Set has precedence over Reset.) When reset (R) is High and S is Low, the flip-flop is reset, output Low, on the Low-to-High or High-to-Low clock transition. Data on the D input is loaded into the flip-flop when S and R are Low on the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
S	R	D	C	Q
1	X	X	↑	1
1	X	X	↓	1
0	1	X	↑	0
0	1	X	↓	0
0	0	1	↑	1
0	0	0	↑	0
0	0	1	↓	1
0	0	0	↓	0



FDDSR Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fddsr is
begin
        process (C) begin
                if (C’ event) then
                        if (S = ’1’) then
                                Q <= ’1’;
                        elsif (R = ’1’) then
                              Q <= ’0’;
                        else
                                Q <= D;
                        end if;
                end if;
        end process;
end Behavioral;
```

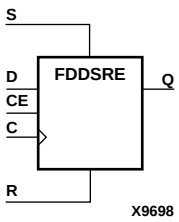
Verilog Inference Code

```
always @ (posedge C or negedge C)
begin
   if (S)
      Q <= 1’b1;
   else
   if (R)
      Q <= 1’b0;
   else
      Q <= D;
end
```

FDDSRE

Dual Edge Triggered D Flip-Flop with Synchronous Set and Reset and Clock Enable

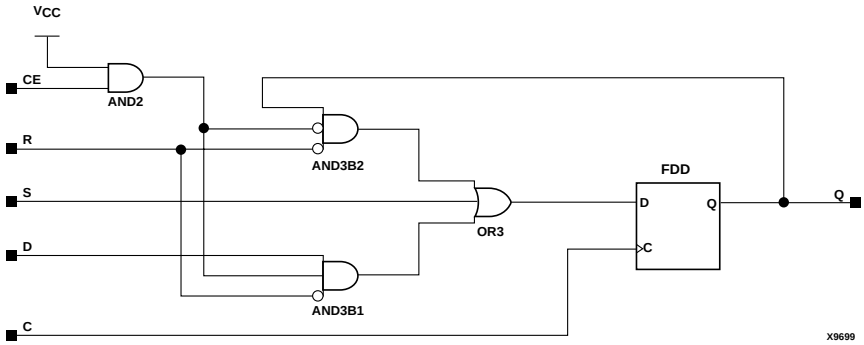
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FDDSRE is a single dual edge triggered D-type flip-flop with synchronous set (S), synchronous reset (R), and clock enable (CE) inputs and data output (Q). When synchronous set (S) is High, it overrides all other inputs and sets the Q output High during the Low-to-High or High-to-Low clock transition. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, output Q is reset Low during the Low-to-High or High-to-Low clock transition. Data is loaded into the flip-flop when S and R are Low and CE is High during the Low-to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
S	R	CE	D	C	Q
1	X	X	X	↑	1
1	X	X	X	↓	1
0	1	X	X	↑	0
0	1	X	X	↓	0
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0
0	0	1	1	↓	1
0	0	1	0	↓	0



FDDSRE Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fddsre is
begin
        process (C) begin
                if (C’ event) then
                        if (S = ’1’) then
                                Q <= ’1’;
                        elsif (R = ’1’) then
                              Q <= ’0’;
                        elsif (CE = ’1’) then
                                Q <= D;
                        end if;
                end if;
        end process;

end Behavioral;
```

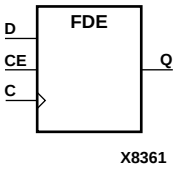
Verilog Inference Code

```
always @ (posedge C or negedge C)
begin
   if (S)
      Q <= 1’b1;
   else
   if (R)
      Q <= 1’b0;
   else
   if (CE)
      Q <= D;
end
```

FDE

D Flip-Flop with Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDE is a single D-type flip-flop with data input (D), clock enable (CE), and data output (Q). When clock enable is High, the data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
0	X	X	No Chg
1	0	↑	0
1	1	↑	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fde is
begin
 process (C)
 begin
  if (C'event and C='1') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C) begin
   if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDE should be placed
-- after architecture statement but before begin keyword

component FDE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDE should be placed
-- in architecture after the begin keyword

FDE_INSTANCE_NAME : FDE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 CE => user_CE,
                 D => user_D);
```

Verilog Instantiation Template

```
FDE FDE_instance_name (.Q (user_Q),
                       .C (user_C),
                       .CE (user_CE),
                       .D (user_D));

defparam FDE_instance_name.INIT = bit_value;
```

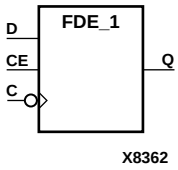
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDE_1

D Flip-Flop with Negative-Edge Clock and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDE_1 is a single D-type flip-flop with data input (D), clock enable (CE), and data output (Q). When clock enable is High, the data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
0	X	X	No Chg
1	0	↓	0
1	1	↓	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fde_1 is
begin
 process (C)
 begin
  if (C'event and C='0') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C) begin
   if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDE_1 should be placed
-- after architecture statement but before begin keyword

component FDE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDE_1 should be placed
-- in architecture after the begin keyword

FDE_1_INSTANCE_NAME : FDE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 CE => user_CE,
                 D => user_D);
```

Verilog Instantiation Template

```
FDE_1 FDE_1_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CE (user_CE),
                           .D (user_D));

defparam FDE_1_instance_name.INIT = bit_value;
```

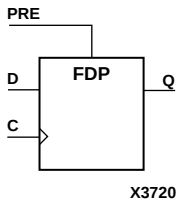
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDP

D Flip-Flop with Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FDP is a single D-type flip-flop with data (D) and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and presets the Q output High. The data on the D input is loaded into the flip-flop when PRE is Low on the Low-to-High clock (C) transition.

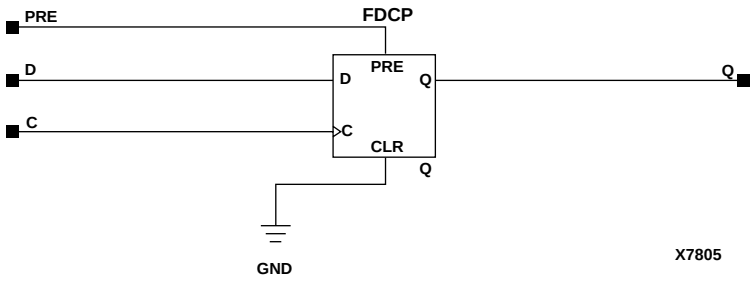
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

The active level of the GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	C	D	Q
1	X	X	1
0	↑	1	1
0	↑	0	0



FDP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be inferred for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdp is
begin
 process (C,PRE) begin
  if (PRE = '1') then
   Q <= '1';
  elsif (C' event and C = '1') then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge PRE or posedge C) begin
   if (PRE)
      Q <= 1;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDP should be placed
-- after architecture statement but before begin keyword

component FDP
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDP_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDP should be placed
-- in architecture after the begin keyword
```

```
FDP_INSTANCE_NAME : FDP
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 D => user_D,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
FDP FDP_instance_name (.Q (user_Q),
                       .C (user_C),
                       .D (user_D),
                       .PRE (user_PRE));

defparam FDP_instance_name.INIT = bit_value;
```

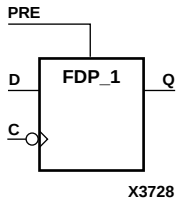
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDP_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDP_1 is a single D-type flip-flop with data (D) and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and presets the Q output High. The data on the D input is loaded into the flip-flop when PRE is Low on the High-to-Low clock (C) transition.

The flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

The active level of the GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	C	D	Q
1	X	X	1
0	↓	1	1
0	↓	0	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdp_1 is
begin
 process (C,PRE) begin
  if (PRE = '1') then
   Q <= '1';
  elsif (C' event and C = '0') then
   Q <= D;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge PRE or negedge C) begin
   if (PRE)
      Q <= 1;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDP_1 should be placed
-- after architecture statement but before begin keyword

component FDP_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDP_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDP_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDP_1 should be placed
-- in architecture after the begin keyword

FDP_1_INSTANCE_NAME : FDP_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 D => user_D,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
FDP_1 FDP_1_instance_name (.Q (user_Q),
                           .C (user_C),
                           .D (user_D),
                           .PRE (user_PRE));

defparam FDP_1_instance_name.INIT = bit_value;
```

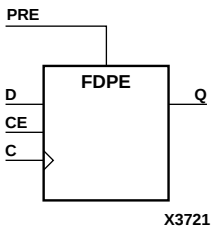
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDPE

D Flip-Flop with Clock Enable and Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive



FDPE is a single D-type flip-flop with data (D), clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and sets the Q output High. Data on the D input is loaded into the flip-flop when PRE is Low and CE is High on the Low-to-High clock (C) transition. When CE is Low, the clock transitions are ignored.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For XC9500XL and XC9500XV devices, logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDCE and FDPE flip-flop primitives may take advantage of the clock-enable p-term.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset, output High, when power is applied. These devices simulate power-on when global set/reset (GSR) is active.

The active level of the GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
PRE	CE	D	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	↑	0
0	1	1	↑	1

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdpe is
begin
 process (C,PRE) begin
  if (PRE = '1') then
   Q <= '1';
  elsif (C' event and C = '1') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge PRE or posedge C) begin
   if (PRE)
      Q <= 1;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDPE should be placed
-- after architecture statement but before begin keyword

component FDPE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDPE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDPE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDPE should be placed
-- in architecture after the begin keyword

FDPE_INSTANCE_NAME : FDPE
   -- synthesis translate_off
```

```
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 CE => user_CE,
                 D => user_D,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
FDPE FDPE_instance_name (.Q (user_Q),
                         .C (user_C),
                         .CE (user_CE),
                         .D (user_D),
                         .PRE (user_PRE));

defparam FDPE_instance_name.INIT = bit_value;
```

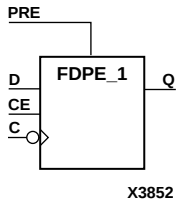
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDPE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDPE_1 is a single D-type flip-flop with data (D), clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and sets the Q output High. Data on the D input is loaded into the flip-flop when PRE is Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

The active level of the GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
PRE	CE	D	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdpe_1 is
begin
 process (C,PRE) begin
  if (PRE = '1') then
   Q <= '1';
  elsif (C' event and C = '0') then
   if (CE = '1') then
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge PRE or negedge C) begin
   if (PRE)
      Q <= 1;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDPE_1 should be placed
-- after architecture statement but before begin keyword

component FDPE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDPE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDPE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDPE_1 should be placed
-- in architecture after the begin keyword --

FDPE_1_INSTANCE_NAME : FDPE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 CE => user_CE,
                 D => user_D,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
FDPE_1 FDPE_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .D (user_D),
                             .PRE (user_PRE));

defparam FDPE_1_instance_name.INIT = bit_value;
```

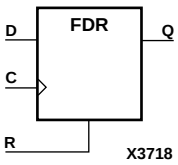
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDR

D Flip-Flop with Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FDR is a single D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
R	D	C	Q
1	X	↑	0
0	1	↑	1
0	0	↑	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdr is
begin
 process (C) begin
  if (C' event and C = '1') then
   if (R = '1') then
    Q <= '0';
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C) begin
   if (R)
      Q <= 0;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDR should be placed
-- after architecture statement but before begin keyword

component FDR
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDR
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDR_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDR should be placed
-- in architecture after the begin keyword

FDR_INSTANCE_NAME : FDR
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 D => user_D,
                 R => user_R);
```

Verilog Instantiation Template

```
FDR FDR_instance_name (.Q (user_Q),
                       .C (user_C),
                       .D (user_D),
                       .R (user_R));

defparam FDR_instance_name.INIT = bit_value;
```

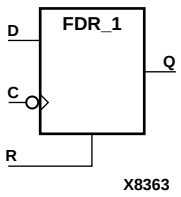
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDR_1

D Flip-Flop with Negative-Edge Clock and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDR_1 is a single D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low during the High-to-Low clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
R	D	C	Q
1	X	↓	0
0	1	↓	1
0	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdr_1 is
begin
 process (C) begin
  if (C' event and C = '0') then
   if (R = '1') then
    Q <= '0';
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C) begin
   if (R)
      Q <= 0;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDR_1 should be placed
-- after architecture statement but before begin keyword

component FDR_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDR_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDR_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDR_1 should be placed
-- in architecture after the begin keyword

FDR_1_INSTANCE_NAME : FDR_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 C => user_C,
                 D => user_D,
                 R => user_R);
```

Verilog Instantiation Template

```
FDR_1 FDR_1_instance_name (.Q (user_Q),
                           .C (user_C),
                           .D (user_D),
                           .R (user_R));

defparam FDR_1_instance_name.INIT = bit_value;
```

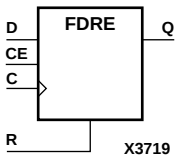
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRE

D Flip-Flop with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FDRE is a single D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low and CE is High during the Low-to-High clock transition.

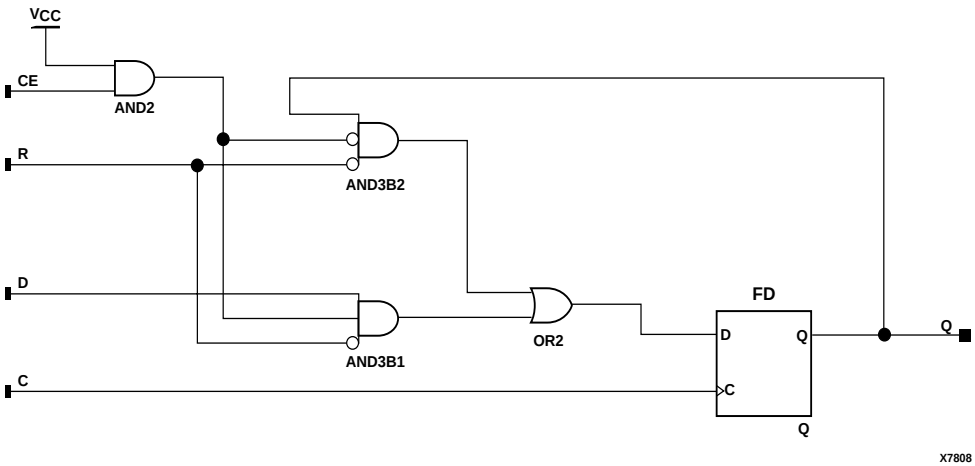
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	CE	D	C	Q
1	X	X	↑	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0



FDRE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdre is
begin
    process (C) begin
        if (C' event and C = '1') then
            if (R = '1') then
                Q <= '0';
            elsif (CE = '1') then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C) begin
   if (R)
      Q <= 0;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDRE should be placed
-- after architecture statement but before begin keyword

component FDRE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDRE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDRE_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDRE should be placed
-- in architecture after the begin keyword

FDRE_INSTANCE_NAME : FDRE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 D  => user_D,
                 R  => user_R);
```

Verilog Instantiation Template

```
FDRE FDRE_instance_name (.Q (user_Q),
                         .C (user_C),
                         .CE (user_CE),
                         .D (user_D),
                         .R (user_R));

defparam FDRE_instance_name.INIT = bit_value;
```

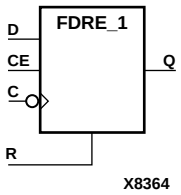
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRE_1

D Flip-Flop with Negative-Clock Edge, Clock Enable, and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDRE_1 is a single D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low and CE is High during the High-to-Low clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	CE	D	C	Q
1	X	X	↓	0
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdre_1 is
begin
    process (C) begin
        if (C' event and C = '0') then
            if (R = '1') then
                Q <= '0';
            elsif (CE = '1') then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C) begin
   if (R)
      Q <= 0;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDRE_1 should be placed
-- after architecture statement but before begin keyword

component FDRE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDRE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDRE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDRE_1 should be placed
-- in architecture after the begin keyword

FDRE_1_INSTANCE_NAME : FDRE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 D  => user_D,
                 R  => user_R);
```

Verilog Instantiation Template

```
FDRE_1 FDRE_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .D (user_D),
                             .R (user_R));

defparam FDRE_1_instance_name.INIT = bit_value;
```

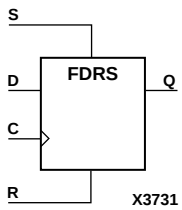
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRS

D Flip-Flop with Synchronous Reset and Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FDRS is a single D-type flip-flop with data (D), synchronous set (S), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High clock (C) transition. (Reset has precedence over Set.) When S is High and R is Low, the flip-flop is set, output High, during the Low-to-High clock transition. When R and S are Low, data on the (D) input is loaded into the flip-flop during the Low-to-High clock transition.

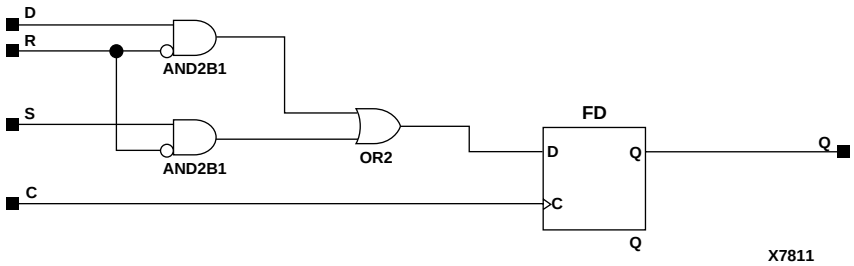
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	S	D	C	Q
1	X	X	↑	0
0	1	X	↑	1
0	0	1	↑	1
0	0	0	↑	0



FDRS Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdrs is
begin
 process (C) begin
  if (C' event and C = '1') then
   if (R = '1') then
      Q <= '0';
   elsif (S = '1') then
    Q <= '1';
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C) begin
   if (R)
      Q <= 0;
   else if (S)
      Q <= 1;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDRS should be placed
-- after architecture statement but before begin keyword

component FDRS
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDRS
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDRS_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDRS should be placed
```

```
-- in architecture after the begin keyword

FDRS_INSTANCE_NAME : FDRS
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 D  => user_D,
                 R  => user_R,
                 S  => user_S);
```

Verilog Instantiation Template

```
FDRS FDRS_instance_name (.Q (user_Q),
                         .C (user_C),
                         .D (user_D),
                         .R (user_R),
                         .S (user_S));

defparam FDRS_instance_name.INIT = bit_value;
```

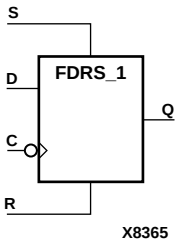
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRS_1

D Flip-Flop with Negative-Clock Edge and Synchronous Reset and Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDRS_1 is a single D-type flip-flop with data (D), synchronous set (S), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low during the High-to-Low clock (C) transition. (Reset has precedence over Set.) When S is High and R is Low, the flip-flop is set, output High, during the High-to-Low clock transition. When R and S are Low, data on the (D) input is loaded into the flip-flop during the High-to-Low clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	S	D	C	Q
1	X	X	↓	0
0	1	X	↓	1
0	0	1	↓	1
0	0	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdrs_1 is
begin
 process (C) begin
  if (C' event and C = '0') then
   if (R = '1') then
      Q <= '0';
   elsif (S = '1') then
    Q <= '1';
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C) begin
   if (R)
      Q <= 0;
   else if (S)
      Q <= 1;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDRS_1 should be placed
-- after architecture statement but before begin keyword

component FDRS_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDRS_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDRS_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDRS_1 should be placed
```

```
-- in architecture after the begin keyword

FDRS_1_INSTANCE_NAME : FDRS_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 D  => user_D,
                 R  => user_R,
                 S  => user_S);
```

Verilog Instantiation Template

```
FDRS_1 FDRS_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .D (user_D),
                             .R (user_R),
                             .S (user_S));

defparam FDRS_1_instance_name.INIT = bit_value;
```

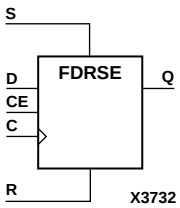
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRSE

D Flip-Flop with Synchronous Reset and Set and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FDRSE is a single D-type flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High clock transition. (Reset has precedence over Set.) When the set (S) input is High and R is Low, the flip-flop is set, output High, during the Low-to-High clock (C) transition. Data on the D input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High clock transition.

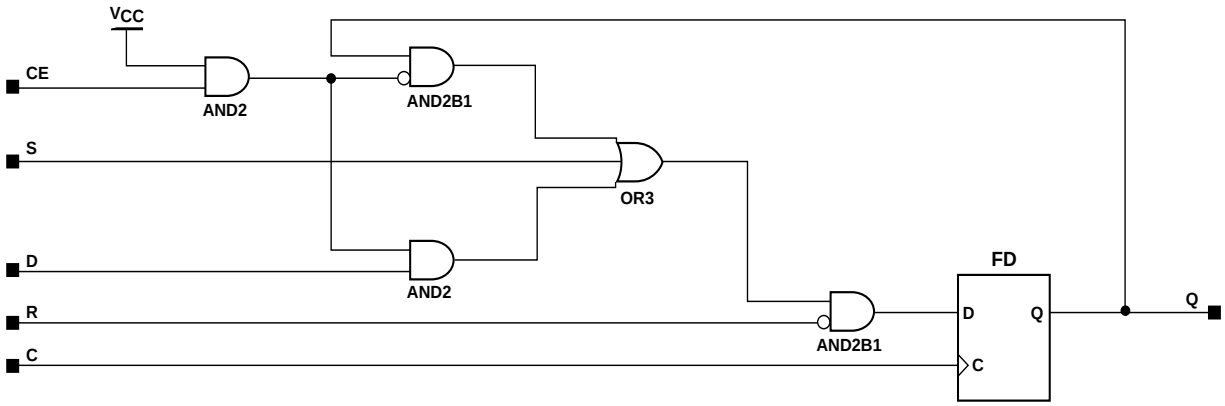
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
R	S	CE	D	C	Q
1	X	X	X	↑	0
0	1	X	X	↑	1
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0



FDRSE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdrse is
begin
    process (C) begin
        if (C' event and C = '1') then
            if (R = '1') then
                Q <= '0';
            elsif (S = '1') then
                Q <= '1';
            elsif (CE = '1') then
                Q <= D;
            end if;
        end if;
end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C) begin
   if (R)
      Q <= 0;
   else if (S)
      Q <= 1;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDRSE should be placed
-- after architecture statement but before begin keyword

component FDRSE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDRSE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDRSE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDRSE should be placed
-- in architecture after the begin keyword

FDRSE_INSTANCE_NAME : FDRSE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 D  => user_D,
                 R  => user_R,
                 S  => user_S);
```

Verilog Instantiation Template

```
FDRSE FDRSE_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CE (user_CE),
                           .D (user_D),
                           .R (user_R),
                           .S (user_S));

defparam FDRSE_instance_name.INIT = bit_value;
```

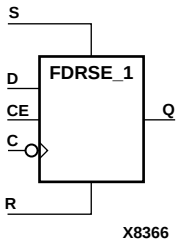
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRSE_1

D Flip-Flop with Negative-Clock Edge, Synchronous Reset and Set, and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDRSE_1 is a single D-type flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during the High-to-Low clock transition. (Reset has precedence over Set.) When the set (S) input is High and R is Low, the flip-flop is set, output High, during the High-to-Low clock (C) transition. Data on the D input is loaded into the flip-flop when R and S are Low and CE is High during the High-to-Low clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
R	S	CE	D	C	Q
1	X	X	X	↓	0
0	1	X	X	↓	1
0	0	0	X	X	No Chg
0	0	1	1	↓	1
0	0	1	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdrse_1 is
begin
    process (C) begin
        if (C' event and C = '0') then
            if (R = '1') then
                Q <= '0';
            elsif (S = '1') then
                Q <= '1';
            elsif (CE = '1') then
                Q <= D;
            end if;
        end if;
end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C) begin
   if (R)
      Q <= 0;
   else if (S)
      Q <= 1;
   else
      Q => D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDRSE_1 should be placed
-- after architecture statement but before begin keyword

component FDRSE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDRSE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDRSE_1_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDRSE_1 should be placed
-- in architecture after the begin keyword

FDRSE_1_INSTANCE_NAME : FDRSE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 D  => user_D,
                 R  => user_R,
                 S  => user_S);
```

Verilog Instantiation Template

```
FDRSE_1 FDRSE_1_instance_name (.Q (user_Q),
                               .C (user_C),
                               .CE (user_CE),
                               .D (user_D),
                               .R (user_R),
                               .S (user_S));

defparam FDRSE_1_instance_name.INIT = bit_value;
```

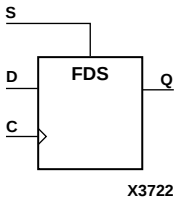
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDS

D Flip-Flop with Synchronous Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FDS is a single D-type flip-flop with data (D) and synchronous set (S) inputs and data output (Q). The synchronous set input, when High, sets the Q output High on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low during the Low-to-High clock (C) transition.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset, output High, when power is applied. For all other devices (XC9500 / XV / XL, CoolRunner XPLA3, CoolRunner-II), the flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FDS will set when GSR is active. For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is preset to active high when GSR is active.

Inputs			Outputs
S	D	C	Q
1	X	↑	1
0	1	↑	1
0	0	↑	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fds is
begin
 process (C) begin
  if (C' event and C = '1') then
   if (S = '1') then
    Q <= '1';
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C) begin
   if (S)
      Q <= 1;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDS should be placed
-- after architecture statement but before begin keyword

component FDS
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDS
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDS_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDS should be placed
-- in architecture after the begin keyword

FDS_INSTANCE_NAME : FDS
   -- synthesis translate_off
   generic map(
```

```
      INIT => bit_value);
-- synthesis translate_on
   port map  (Q  => user_Q,
              C  => user_C,
              D  => user_D,
              S  => user_S);
```

Verilog Instantiation Template

```
FDS FDS_instance_name (.Q (user_Q),
                       .C (user_C),
                       .D (user_D),
                       .S (user_S));

defparam FDS_instance_name.INIT = bit_value;
```

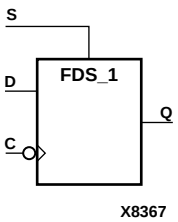
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDS_1

D Flip-Flop with Negative-Edge Clock and Synchronous Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDS_1 is a single D-type flip-flop with data (D) and synchronous set (S) inputs and data output (Q). The synchronous set input, when High, sets the Q output High on the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low during the High-to-Low clock (C) transition.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FDS_1 will set when GSR is active. For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is preset to active high when GSR is active.

Inputs			Outputs
S	D	C	Q
1	X	↓	1
0	1	↓	1
0	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fds_1 is
begin
 process (C) begin
  if (C' event and C = '0') then
   if (S = '1') then
    Q <= '1';
   else
    Q <= D;
   end if;
  end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C) begin
   if (S)
      Q <= 1;
   else
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDS_1 should be placed
-- after architecture statement but before begin keyword

component FDS_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDS_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDS_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDS_1 should be placed
-- in architecture after the begin keyword

FDS_1_INSTANCE_NAME : FDS_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 D  => user_D,
                 S  => user_S);
```

Verilog Instantiation Template

```
FDS_1 FDS_1_instance_name (.Q (user_Q),
                           .C (user_C),
                           .D (user_D),
                           .S (user_S));

defparam FDS_1_instance_name.INIT = bit_value;
```

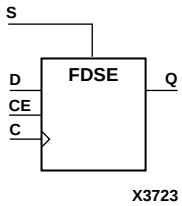
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDSE

D Flip-Flop with Clock Enable and Synchronous Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Macro	Macro



FDSE is a single D-type flip-flop with data (D), clock enable (CE), and synchronous set (S) inputs and data output (Q). The synchronous set (S) input, when High, overrides the clock enable (CE) input and sets the Q output High during the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low and CE is High during the Low-to-High clock (C) transition.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro the flip-flop is asynchronously preset, output High, when power is applied.

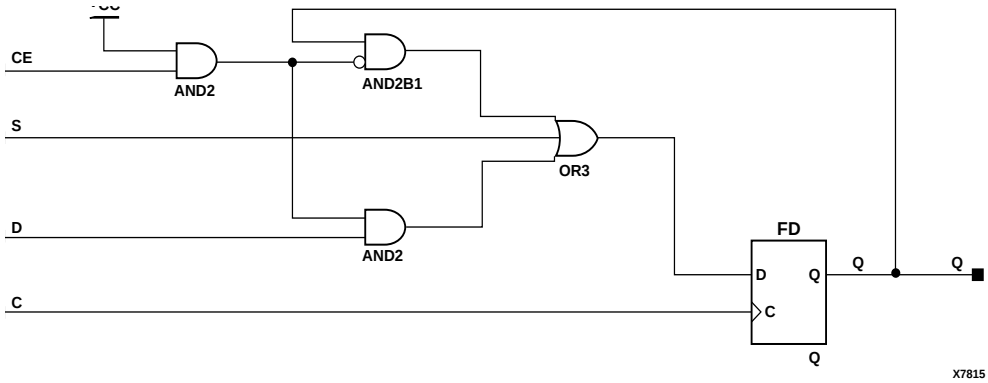
For all other devices (XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II), the flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FDSE will set when GSR is active. For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is preset to active high when GSR is active.

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↑	1
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0



FDSE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdse is
begin
    process (C) begin
        if (C' event and C = '1') then
            if (S = '1') then
                Q <= '1';
            elsif (CE = '1') then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C) begin
   if (S)
      Q <= 1;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDSE should be placed
-- after architecture statement but before begin keyword

component FDSE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDSE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDSE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDSE should be placed
-- in architecture after the begin keyword

FDSE_INSTANCE_NAME : FDSE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 D  => user_D,
                 S  => user_S);
```

Verilog Instantiation Template

```
FDSE FDSE_instance_name (.Q (user_Q),
                         .C (user_C),
                         .CE (user_CE),
                         .D (user_D),
                         .S (user_S));

defparam FDSE_instance_name.INIT = bit_value;
```

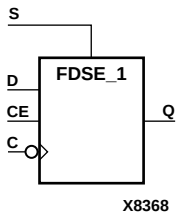
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDSE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Synchronous Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



FDSE_1 is a single D-type flip-flop with data (D), clock enable (CE), and synchronous set (S) inputs and data output (Q). The synchronous set (S) input, when High, overrides the clock enable (CE) input and sets the Q output High during the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low and CE is High during the High-to-Low clock (C) transition.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FDSE_1 will set when GSR is active. For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is preset to active high when GSR is active.

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↓	1
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of fdse_1 is
begin
    process (C) begin
        if (C' event and C = '0') then
            if (S = '1') then
                Q <= '1';
            elsif (CE = '1') then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C) begin
   if (S)
      Q <= 1;
   else if (CE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FDSE_1 should be placed
-- after architecture statement but before begin keyword

component FDSE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDSE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDSE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for FDSE_1 should be placed
-- in architecture after the begin keyword

FDSE_1_INSTANCE_NAME : FDSE_1
   -- synthesis translate_off
```

```
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q  => user_Q,
                 C  => user_C,
                 CE => user_CE,
                 D  => user_D,
                 S  => user_S);
```

Verilog Instantiation Template

```
FDSE_1 FDSE_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .D (user_D),
                             .S (user_S));

defparam FDSE_1_instance_name.INIT = bit_value;
```

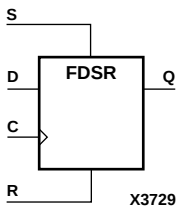
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDSR

D Flip-Flop with Synchronous Set and Reset

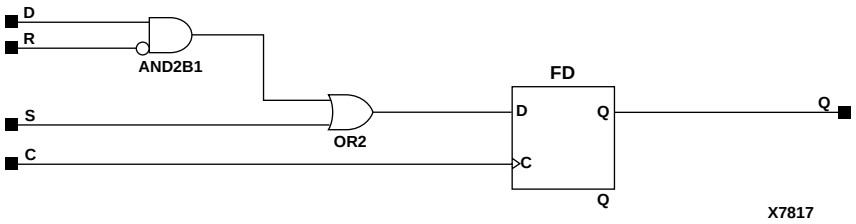
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



FDSR is a single D-type flip-flop with data (D), synchronous reset (R) and synchronous set (S) inputs and data output (Q). When the set (S) input is High, it overrides all other inputs and sets the Q output High during the Low-to-High clock transition. (Set has precedence over Reset.) When reset (R) is High and S is Low, the flip-flop is reset, output Low, on the Low-to-High clock transition. Data on the D input is loaded into the flip-flop when S and R are Low on the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
S	R	D	C	Q
1	X	X	↑	1
0	1	X	↑	0
0	0	1	↑	1
0	0	0	↑	0



FDSR Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fdsr is
begin
        process (C) begin
                if (C' event and C = '1') then
                        if (S = '1') then
                                Q <= '1';
                        elsif (R = '1') then
                                Q <= '0';
                        else
                                Q <= D;
                        end if;
                end if;
        end process;
end Behavioral;
```

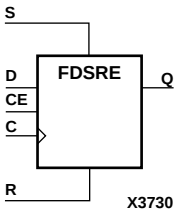
Verilog Inference Code

```
always @ (posedge C) begin
        if (S)
                Q <= 1;
        else if (R)
                Q <= 0;
        else
                Q <= D;
end
```

FDSRE

D Flip-Flop with Synchronous Set and Reset and Clock Enable

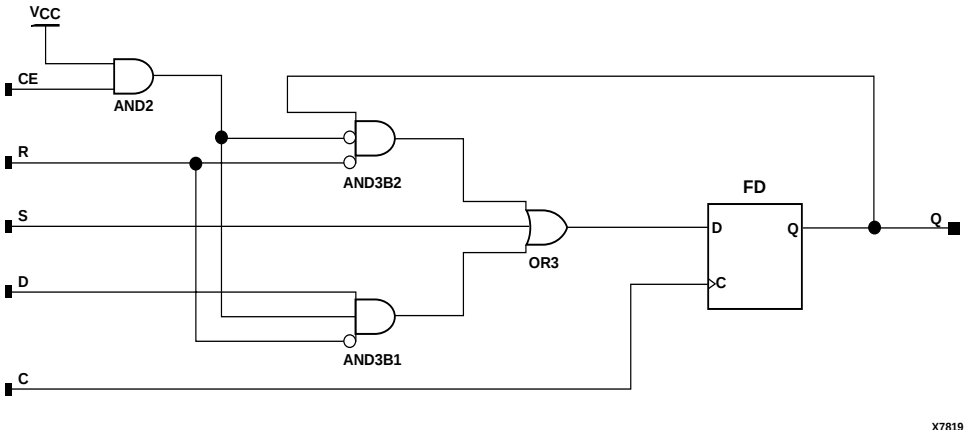
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



FDSRE is a single D-type flip-flop with synchronous set (S), synchronous reset (R), and clock enable (CE) inputs and data output (Q). When synchronous set (S) is High, it overrides all other inputs and sets the Q output High during the Low-to-High clock transition. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, output Q is reset Low during the Low-to-High clock transition. Data is loaded into the flip-flop when S and R are Low and CE is High during the Low-to-high clock transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
S	R	CE	D	C	Q
1	X	X	X	↑	1
0	1	X	X	↑	0
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0



FDSRE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fdsre is

begin
        process (C) begin
                if (C' event and C = '1') then
                        if (S = '1') then
                                Q <= '1';
                        elsif (R = '1') then
                                Q <= '0';
                        elsif (CE = '1') then
                                Q <= D;
                        end if;
                end if;
        end process;
end Behavioral;
```

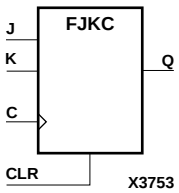
Verilog Inference Code

```
always @ (posedge C) begin
        if (S)
                Q <= 1;
        else if (R)
                Q <= 0;
        else if (CE)
                Q <= D;
end
```

FJKC

J-K Flip-Flop with Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FJKC is a single J-K-type flip-flop with J, K, and asynchronous clear (CLR) inputs and data output (Q). The asynchronous clear (CLR) input, when High, overrides all other inputs and resets the Q output Low. When CLR is Low, the output responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock (C) transition.

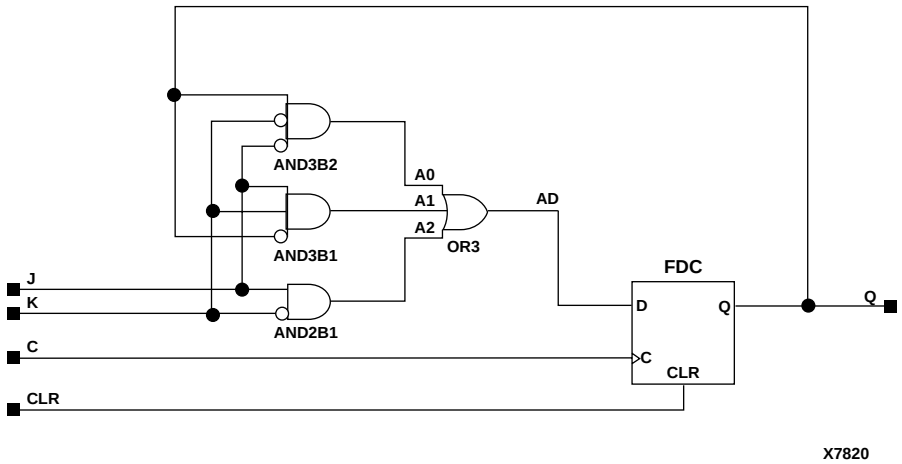
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

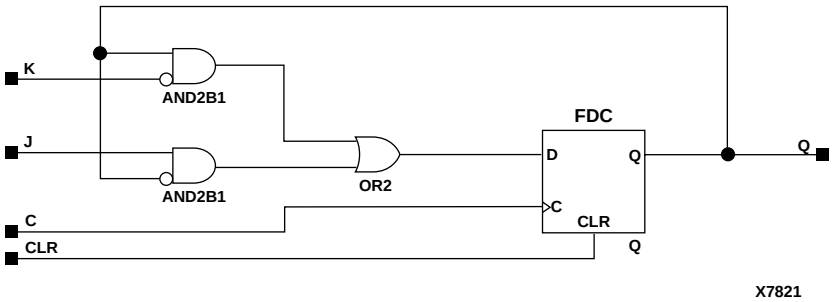
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	J	K	C	Q
1	X	X	X	0
0	0	0	↑	No Chg
0	0	1	↑	0
0	1	0	↑	1
0	1	1	↑	Toggle



FJKC Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



FJKC Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fjkc is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event and C='1') then
                if (J='0') then
                        if (K='1') then
                                Q <= '0';
                        end if;
                else
                        if (K='0') then
                                Q <= '1';
                        else
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

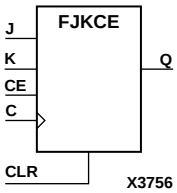
Verilog Inference Code

```
always @(posedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else
        begin
                if (!J)
                begin
                        if (K)
                                Q <= 0;
                end
                else
                begin
                        if (!K)
                                Q <= 1;
                        else
                                Q <= !Q;
                end
        end
end
```

FJKCE

J-K Flip-Flop with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FJKCE is a single J-K-type flip-flop with J, K, clock enable (CE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous clear (CLR), when High, overrides all other inputs and resets the Q output Low. When CLR is Low and CE is High, Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition. When CE is Low, the clock transitions are ignored.

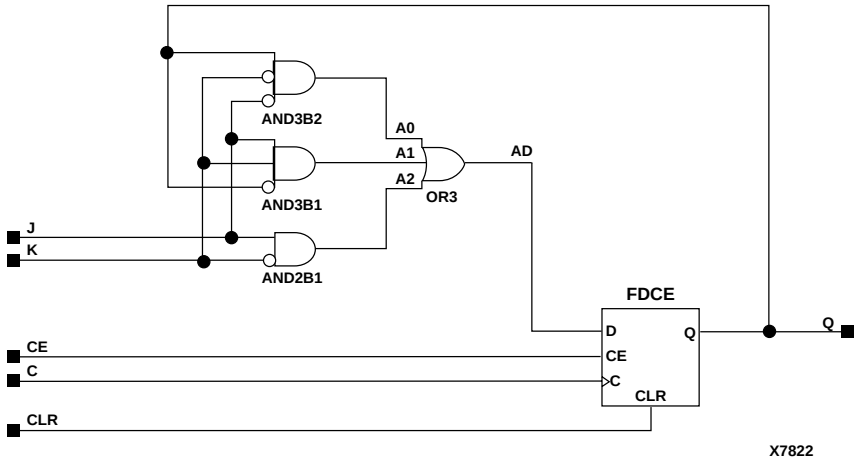
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

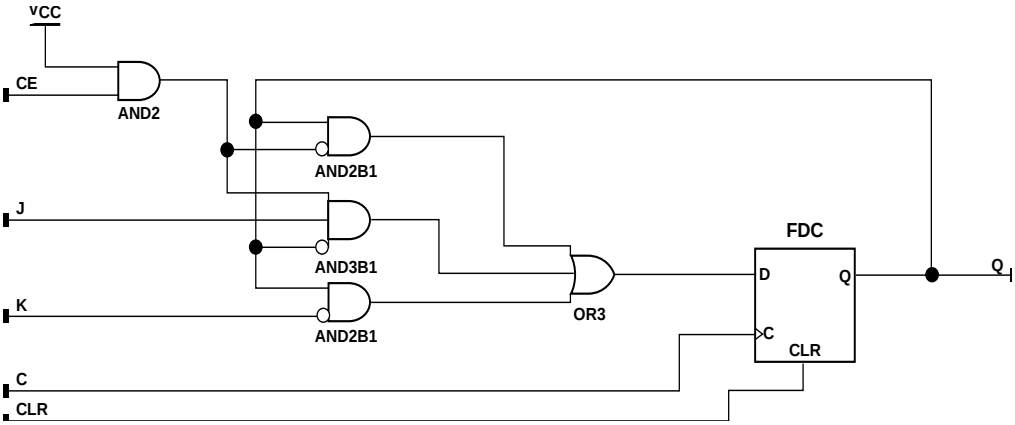
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	CE	J	K	C	Q
1	X	X	X	X	0
0	0	X	X	X	No Chg
0	1	0	0	X	No Chg
0	1	0	1	↑	0
0	1	1	0	↑	1
0	1	1	1	↑	Toggle



FJKCE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



FJKCE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fjkce is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event and C='1') then
                if (CE='1') then
                        if (J='0') then
                                if (K='1') then
                                        Q <= '0';
                                end if;
                        else
                                if (K='0') then
                                        Q <= '1';
                                else
                                        Q <= not Q;
                                end if;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

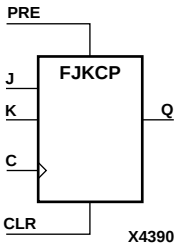
Verilog Inference Code

```
always @(posedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (CE)
        begin
                if (!J)
                begin
                        if (K)
                                Q <= 0;
                end
                else
                begin
                        if (!K)
                                Q <= 1;
                        else
                                Q <= !Q;
                end
        end
end
```

FJKCP

J-K Flip-Flop with Asynchronous Clear and Preset

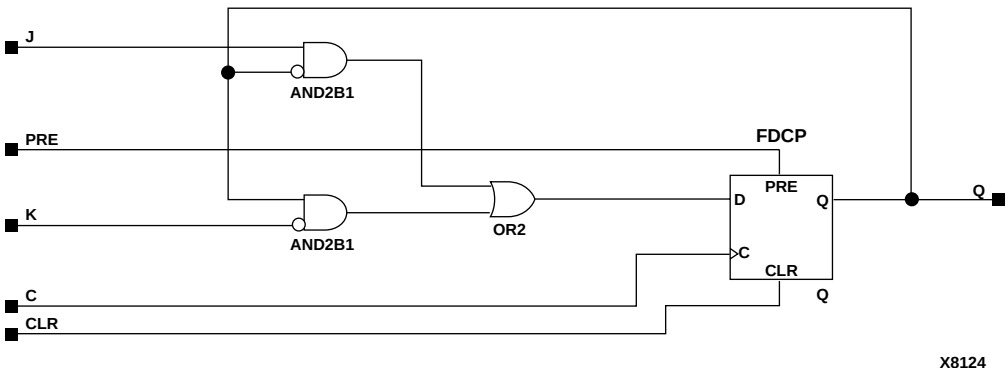
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



FJKCP is a single J-K-type flip-flop with J, K, asynchronous clear (CLR), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous clear input (CLR), when High, overrides all other inputs and resets the Q output Low. The asynchronous preset (PRE) input, when High, overrides all other inputs and sets the Q output High. When CLR and PRE are Low, Q responds to the state of the J and K inputs during the Low-to-High clock transition, as shown in the following truth table.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
CLR	PRE	J	K	C	Q
1	0	X	X	X	0
0	1	X	X	X	1
0	0	0	0	X	No Chg
0	0	0	1	↑	0
0	0	1	0	↑	1
0	0	1	1	↑	Toggle



FJKCP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fjkcp is

begin

process (C, CLR, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (J='0') then
                        if (K='1') then
                                Q <= '0';
                        end if;
                else
                        if (K='0') then
                                Q <= '1';
                        else
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

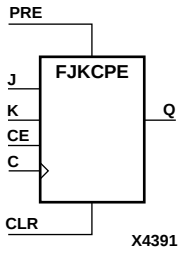
Verilog Inference Code

```
always @(posedge C or posedge CLR or posedge PRE)
begin
        if (CLR)
                Q <= 0;
        else if (PRE)
                Q <= 1;
        else
        begin
                if (!J)
                begin
                        if (K)
                                Q <= 0;
                end
                else
                begin
                        if (!K)
                                Q <= 1;
                        else
                                Q <= !Q;
                end
        end
end
```

FJKCPE

J-K Flip-Flop with Asynchronous Clear and Preset and Clock Enable

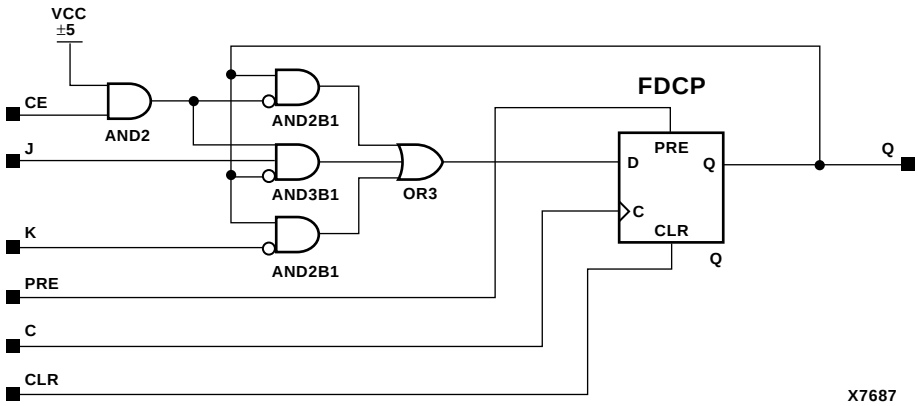
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



FJKCPE is a single J-K-type flip-flop with J, K, asynchronous clear (CLR), asynchronous preset (PRE), and clock enable (CE) inputs and data output (Q). The asynchronous clear input (CLR), when High, overrides all other inputs and resets the Q output Low. The asynchronous preset (PRE) input, when High, overrides all other inputs and sets the Q output High. When CLR and PRE are Low and CE is High, Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs
CLR	PRE	CE	J	K	C	Q
1	X	X	X	X	X	0
0	1	X	X	X	X	1
0	0	0	0	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	0	↑	1
0	0	1	1	1	↑	Toggle



FJKCPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fjkcpe is

begin

process (C, CLR, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (CE='1') then
                        if (J='0') then
                                if (K='1') then
                                        Q <= '0';
                                end if;
                        else
                                if (K='0') then
                                        Q <= '1';
                                else
                                        Q <= not Q;
                                end if;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

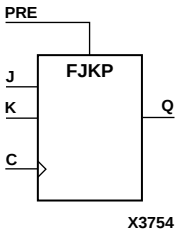
Verilog Inference Code

```
always @(posedge C or posedge CLR or posedge PRE)
begin
        if (CLR)
                Q <= 0;
        else if (PRE)
                Q <= 1;
        else if (CE)
        begin
                if (!J)
                begin
                        if (K)
                                Q <= 0;
                end
                else
                begin
                        if (!K)
                                Q <= 1;
                        else
                                Q <= !Q;
                end
        end
end
```

FJKP

J-K Flip-Flop with Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FJKP is a single J-K-type flip-flop with J, K, and asynchronous preset (PRE) inputs and data output (Q). The asynchronous preset (PRE) input, when High, overrides all other inputs and sets the Q output High. When PRE is Low, the Q output responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition.

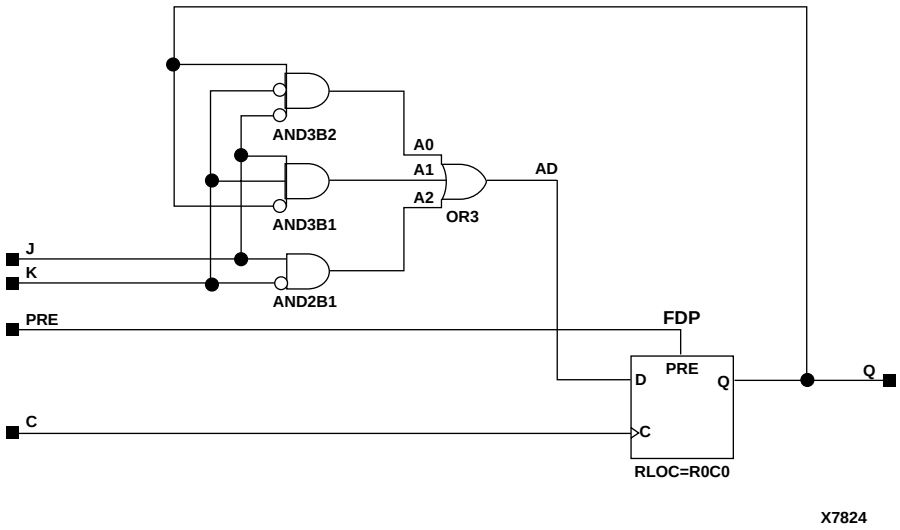
For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

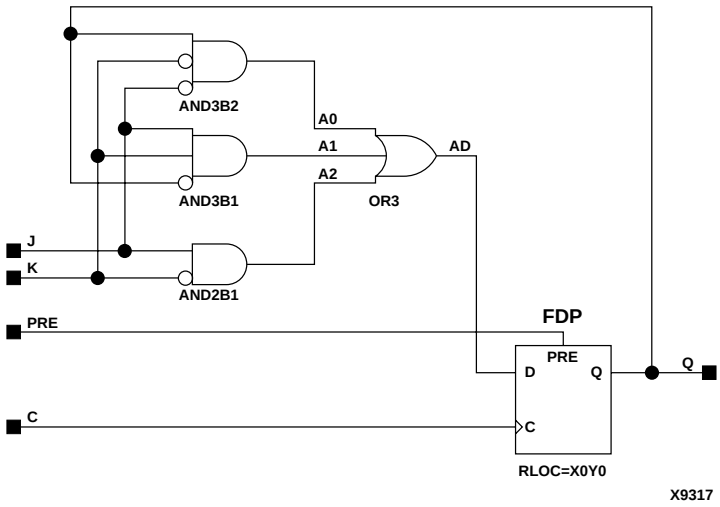
The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

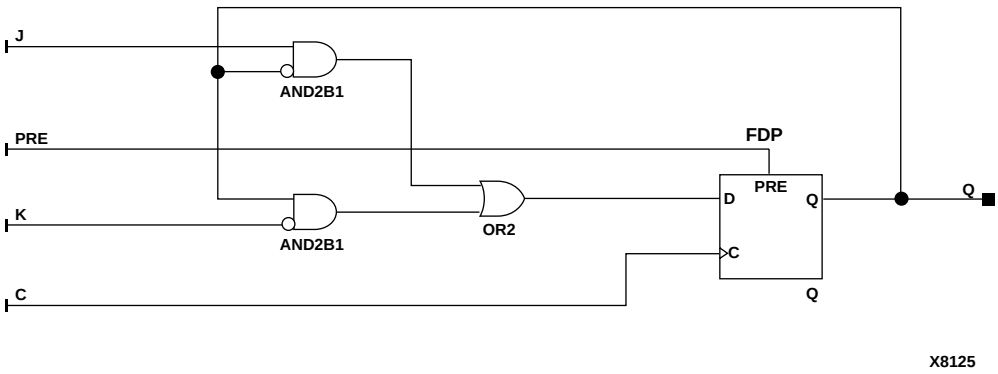
Inputs				Outputs
PRE	J	K	C	Q
1	X	X	X	1
0	0	0	X	No Chg
0	0	1	↑	0
0	1	0	↑	1
0	1	1	↑	Toggle



FJKP Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FJKP Implementation Virtex-II, Virtex-II Pro



FJKP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fjkp is

begin

process (C, PRE)
begin
        if (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (J='0') then
                        if (K='1') then
                                Q <= '0';
                        end if;
                else
                        if (K='0') then
                                Q <= '1';
                        else
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

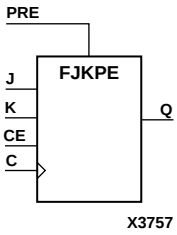
Verilog Inference Code

```
always @(posedge C or posedge PRE)
begin
        if (PRE)
                Q <= 1;
        else
        begin
                if (!J)
                begin
                        if (K)
                                Q <= 0;
                end
                else
                begin
                        if (!K)
                                Q <= 1;
                        else
                                Q <= !Q;
                end
        end
end
```

FJKPE

J-K Flip-Flop with Clock Enable and Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FJKPE is a single J-K-type flip-flop with J, K, clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous preset (PRE), when High, overrides all other inputs and sets the Q output High. When PRE is Low and CE is High, the Q output responds to the state of the J and K inputs, as shown in the truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

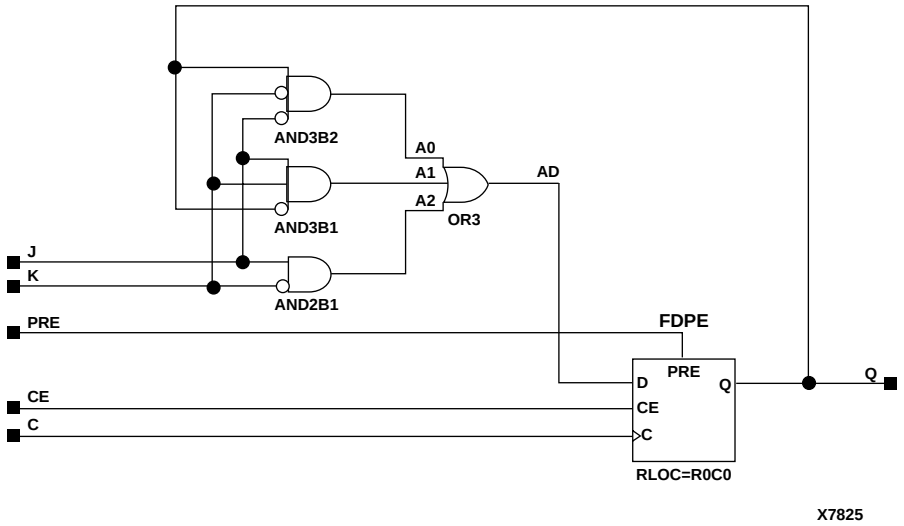
For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

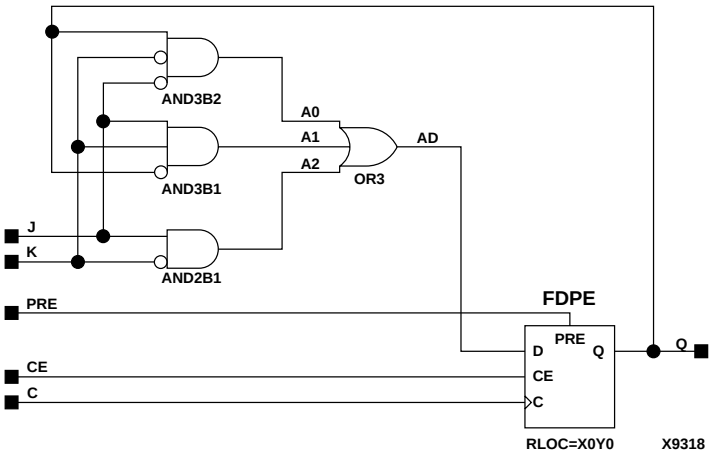
The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

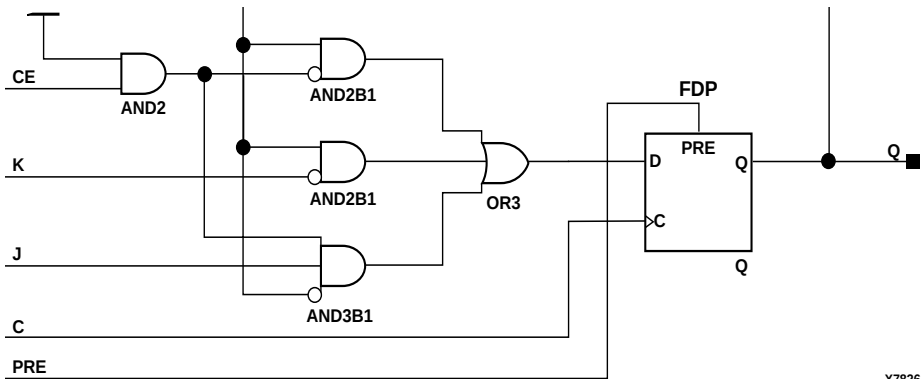
Inputs					Outputs
PRE	CE	J	K	C	Q
1	X	X	X	X	1
0	0	X	X	X	No Chg
0	1	0	0	X	No Chg
0	1	0	1	↑	0
0	1	1	0	↑	1
0	1	1	1	↑	Toggle



FJKPE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FJKPE Implementation Virtex-II, Virtex-II Pro



FJKPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fjkpe is

begin

process (C, PRE)
begin
        if (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (CE='1') then
                        if (J='0') then
                                if (K='1') then
                                        Q <= '0';
                                end if;
                        else
                                if (K='0') then
                                        Q <= '1';
                                else
                                        Q <= not Q;
                                end if;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

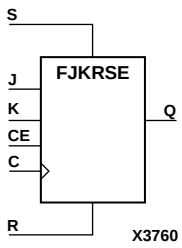
Verilog Inference Code

```
always @(posedge C or posedge PRE)
begin
        if (PRE)
                Q <= 1;
        else if (CE)
        begin
                if (!J)
                begin
                        if (K)
                                Q <= 0;
                end
                else
                begin
                        if (!K)
                                Q <= 1;
                        else
                                Q <= !Q;
                end
        end
end
```

FJKRSE

J-K Flip-Flop with Clock Enable and Synchronous Reset and Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FJKRSE is a single J-K-type flip-flop with J, K, synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). When synchronous reset (R) is High, all other inputs are ignored and output Q is reset Low. (Reset has precedence over Set.) When synchronous set (S) is High and R is Low, output Q is set High. When R and S are Low and CE is High, output Q responds to the state of the J and K inputs, according to the following truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

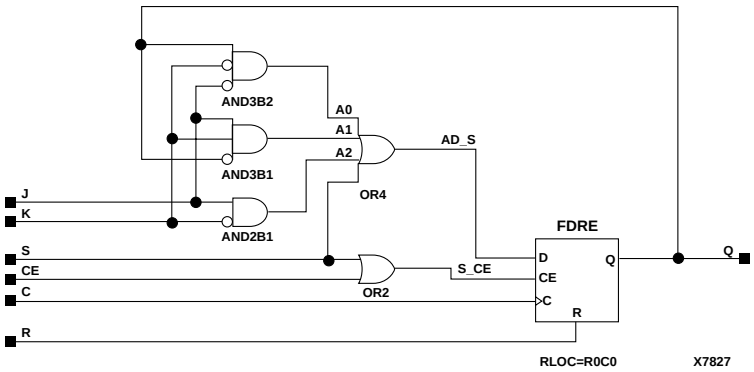
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

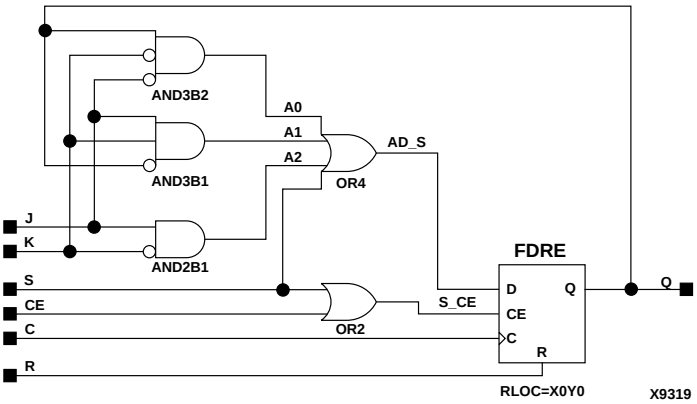
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

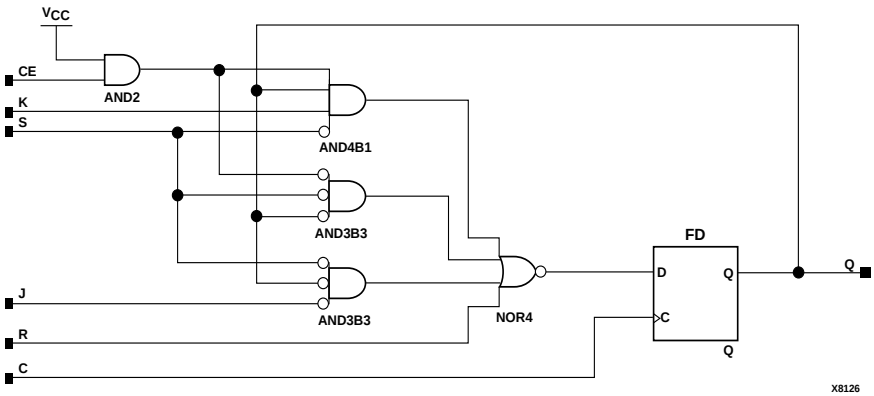
Inputs						Outputs
R	S	CE	J	K	C	Q
1	X	X	X	X	↑	0
0	1	X	X	X	↑	1
0	0	0	X	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	1	↑	Toggle
0	0	1	1	0	↑	1



FJKRSE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FJKRSE Implementation Virtex-II, Virtex-II Pro



FJKRSE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fjkrse is

begin
process (C)
begin
        if (C'event and C='1') then
                if (R='1') then
                        Q <= '0';
                elsif (S='1') then
                        Q <= '1';
                elsif (CE='1') then
                        if (J='0') then
                                if (K='1') then
                                        Q <= '0';
                                end if;
                        else
                                if (K='0') then
                                        Q <= '1';
                                else
                                        Q <= not Q;
                                end if;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

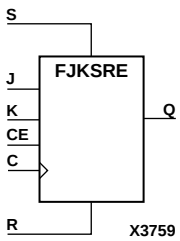
Verilog Inference Code

```
always @(posedge C)
begin
        if (R)
                Q <= 0;
        else if (S)
                Q <= 1;
        else if (CE)
        begin
                if (!J)
                begin
                        if (K)
                                Q <= 0;
                end
                else
                begin
                        if (!K)
                                Q <= 1;
                        else
                                Q <= !Q;
                end
        end
end
```

FJKSRE

J-K Flip-Flop with Clock Enable and Synchronous Set and Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FJKSRE is a single J-K-type flip-flop with J, K, synchronous set (S), synchronous reset (R), and clock enable (CE) inputs and data output (Q). When synchronous set (S) is High, all other inputs are ignored and output Q is set High. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, output Q is reset Low. When S and R are Low and CE is High, output Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

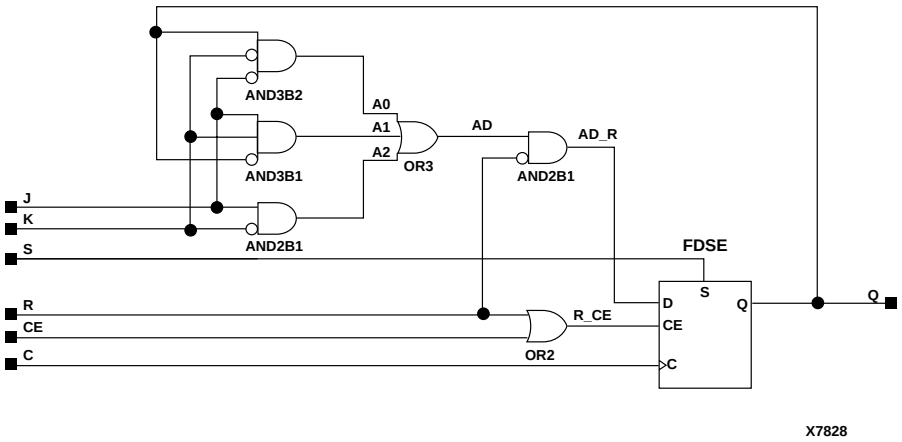
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

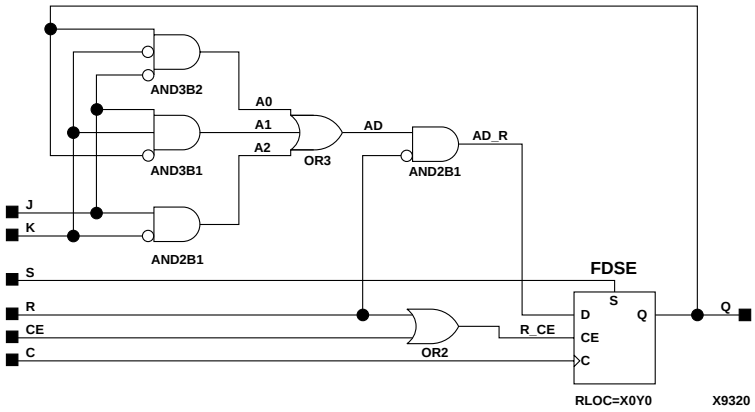
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FJKSRE will set when GSR is active. For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is preset to active high when GSR is active.

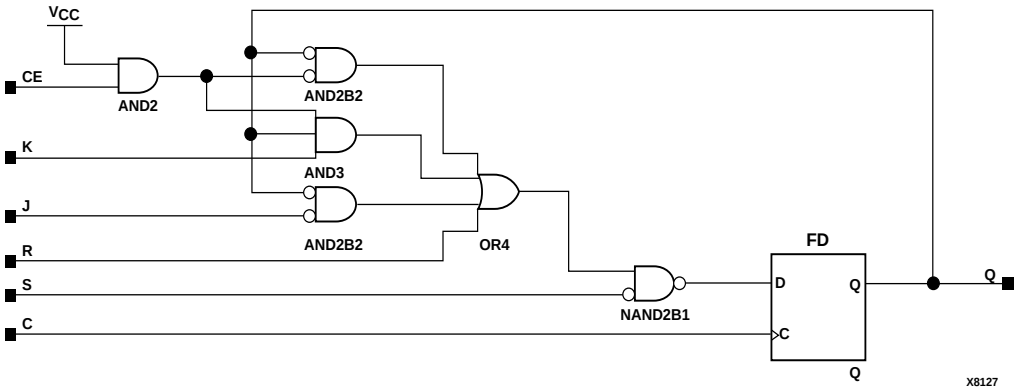
Inputs						Outputs
S	R	CE	J	K	C	Q
1	X	X	X	X	↑	1
0	1	X	X	X	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	0	↑	1
0	0	1	1	1	↑	Toggle



FJKSRE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FJKSRE Implementation Virtex-II, Virtex-II Pro



FJKSRE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of fjksre is

begin

process (C)
begin
        if (C'event and C='1') then
                if (S='1') then
                        Q <= '1';
                elsif (R='1') then
                        Q <= '0';
                elsif (CE='1') then
                        if (J='0') then
                                if (K='1') then
                                        Q <= '0';
                                end if;
                        else
                                if (K='0') then
                                        Q <= '1';
                                else
                                        Q <= not Q;
                                end if;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

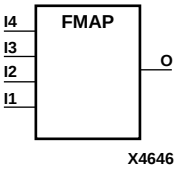
```
always @(posedge C)
begin
        if (S)
                Q <= 1;
        else if (R)
                Q <= 0;
        else if (CE)
        begin
                if (!J)
                begin
                        if (K)
                                Q <= 0;
                end
                else
                begin
                        if (!K)
                                Q <= 1;
                        else
                                Q <= !Q;
                end
```

```
                end
        end
```

FMAP

F Function Generator Partitioning Control Symbol

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



The FMAP symbol is used to map logic to the function generator of a slice. See the appropriate CAE tool interface user guide for information about specifying this attribute in your schematic design editor.

The MAP=*type* parameter can be used with the FMAP symbol to further define how much latitude you want to give the mapping program. The following table shows MAP option characters and their meanings.

MAP Option Character	Function
P	Pins.
C	Closed — Adding logic to or removing logic from the CLB is not allowed.
L	Locked — Locking CLB pins.
O	Open — Adding logic to or removing logic from the CLB is allowed.
U	Unlocked — No locking on CLB pins.

Possible types of MAP parameters for FMAP are MAP=PUC, MAP=PLC, MAP=PLO, and MAP=PUO. The default parameter is PUO. If one of the "open" parameters is used (PLO or PUO), only the output signals must be specified.

Note Currently, only PUC and PUO are observed. PLC and PLO are translated into PUC and PUO, respectively.

The FMAP symbol can be assigned to specific CLB locations using LOC attributes. See the "LOC" section of the *Constraints Guide* and the appropriate CAE tool interface user guide for more information on assigning LOC attributes.

Usage

FMAPs are generally inferred with the logic portions of the HDL code. Xilinx suggests that you instantiate FMAPs only if you have a need to implicitly specify the logic mapping, or if you need to manually place or relationally place the logic.

VHDL Instantiation Template

```
-- Component Declaration for FMAP should be placed
-- after architecture statement but before begin keyword

component FMAP
   port (I1 : in STD_ULOGIC;
         I2 : in STD_ULOGIC;
         I3 : in STD_ULOGIC;
         I4 : in STD_ULOGIC;
         O  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FMAP
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here


-- Component Instantiation for FMAP should be placed
-- in architecture after the begin keyword

FMAP_INSTANCE_NAME : FMAP

      port map  (I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3,
                 I4 => user_I4,
                 O  => user_O);
```

Verilog Instantiation Template

```
FMAP FMAP_instance_name (.I1 (user_I1),
                         .I2 (user_I2),
                         .I3 (user_I3),
                         .I4 (user_I4),
                         .O (user_O));
```

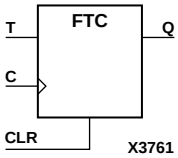
Commonly Used Constraints

BEL, BLKNM, HBLKNM, HU_SET, LOC, MAP, U_SET

FTC

Toggle Flip-Flop with Toggle Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTC is a synchronous, resettable toggle flip-flop. The asynchronous clear (CLR) input, when High, overrides all other inputs and resets the data output (Q) Low. The Q output toggles, or changes state, when the toggle enable (T) input is High and CLR is Low during the Low-to-High clock transition.

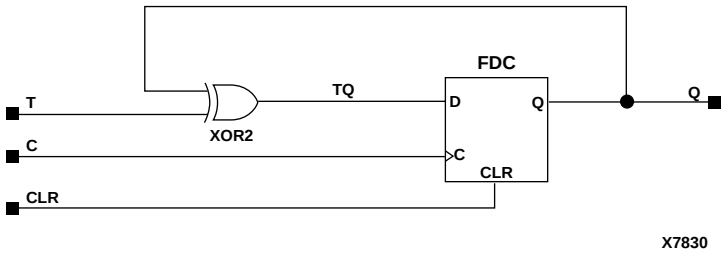
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

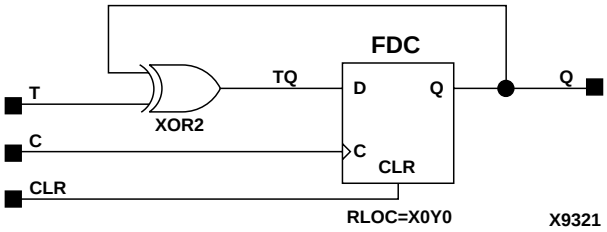
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

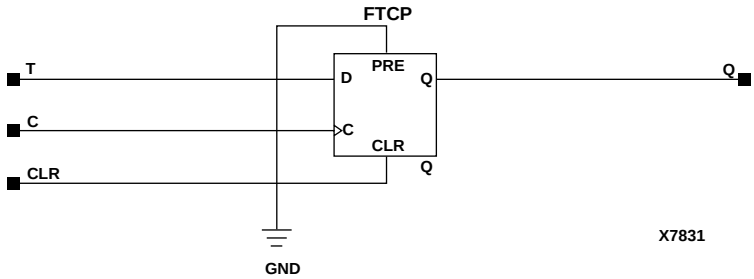
Inputs			Outputs
CLR	T	C	Q
1	X	X	0
0	0	X	No Chg
0	1	↑	Toggle



FTC Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTC Implementation Virtex-II, Virtex-II Pro



FTC Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Inference Code

```
architecture Behavioral of ftc is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event and C='1') then
                if (T='1') then
                        Q <= not Q;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @(posedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (T)
                Q <= !Q;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FTC should be placed
-- after architecture statement but before begin keyword

component FTC
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         T    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FTC
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes here


-- Component Instantiation for FTC should be placed
-- in architecture after the begin keyword

FTC_INSTANCE_NAME : FTC
      port map  (Q => user_Q,
                 C => user_C,
                 CLR => user_CLR,
                 T => user_T);
```

Verilog Instantiation Template

```
FTC FTC_instance_name (.Q (user_Q),
                       .C (user_C),
                       .CLR (user_CLR),
                       .T (user_T));
```

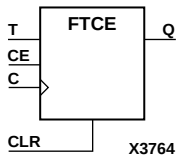
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FTCE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTCE is a toggle flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the data output (Q) is reset Low. When CLR is Low and toggle enable (T) and clock enable (CE) are High, Q output toggles, or changes state, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

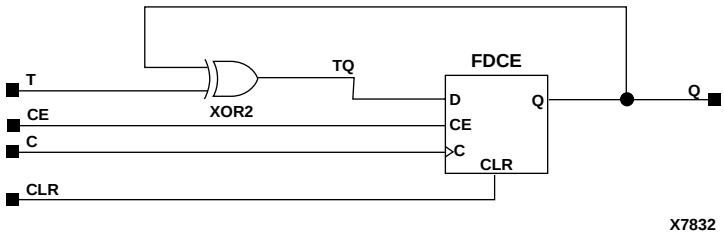
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

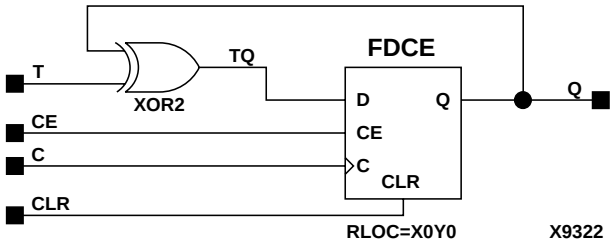
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

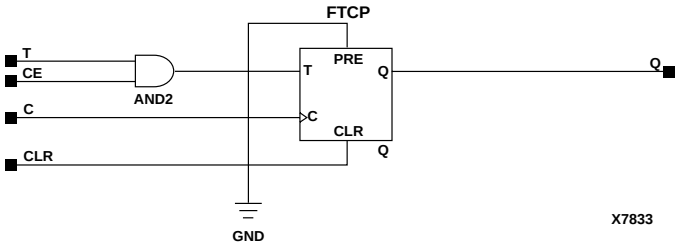
Inputs				Outputs
CLR	CE	T	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	0	X	No Chg
0	1	1	↑	Toggle



FTCE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTCE Implementation Virtex-II, Virtex-II Pro



FTCE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of ftce is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event and C='1') then
                if (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

Q <= Q;

end Behavioral;
```

Verilog Inference Code

```
always @(posedge C or posedge CLR)
begin
        if (CLR)
              Q <= 0;
        else if (CE)
                if (T)
                  Q <= !Q;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FTCE should be placed
-- after architecture statement but before begin keyword

component FTCE
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         PRE  : in STD_ULOGIC;
         T    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FTCE
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes here


-- Component Instantiation for FTCE should be placed
-- in architecture after the begin keyword

FTCE_INSTANCE_NAME : FTCE
      port map  (Q => user_Q,
                 C => user_C,
                 CE => user_CE,
                 CLR => user_CLR,
                 PRE => user_PRE,
                 T => user_T);
```

Verilog Instantiation Template

```
FTCE FTCE_instance_name (.Q (user_Q),
                         .C (user_C),
                         .CE (user_CE),
                         .CLR (user_CLR),
                         .PRE (user_PRE),
                         .T (user_T));
```

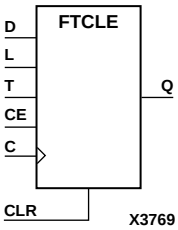
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FTCLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTCLE is a toggle / loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High and CLR is Low, clock enable (CE) is overridden and the data on data input (D) is loaded into the flip-flop during the Low-to-High clock (C) transition. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High clock transition. When CE is Low, clock transitions are ignored.

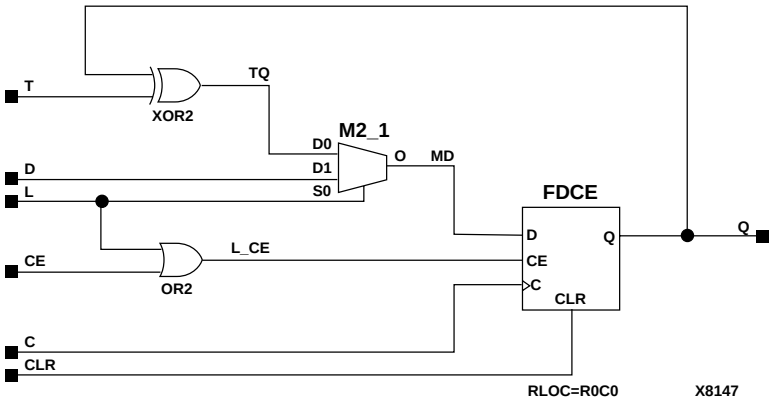
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

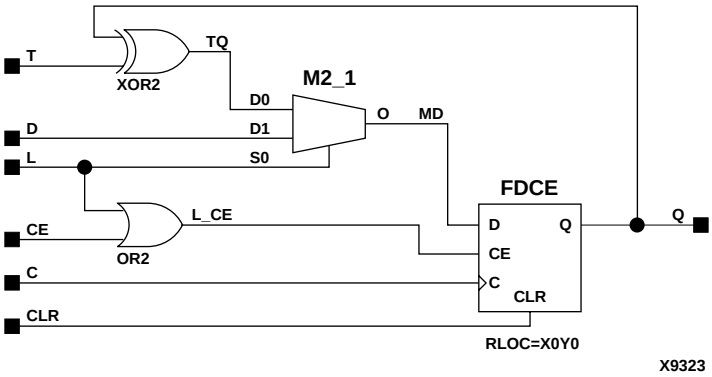
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

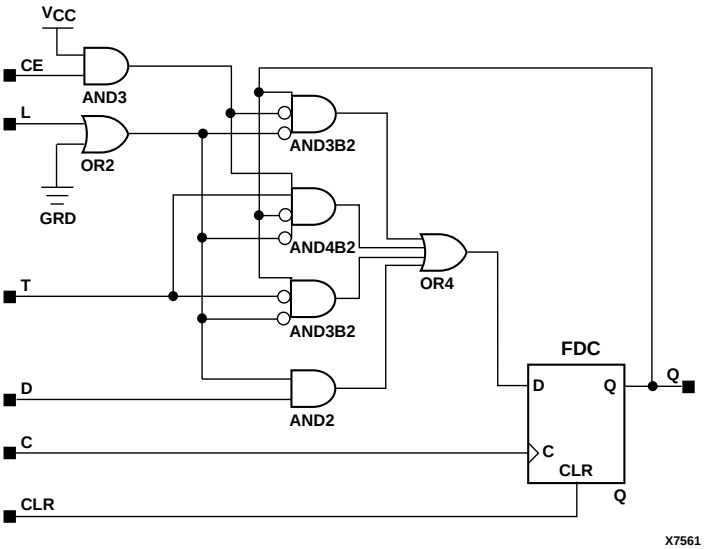
Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	X	X	1	↑	1
0	1	X	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle



FTCLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTCLE Implementation Virtex-II, Virtex-II Pro



FTCLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftcle is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event and C='1') then
                if (L='1') then
                        Q <= D;
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

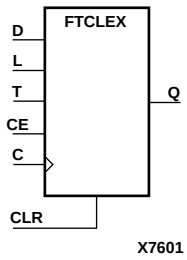
Verilog Inference Code

```
always @(posedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTCLEX

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



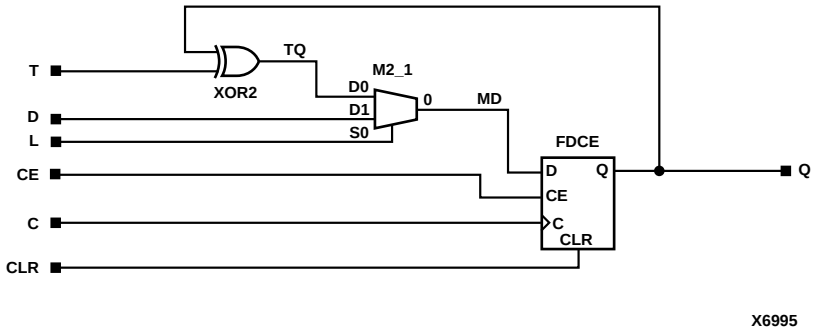
FTCLEX is a toggle / loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High, CLR is Low, and CE is High, the data on data input (D) is loaded into the flip-flop during the Low-to-High clock (C) transition. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High clock transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

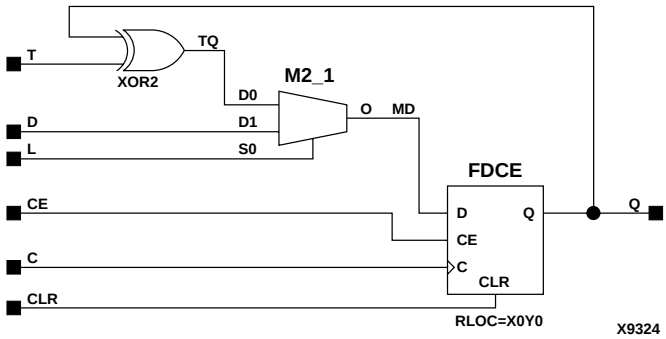
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	1	X	1	↑	1
0	1	1	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle



FTCLEX Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTCLEX Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftclex is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event and C='1') then
                if (CE='1') then
                        if (L='1') then
                                Q <= D;
                        elsif (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

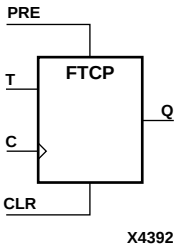
Verilog Inference Code

```
always @(posedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTCP

Toggle Flip-Flop with Toggle Enable and Asynchronous Clear and Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	Primitive	Primitive	Primitive



FTCP is a toggle flip-flop with toggle enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) input is High, all other inputs are ignored and Q is set High. When the toggle enable input (T) is High and CLR and PRE are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	PRE	T	C	Q
1	0	X	X	0
0	1	X	X	1
0	0	0	X	No Chg
0	0	1	↑	Toggle

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of ftcp is

begin
process (C, CLR, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (T='1') then
                  Q <= not Q;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @(posedge C or posedge CLR or posedge PRE)
begin
        if (CLR)
                Q <= 0;
        else if (PRE)
                Q <= 1;
        else if (T)
                Q <= !Q;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FTCP should be placed
-- after architecture statement but before begin keyword

component FTCP
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         PRE  : in STD_ULOGIC;
         T    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FTCP
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes here


-- Component Instantiation for FTCP should be placed
-- in architecture after the begin keyword

FTCP_INSTANCE_NAME : FTCP
      port map  (Q => user_Q,
                 C => user_C,
                 CLR => user_CLR,
                 PRE => user_PRE,
                 T => user_T);
```

Verilog Instantiation Template

```
FTCP FTCP_instance_name (.Q (user_Q),
                         .C (user_C),
                         .CLR (user_CLR),
                         .PRE (user_PRE),
                         .T (user_T));
```

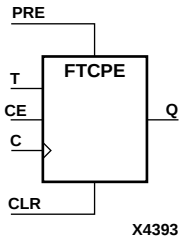
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FTCPE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset

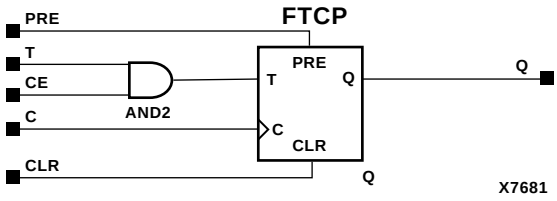
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



FTCPE is a toggle flip-flop with toggle and clock enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) input is High, all other inputs are ignored and Q is set High. When the toggle enable input (T) and the clock enable input (CE) are High and CLR and PRE are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
CLR	PRE	CE	T	C	Q
1	0	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle



FTCPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftcpe is

begin

process (C, CLR, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

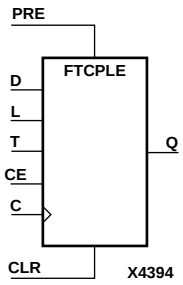
Verilog Inference Code

```
always @(posedge C or posedge CLR or posedge PRE)
begin
        if (CLR)
                Q <= 0;
        else if (PRE)
                Q <= 1;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTCPLE

Loadable Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset

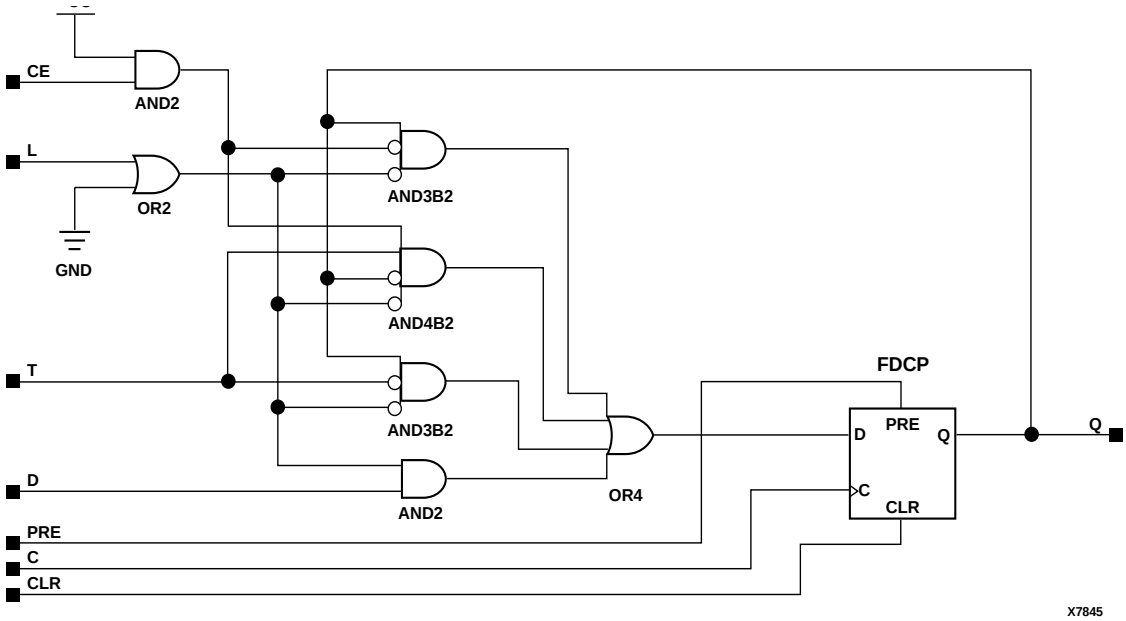
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	Macro	Macro	Macro



FTCPLE is a loadable toggle flip-flop with toggle and clock enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) input is High, all other inputs are ignored and Q is set High. When the load input (L) is High, the clock enable input (CE) is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High clock transition. When the toggle enable input (T) and the clock enable input (CE) are High and CLR, PRE, and L are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs							Outputs
CLR	PRE	L	CE	T	C	D	Q
1	X	X	X	X	X	X	0
0	1	X	X	X	X	X	1
0	0	1	X	X	↑	0	0
0	0	1	X	X	↑	1	1
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	↑	X	Toggle



FTCPLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftcple is

begin

process (C, CLR, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (L='1') then
                        Q <= D;
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

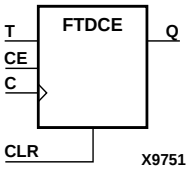
Verilog Inference Code

```
always @(posedge C or posedge CLR or posedge PRE)
begin
        if (CLR)
                Q <= 0;
        else if (PRE)
                Q <= 1;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTDCE

Dual Edge Triggered Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

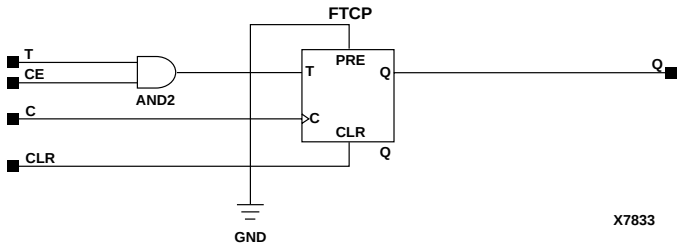
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FTDCE is a dual edge triggered toggle flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the data output (Q) is reset Low. When CLR is Low and toggle enable (T) and clock enable (CE) are High, Q output toggles, or changes state, during the Low-to-High and High-to-Low clock (C) transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	CE	T	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	0	X	No Chg
0	1	1	↑	Toggle
0	1	1	↓	Toggle



FTDCE Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftdce is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event) then
                if (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

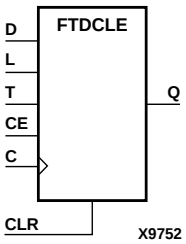
Verilog Inference Code

```
always @(posedge C or negedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTDCLE

Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

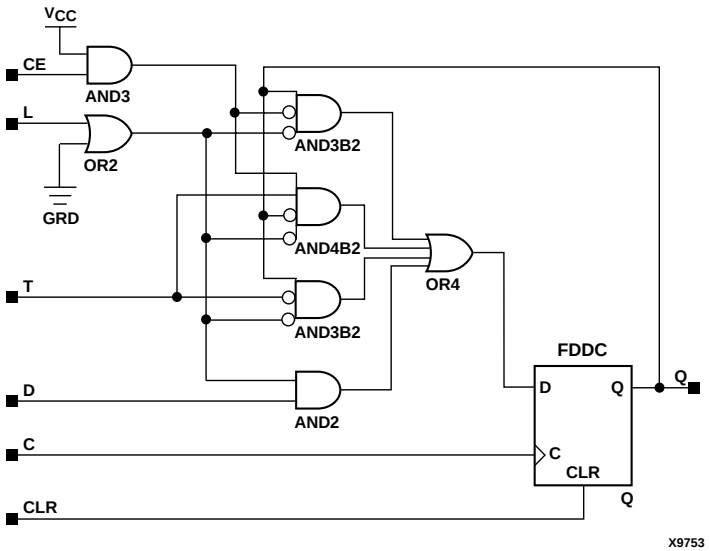
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FTDCLE is a dual edge triggered toggle / loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High and CLR is Low, clock enable (CE) is overridden and the data on data input (D) is loaded into the flip-flop during the Low-to-High and High-to-Low clock (C) transitions. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	X	X	1	↑	1
0	1	X	X	1	↓	1
0	1	X	X	0	↑	0
0	1	X	X	0	↓	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle
0	0	1	1	X	↓	Toggle



FTDCLE Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftdcle is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event) then
                if (L='1') then
                        Q <= D;
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

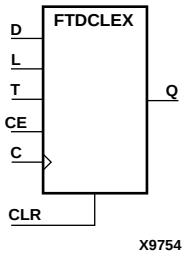
Verilog Inference Code

```
always @(posedge C or negedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTDCLEX

Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Macro



FTDCLEX is a dual edge triggered toggle/loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High, CLR is Low, and CE is High, the data on data input (D) is loaded into the flip-flop during the Low-to-High and High-to-Low clock (C) transitions. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	1	X	1	↑	1
0	1	1	X	1	↓	1
0	1	1	X	0	↑	0
0	1	1	X	0	↓	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle
0	0	1	1	X	↓	Toggle

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftdclex is

begin

process (C, CLR)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (C'event) then
                if (CE='1') then
                        if (L='1') then
                                Q <= D;
                        elsif (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

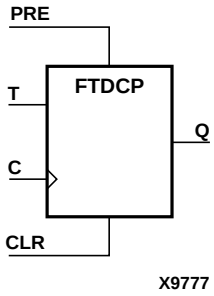
Verilog Inference Code

```
always @(posedge C or negedge C or posedge CLR)
begin
        if (CLR)
                Q <= 0;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTDCP

Toggle Flip-Flop with Toggle Enable and Asynchronous Clear and Preset

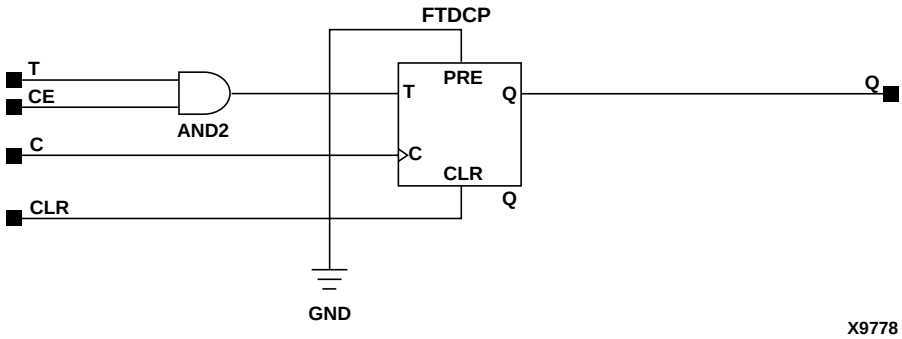
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	Primitive	Primitive	Primitive



FTDCP is a toggle flip-flop with toggle enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) input is High, all other inputs are ignored and Q is set High. When the toggle enable input (T) is High and CLR and PRE are Low, output Q toggles, or changes state, during the Low-to-High and High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	PRE	T	C	Q
1	0	X	X	0
0	1	X	X	1
0	0	0	X	No Chg
0	0	1	↑	Toggle
0	0	1	↓	Toggle



FTDCP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftdcp is

begin

process (C, CLR, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (C'event) then
                if (T='1') then
                        Q <= not Q;
                end if;
        end if;
end process;

end Behavioral;
```

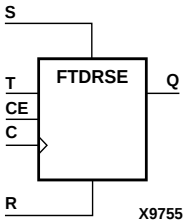
Verilog Inference Code

```
always @(posedge C or negedge C or posedge CLR or posedge PRE)
begin
        if (CLR)
                Q <= 0;
        else if (PRE)
                Q <= 1;
        else if (T)
                Q <= !Q;
end
```

FTDRSE

Dual Edge Triggered Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

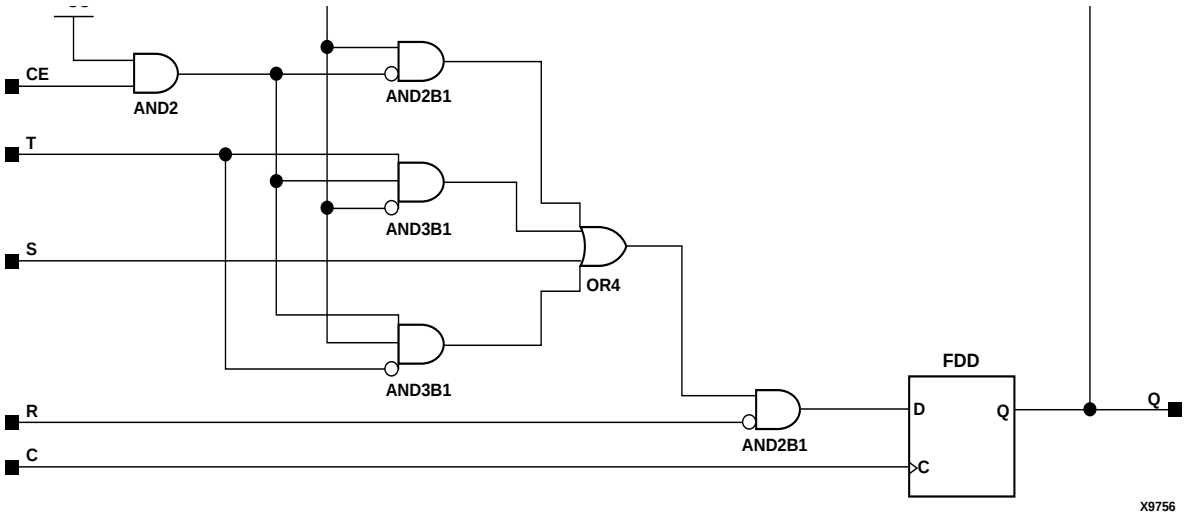
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FTDRSE is a dual edge triggered toggle flip-flop with toggle and clock enable and synchronous reset and set. When the synchronous reset input (R) is High, it overrides all other inputs and the data output (Q) is reset Low. When the synchronous set input (S) is High and R is Low, clock enable input (CE) is overridden and output Q is set High. (Reset has precedence over Set.) When toggle enable input (T) and CE are High and R and S are Low, output Q toggles, or changes state, during the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
R	S	CE	T	C	Q
1	X	X	X	↑	0
1	X	X	X	↓	0
0	1	X	X	↑	1
0	1	X	X	↓	1
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle
0	0	1	1	↓	Toggle



FTDRSE Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftdrse is

begin

process (C)
begin
        if (C'event) then
                if (R='1') then
                        Q <= '0';
                elsif (S='1') then
                        Q <= '1';
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

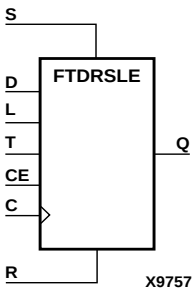
Verilog Inference Code

```
always @(posedge C or negedge C)
begin
        if (R)
                Q <= 0;
        else if (S)
                Q <= 1;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTDRSLE

Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

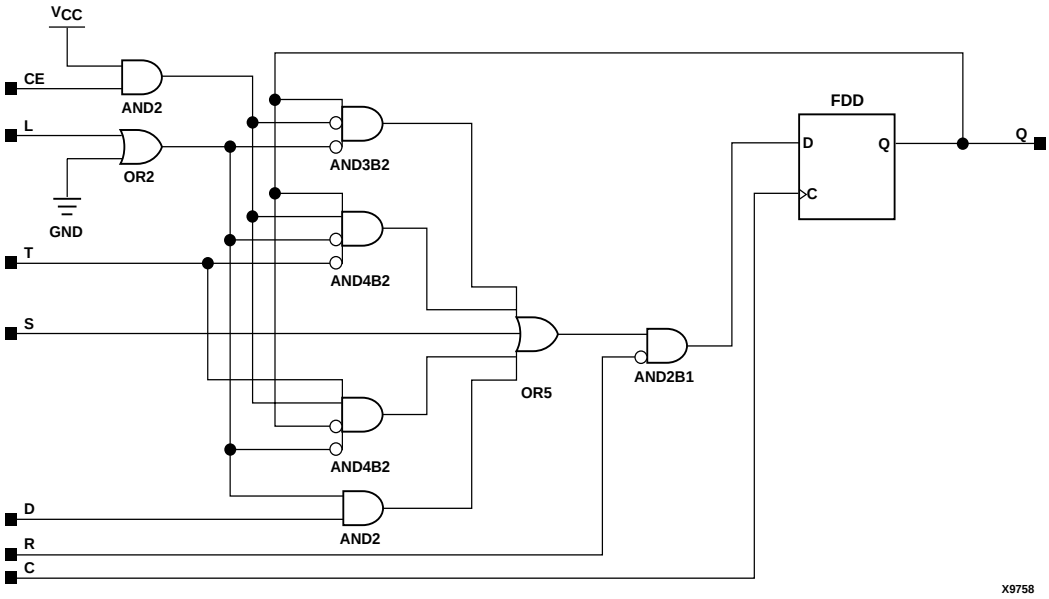
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro



FTDRSLE is a dual edge triggered toggle / loadable flip-flop with toggle and clock enable and synchronous reset and set. The synchronous reset input (R), when High, overrides all other inputs and resets the data output (Q) Low. (Reset has precedence over Set.) When R is Low and synchronous set input (S) is High, the clock enable input (CE) is overridden and output Q is set High. When R and S are Low and load enable input (L) is High, CE is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High and High-to-Low clock transitions. When R, S, and L are Low and CE is High, output Q toggles, or changes state, during the Low-to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs							Outputs
R	S	L	CE	T	D	C	Q
1	0	X	X	X	X	↑	0
1	0	X	X	X	X	↓	0
0	1	X	X	X	X	↑	1
0	1	X	X	X	X	↓	1
0	0	1	X	X	1	↑	1
0	0	1	X	X	1	↓	1
0	0	1	X	X	0	↑	0
0	0	1	X	X	0	↓	0
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	X	↑	Toggle
0	0	0	1	1	X	↓	Toggle



FTDRSLE Implementation CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftdrsle is

begin

process (C)
begin
        if (C'event) then
                if (R='1') then
                        Q <= '0';
                elsif (S='1') then
                        Q <= '1';
                elsif (L='1') then
                        Q <= D;
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

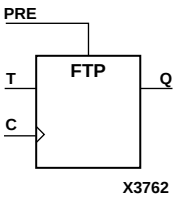
Verilog Inference Code

```
always @(posedge C or negedge C)
begin
        if (R)
                Q <= 0;
        else if (S)
                Q <= 1;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTP

Toggle Flip-Flop with Toggle Enable and Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTP is a toggle flip-flop with toggle enable and asynchronous preset. When the asynchronous preset (PRE) input is High, all other inputs are ignored and output Q is set High. When toggle-enable input (T) is High and PRE is Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition.

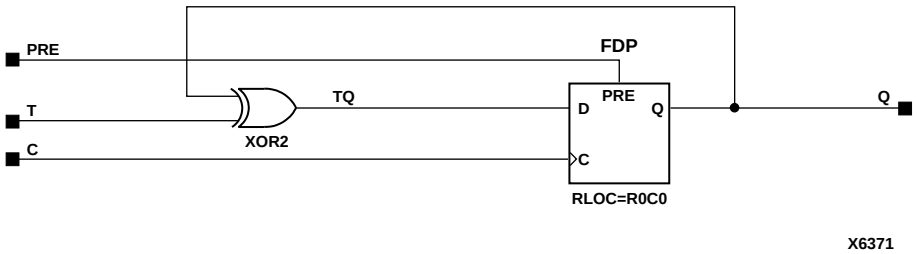
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset to output High, when power is applied.

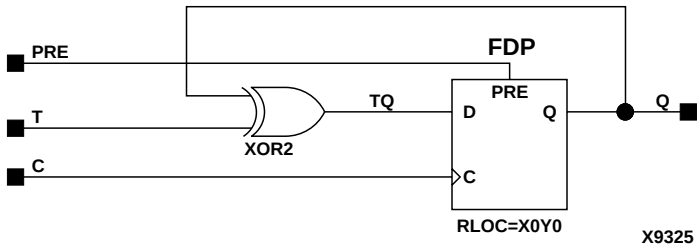
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, or the STARTUP_VIRTEX symbol.

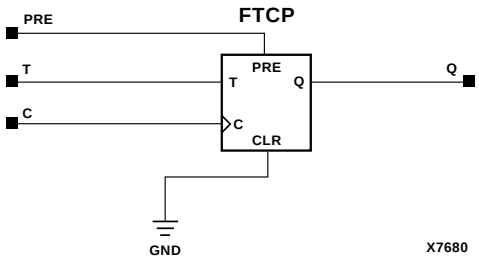
Inputs			Outputs
PRE	T	C	Q
1	X	X	1
0	0	X	No Chg
0	1	↑	Toggle



FTP Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTP Implementation Virtex-II, Virtex-II Pro



FTP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of ftp is

begin

process (C, PRE)
begin
        if (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (T='1') then
                        Q <= not Q;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @(posedge C or posedge PRE)
begin
        if (PRE)
                Q <= 1;
        else if (T)
                Q <= !Q;
end
```

VHDL Instantiation Template

```
-- Component Declaration for FTP should be placed
-- after architecture statement but before begin keyword

component FTP
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC;
         T    : in STD_ULOGIC);
end component;

-- Component Attribute specification for FTP
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes here


-- Component Instantiation for FTP should be placed
-- in architecture after the begin keyword

FTP_INSTANCE_NAME : FTP
      port map  (Q => user_Q,
                 C => user_C,
                 PRE => user_PRE,
                 T => user_T);
```

Verilog Instantiation Template

```
FTP FTP_instance_name (.Q (user_Q),
                       .C (user_C),
                       .PRE (user_PRE),
                       .T (user_T));
```

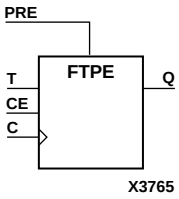
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FTPE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTPE is a toggle flip-flop with toggle and clock enable and asynchronous preset. When the asynchronous preset (PRE) input is High, all other inputs are ignored and output Q is set High. When the toggle enable input (T) is High, clock enable (CE) is High, and PRE is Low, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

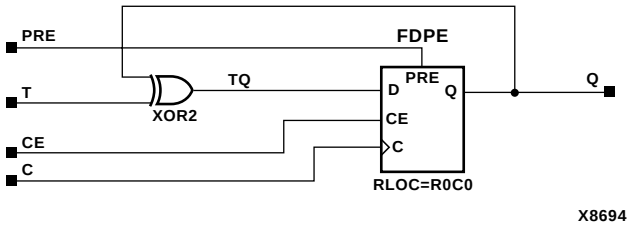
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset to output High, when power is applied.

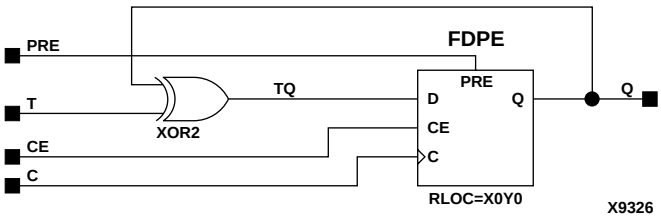
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

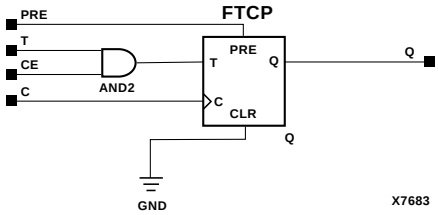
Inputs				Outputs
PRE	CE	T	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	X	No Chg
0	1	1	↑	Toggle



FTPE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTPE Implementation Virtex-II, Virtex-II Pro



FTPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftpe is

begin

process (C, PRE)
begin
        if (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

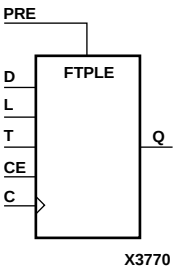
Verilog Inference Code

```
always @(posedge C or posedge PRE)
begin
        if (PRE)
                Q <= 1;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTPLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTPLE is a toggle/loadable flip-flop with toggle and clock enable and asynchronous preset. When the asynchronous preset input (PRE) is High, all other inputs are ignored and output Q is set High. When the load enable input (L) is High and PRE is Low, the clock enable (CE) is overridden and the data (D) is loaded into the flip-flop during the Low-to-High clock transition. When L and PRE are Low and toggle-enable input (T) and CE are High, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

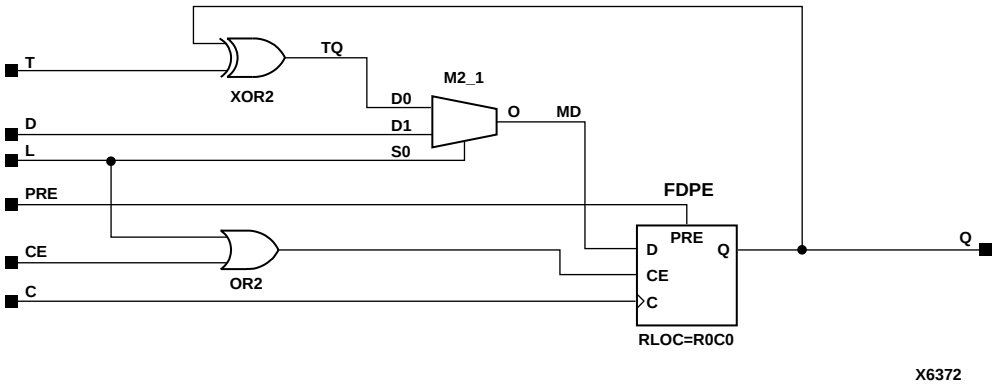
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously preset to output High, when power is applied.

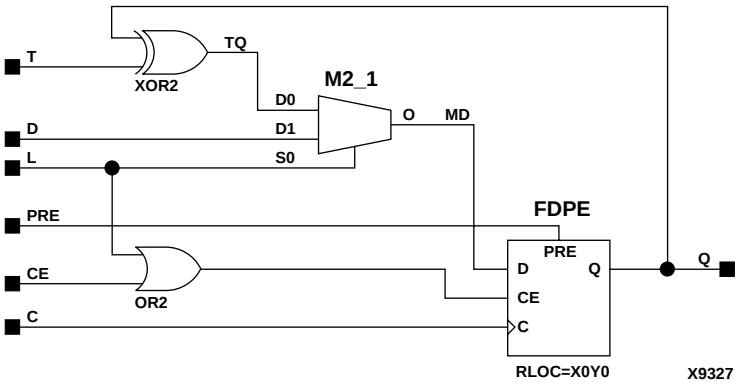
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

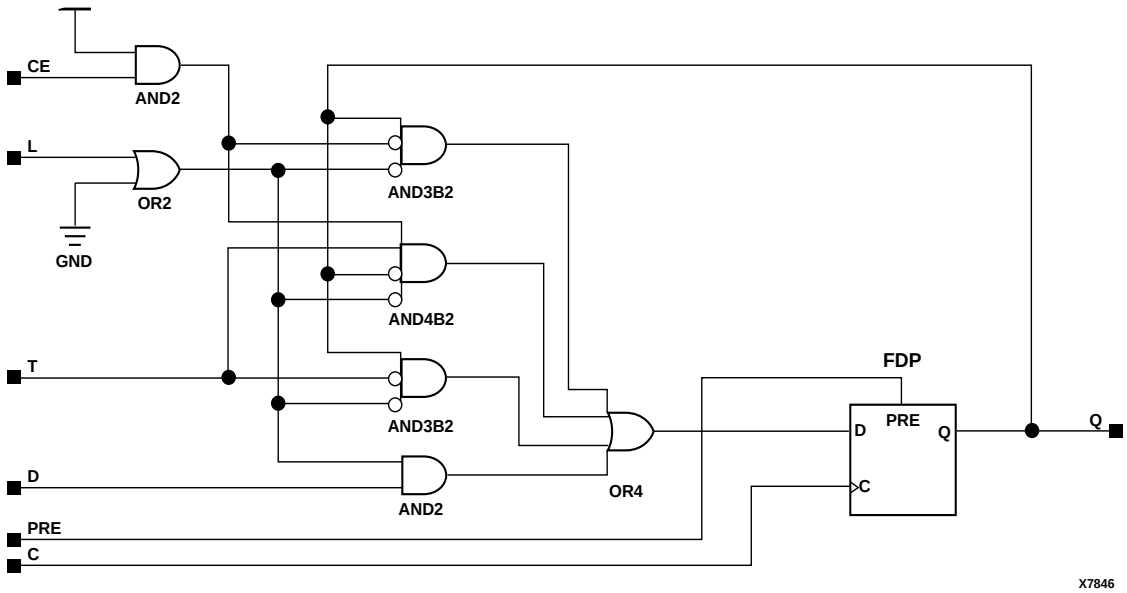
Inputs						Outputs
PRE	L	CE	T	D	C	Q
1	X	X	X	X	X	1
0	1	X	X	1	↑	1
0	1	X	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle



FTPLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTPLE Implementation Virtex-II, Virtex-II Pro



FTPLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftple is

begin

process (C, PRE)
begin
        if (PRE='1') then
                Q <= '1';
        elsif (C'event and C='1') then
                if (L='1') then
                        Q <= D;
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

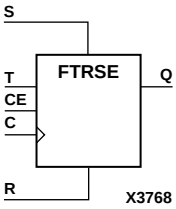
Verilog Inference Code

```
always @(posedge C or posedge PRE)
begin
        if (PRE)
                Q <= 1;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTRSE

Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTRSE is a toggle flip-flop with toggle and clock enable and synchronous reset and set. When the synchronous reset input (R) is High, it overrides all other inputs and the data output (Q) is reset Low. When the synchronous set input (S) is High and R is Low, clock enable input (CE) is overridden and output Q is set High. (Reset has precedence over Set.) When toggle enable input (T) and CE are High and R and S are Low, output Q toggles, or changes state, during the Low-to-High clock transition.

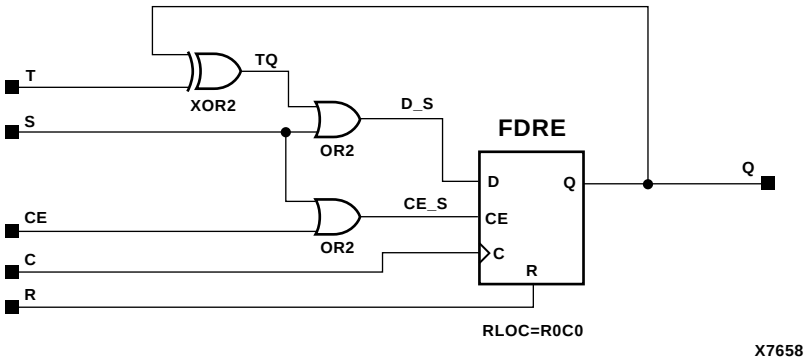
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

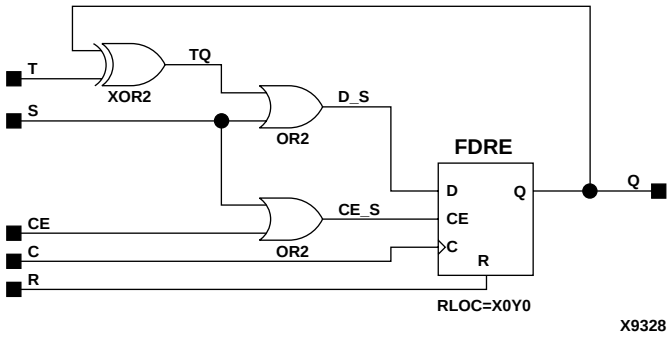
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

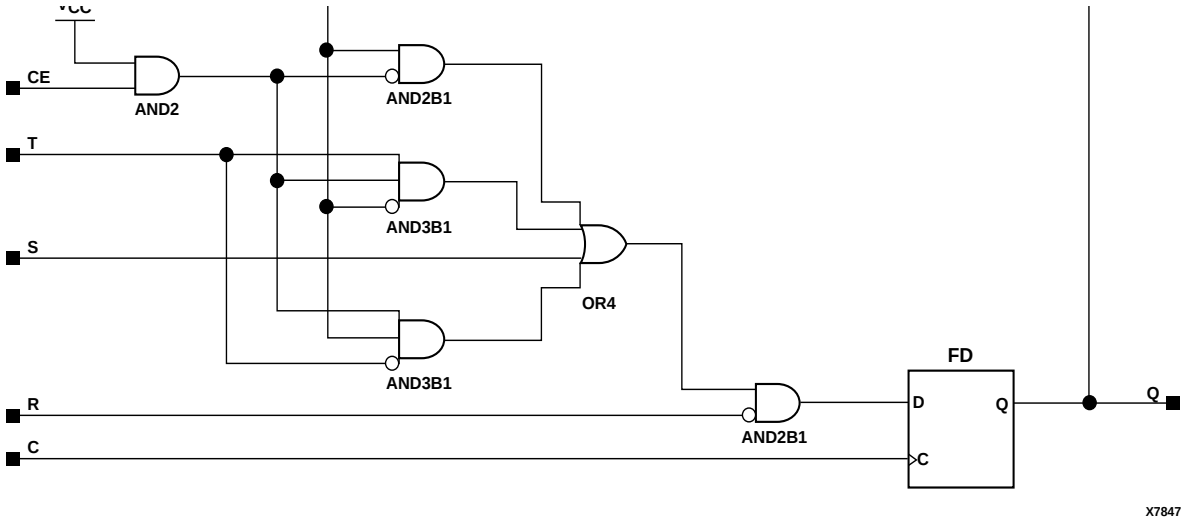
Inputs					Outputs
R	S	CE	T	C	Q
1	X	X	X	↑	0
0	1	X	X	↑	1
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle



FTRSE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTRSE Implementation Virtex-II, Virtex-II Pro



FTRSE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftrse is

begin

process (C)
begin
        if (C'event and C='1') then
                if (R='1') then
                        Q <= '0';
                elsif (S='1') then
                        Q <= '1';
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

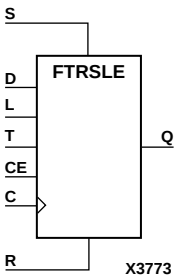
Verilog Inference Code

```
always @(posedge C)
begin
        if (R)
                Q <= 0;
        else if (S)
                Q <= 1;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTRSLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTRSLE is a toggle / loadable flip-flop with toggle and clock enable and synchronous reset and set. The synchronous reset input (R), when High, overrides all other inputs and resets the data output (Q) Low. (Reset has precedence over Set.) When R is Low and synchronous set input (S) is High, the clock enable input (CE) is overridden and output Q is set High. When R and S are Low and load enable input (L) is High, CE is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High clock transition. When R, S, and L are Low, CE is High and T is High, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

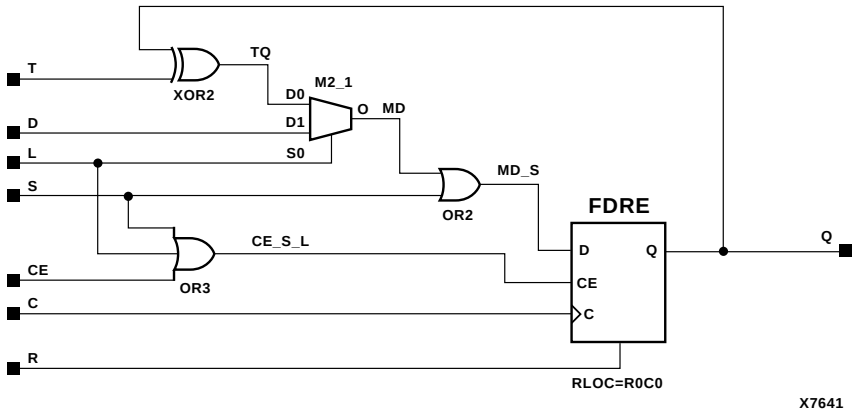
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

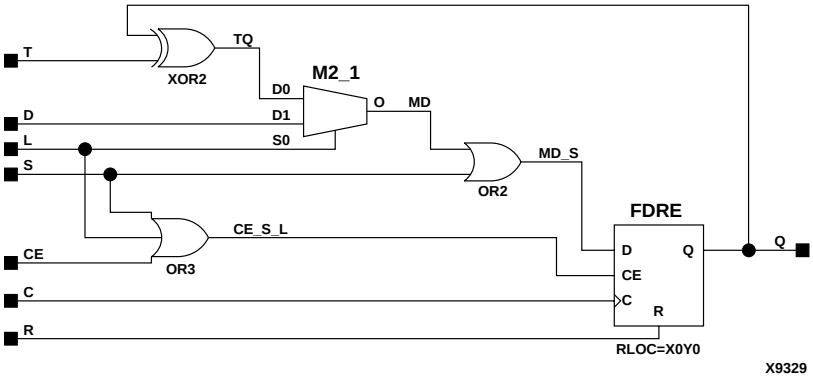
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

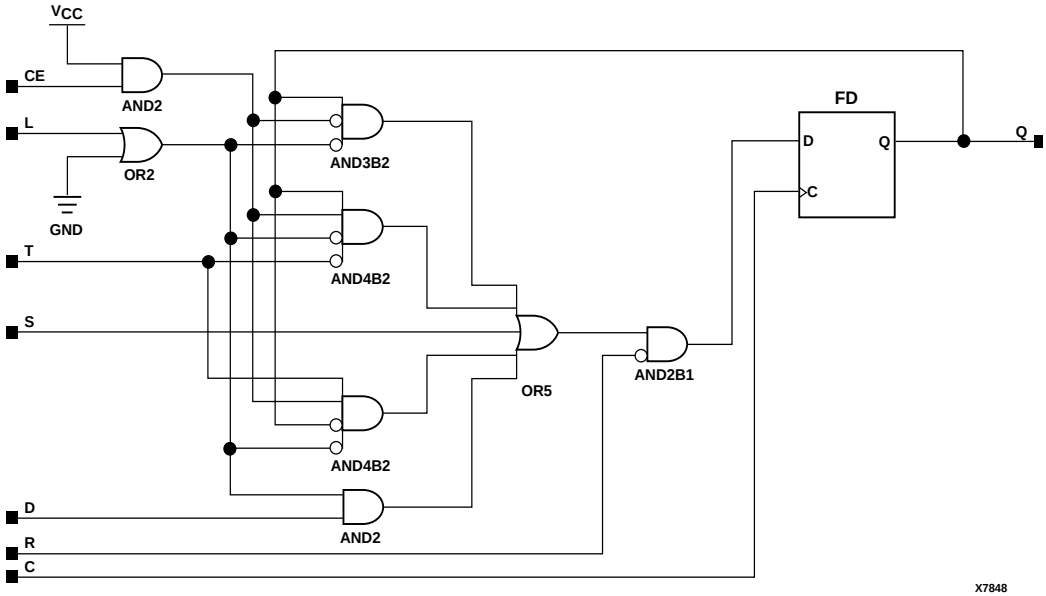
Inputs							Outputs
R	S	L	CE	T	D	C	Q
1	0	X	X	X	X	↑	0
0	1	X	X	X	X	↑	1
0	0	1	X	X	1	↑	1
0	0	1	X	X	0	↑	0
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	X	↑	Toggle



FTRSLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTRSLE Implementation Virtex-II, Virtex-II Pro



FTRSLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftrsle is

begin

process (C)
begin
        if (C’event and C=’1’) then
                if (R=’1’) then
                        Q <= ’0’;
                elsif (S=’1’) then
                        Q <= ’1’;
                elsif (L=’1’) then
                        Q <= D;
                elsif (CE=’1’) then
                        if (T=’1’) then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

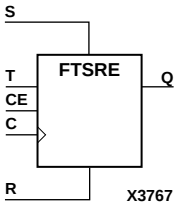
Verilog Inference Code

```
always @(posedge C)
begin
        if (R)
                Q <= 0;
        else if (S)
                Q <= 1;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTSRE

Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



FTSRE is a toggle flip-flop with toggle and clock enable and synchronous set and reset. The synchronous set input, when High, overrides all other inputs and sets data output (Q) High. (Set has precedence over Reset.) When synchronous reset input (R) is High and S is Low, clock enable input (CE) is overridden and output Q is reset Low. When toggle enable input (T) and CE are High and S and R are Low, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

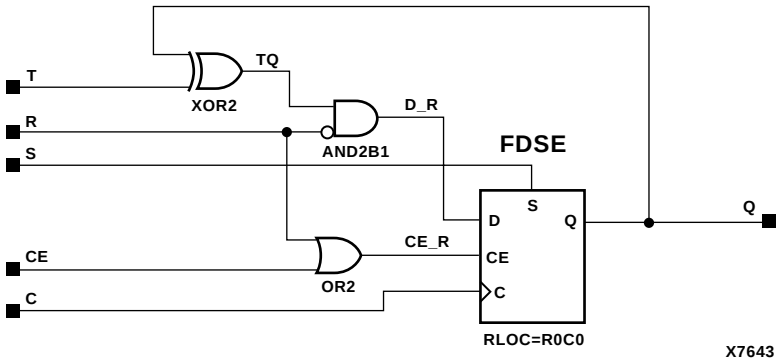
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

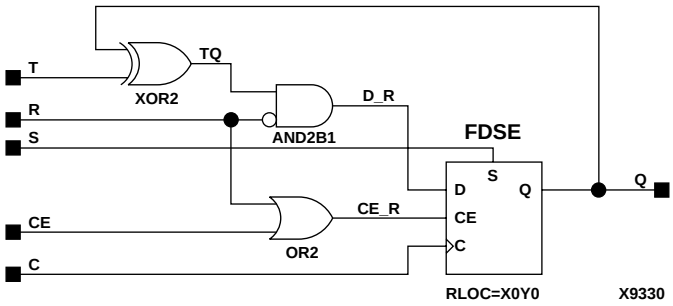
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FTSRE will set when GSR is active. For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is preset to active high when GSR is active.

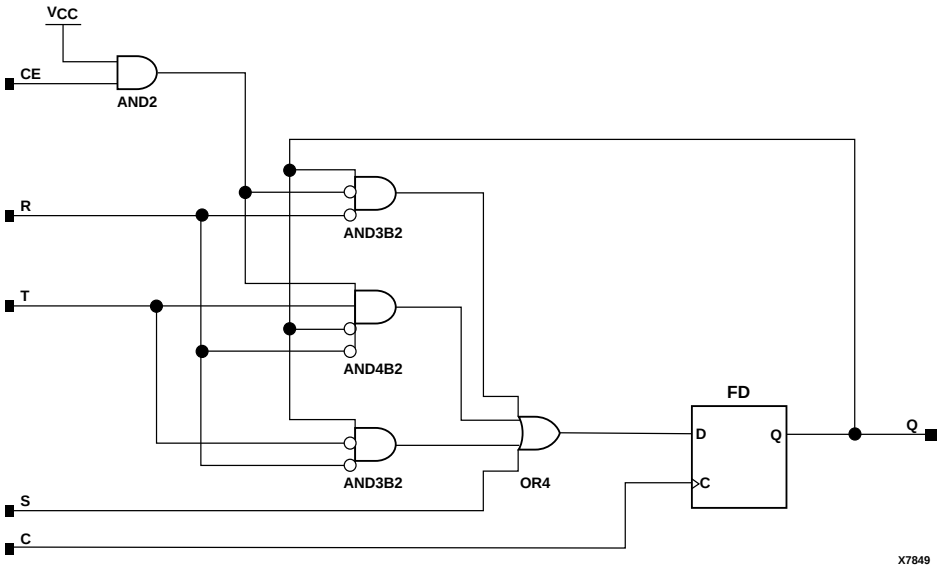
Inputs					Outputs
S	R	CE	T	C	Q
1	X	X	X	↑	1
0	1	X	X	↑	0
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle



FTSRE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTSRE Implementation Virtex-II, Virtex-II Pro



FTSRE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftsre is

begin

process (C)
begin
        if (C'event and C='1') then
                if (S='1') then
                        Q <= '1';
                elsif (R='1') then
                        Q <= '0';
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

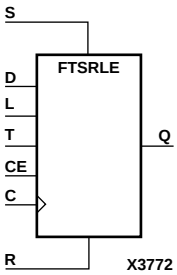
Verilog Inference Code

```
always @(posedge C)
begin
        if (S)
                Q <= 1;
        else if (R)
                Q <= 0;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

FTSRLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



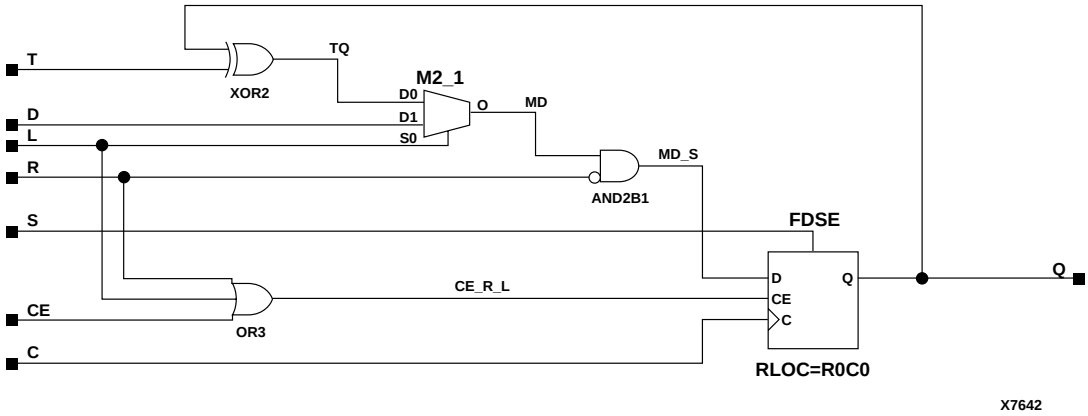
FTSRLE is a toggle/loadable flip-flop with toggle and clock enable and synchronous set and reset. The synchronous set input (S), when High, overrides all other inputs and sets data output (Q) High. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, clock enable input (CE) is overridden and output Q is reset Low. When load enable input (L) is High and S and R are Low, CE is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High clock transition. When the toggle enable input (T) and CE are High and S, R, and L are Low, output Q toggles, or changes state, during the Low-to- High clock transition. When CE is Low, clock transitions are ignored.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously preset when a High-level pulse is applied on the PRLD global net.

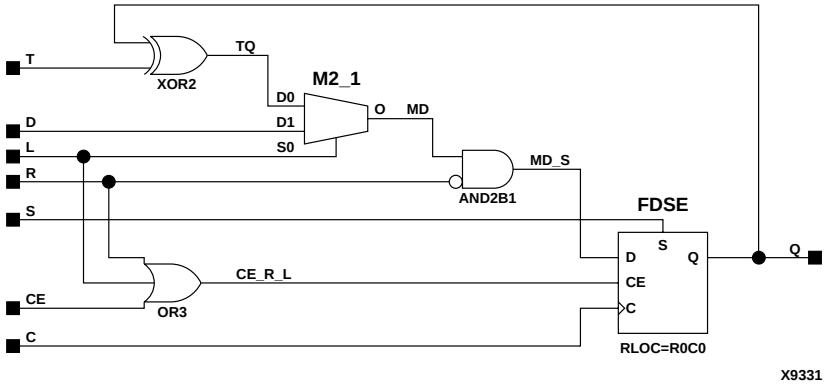
For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is asynchronously cleared, output Low, when global set/reset (GSR) is active.

The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FTSRLE will set when GSR is active. For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, the flip-flop is preset to active high when GSR is active.

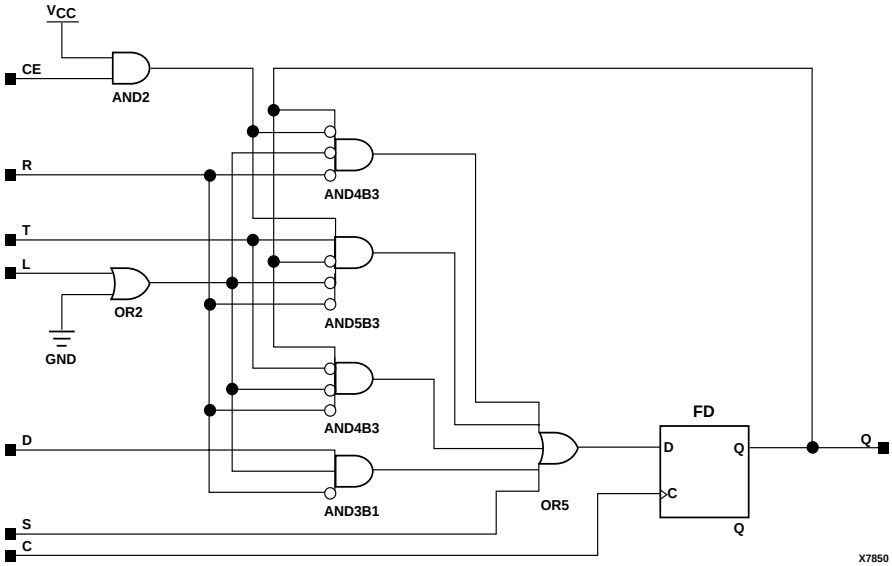
Inputs							Outputs
S	R	L	CE	T	D	C	Q
1	0	X	X	X	X	↑	1
0	1	X	X	X	X	↑	0
0	0	1	X	X	1	↑	1
0	0	1	X	X	0	↑	0
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	X	↑	Toggle



FTSRLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTSRLE Implementation Virtex-II, Virtex-II Pro



FTSRLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ftsrle is

begin

process (C)
begin
        if (C'event and C='1') then
                if (S='1') then
                        Q <= '1';
                elsif (R='1') then
                        Q <= '0';
                elsif (L='1') then
                        Q <= D;
                elsif (CE='1') then
                        if (T='1') then
                                Q <= not Q;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @(posedge C)
begin
        if (S)
                Q <= 1;
        else if (R)
                Q <= 0;
        else if (L)
                Q <= D;
        else if (CE)
                if (T)
                        Q <= !Q;
end
```

GND

Ground-Connection Signal Tag

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

X3858

The GND signal tag, or parameter, forces a net or input function to a Low logic level. A net tied to GND cannot have any other source.

When the logic-trimming software or fitter encounters a net or input function tied to GND, it removes any logic that is disabled by the GND signal. The GND signal is only implemented when the disabled logic cannot be removed.

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Inference Code:

```
gnd_signal <= '0';
```

Verilog Inference Code:

```
assign gnd_signal = 0;
```

VHDL Instantiation Template

```
-- Component Declaration for GND should be placed
-- after architecture statement but before begin keyword

component GND
   port (G  : out STD_ULOGIC);
end component;

-- Component Attribute specification for GND
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here


-- Component Instantiation for GND should be placed
-- in architecture after the begin keyword

GND_INSTANCE_NAME : GND
      port map  (G => user_G);
```

Verilog Instantiation Template

```
GND GND_instance_name (.G (user_G));
```

Commonly Used Constraints

None

GT_AURORA_*n*

Gigabit Transceiver for High-Speed I/O

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II Pro*	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A

*Supported for Virtex-II Pro but not for Virtex-II

This Xilinx protocol gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

You can also set attributes for the primitives. See Table 1-8 of the *Rocket I/O Transceiver User Guide* for a description of these attributes. See Table 1-9 of the *Rocket I/O Transceiver User Guide* for the default attribute values.

The following table lists the input and output ports for all values of *n*. For a description of each of the ports, see Table 1-7 in the *Rocket I/O Transceiver User Guide.*

Inputs	Outputs
BREFCLK	CHBONDDONE
BREFCLK2	CHBONDO [3:0]
CHBONDI [3:0]	CONFIGOUT
CONFIGENABLE	RXBUFSTATUS [1:0]
CONFIGIN	RXCHARISCOMMA [0:0] (GT_AURORA_1) RXCHARISCOMMA [1:0] (GT_AURORA_2) RXCHARISCOMMA [3:0] (GT_AURORA_4)
ENCHANSYNC	RXCHARISK [0:0] (GT_AURORA_1) RXCHARISK [1:0] (GT_AURORA_2) RXCHARISK [3:0] (GT_AURORA_4)
ENMCOMMAALIGN	RXCHECKINGCRC
ENPCOMMAALIGN	RXCLKCORCNT [2:0]
LOOPBACK [1:0]	RXCOMMADET
POWERDOWN	RXCRCERR
REFCLK	RXDATA [7:0] (GT_AURORA_1) RXDATA [15:0] (GT_AURORA_2) RXDATA [31:0] (GT_AURORA_4)
REFCLK2	RXDISPERR [0:0] (GT_AURORA_1) RXDISPERR [1:0] (GT_AURORA_2) RXDISPERR [3:0] (GT_AURORA_4
REFCLKSEL	RXLOSSOFSYNC [1:0]
RXN	RXNOTINTABLE [0:0] (GT_AURORA_1) RXNOTINTABLE [1:0] (GT_AURORA_2) RXNOTINTABLE [3:0] (GT_AURORA_4)
RXP	RXREALIGN
RXPOLARITY	RXRECCLK
RXRESET	RXRUNDISP [0:0] (GT_AURORA_1) RXRUNDISP [1:0] (GT_AURORA_2) RXRUNDISP [3:0] (GT_AURORA_4)

Inputs	Outputs
RXUSRCLK	TXBUFERR
RXUSRCLK2	TXKERR [0:0] (GT_AURORA_1) TXKERR [1:0] (GT_AURORA_2) TXKERR [3:0] (GT_AURORA_4)
TXBYPASS8B10B [0:0] (GT_AURORA_1) TXBYPASS8B10B [1:0] (GT_AURORA_2) TXBYPASS8B10B [3:0] (GT_AURORA_4)	TXN
TXCHARDISPMODE [0:0] (GT_AURORA_1) TXCHARDISPMODE [1:0] (GT_AURORA_2) TXCHARDISPMODE [3:0] (GT_AURORA_4)	TXP
TXCHARDISPVAL [0:0] (GT_AURORA_1) TXCHARDISPVAL [1:0] (GT_AURORA_2) TXCHARDISPVAL [3:0] (GT_AURORA_4)	TXRUNDISP [0:0] (GT_AURORA_1) TXRUNDISP [1:0] (GT_AURORA_2) TXRUNDISP [3:0] (GT_AURORA_4)
TXCHARISK [0:0] (GT_AURORA_1) TXCHARISK [1:0] (GT_AURORA_2) TXCHARISK [3:0] (GT_AURORA_4)	
TXDATA [7:0] (GT_AURORA_1) TXDATA [15:0](GT_AURORA_2) TXDATA [31:0](GT_AURORA_4)	
TXFORCECRCERR	
TXINHIBIT	
TXPOLARITY	
TXRESET	
TXUSRCLK	
TXUSRCLK2	

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

```
-- Component Declaration for GT_AURORA_n should be placed
-- after architecture statement but before begin keyword

component GT_AURORA_n
  -- synthesis translate_off
  generic (CHAN_BOND_MODE            : string := "OFF";
           CHAN_BOND_ONE_SHOT        : boolean:= FALSE;
           CLK_COR_INSERT_IDLE_FLAG  : boolean := FALSE;
           CLK_COR_KEEP_IDLE         : boolean := FALSE;
           CLK_COR_REPEAT_WAIT       : integer := 1;
           CRC_END_OF_PKT            : string := "K29_7";
           CRC_START_OF_PKT          : string := "K27_7";
           REF_CLK_V_SEL             : integer :=0;
           RX_CRC_USE                : boolean := FALSE;
           RX_LOSS_OF_SYNC_FSM       : boolean := FALSE;
           RX_LOS_INVALID_INCR       : integer := 1;
           RX_LOS_THRESHOLD          : integer := 4;
           SERDES_10B                : boolean := FALSE;
           TERMINATION_IMP           : integer := 50;
```

```
          TX_CRC_FORCE_VALUE        : bit_vector  := "11010110";
          TX_CRC_USE                : boolean := FALSE;
          TX_DIFF_CTRL              : integer := 500;
          TX_PREEMPHASIS            : integer := 0);
  -- synthesis translate_on
  port(CHBONDDONE    : out STD_ULOGIC;
       CHBONDO       : out STD_LOGIC_VECTOR (3 downto 0);
       CONFIGOUT     : out STD_ULOGIC;
       RXBUFSTATUS   : out STD_LOGIC_VECTOR (1 downto 0);
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
       RXCHARISK     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
       RXCHARISK     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
       RXCHARISK     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
       RXCHECKINGCRC : out STD_ULOGIC;
       RXCLKCORCNT   : out STD_LOGIC_VECTOR (2 downto 0);
       RXCOMMADET    : out STD_ULOGIC;
       RXCRCERR      : out STD_ULOGIC;
       RXDATA        : out STD_LOGIC_VECTOR (7 downto 0); -- for GT_AURORA_1
       RXDATA        : out STD_LOGIC_VECTOR (15 downto 0); -- for GT_AURORA_2
       RXDATA        : out STD_LOGIC_VECTOR (31 downto 0); -- for GT_AURORA_4
       RXDISPERR     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
       RXDISPERR     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
       RXDISPERR     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
       RXLOSSOFSYNC  : out STD_LOGIC_VECTOR (1 downto 0);
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
       RXREALIGN     : out STD_ULOGIC;
       RXRECCLK      : out STD_ULOGIC;
       RXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
       RXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
       RXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
       TXBUFERR      : out STD_ULOGIC;
       TXKERR        : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
       TXKERR        : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
       TXKERR        : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
       TXN           : out STD_ULOGIC;
       TXP           : out STD_ULOGIC;
       TXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
       TXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
       TXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
       BREFCLK       : in STD_ULOGIC;
       BREFCLK2      : in STD_ULOGIC;
       CHBONDI       : in STD_LOGIC_VECTOR (3 downto 0);
       CONFIGENABLE  : in STD_ULOGIC;
       CONFIGIN      : in STD_ULOGIC;
       ENCHANSYNC    : in STD_ULOGIC;
       ENMCOMMAALIGN : in STD_ULOGIC;
       ENPCOMMAALIGN : in STD_ULOGIC;
       LOOPBACK      : in STD_LOGIC_VECTOR (1 downto 0);
       POWERDOWN     : in STD_ULOGIC;
       REFCLK        : in STD_ULOGIC;
```

```
      REFCLK2       : in STD_ULOGIC;
      REFCLKSEL     : in STD_ULOGIC;
      RXN           : in STD_ULOGIC;
      RXP           : in STD_ULOGIC;
      RXPOLARITY    : in STD_ULOGIC;
      RXRESET       : in STD_ULOGIC;
      RXUSRCLK      : in STD_ULOGIC;
      RXUSRCLK2     : in STD_ULOGIC;
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
      TXCHARISK     : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_AURORA_1
      TXCHARISK     : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_AURORA_2
      TXCHARISK     : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_AURORA_4
      TXDATA        : in STD_LOGIC_VECTOR (7 downto 0); -- for GT_AURORA_1
      TXDATA        : in STD_LOGIC_VECTOR (15 downto 0); -- for GT_AURORA_2
      TXDATA        : in STD_LOGIC_VECTOR (31 downto 0); -- for GT_AURORA_4
      TXFORCECRCERR : in STD_ULOGIC;
      TXINHIBIT     : in STD_ULOGIC;
      TXPOLARITY    : in STD_ULOGIC;
      TXRESET       : in STD_ULOGIC;
      TXUSRCLK      : in STD_ULOGIC;
      TXUSRCLK2     : in STD_ULOGIC);
end component;

-- Component Attribute specification for GT_AURORA_n
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here


-- Component Instantiation for GT_AURORA_n should be placed
-- in architecture after the begin keyword

GT_AURORA_n_INSTANCE_NAME : GT_AURORA_n
  -- synthesis translate_off
  generic map(CHAN_BOND_MODE        => string_value,
              CHAN_BOND_ONE_SHOT        => boolean_value,
              CLK_COR_INSERT_IDLE_FLAG => boolean_value,
              CLK_COR_KEEP_IDLE         => boolean_value,
              CLK_COR_REPEAT_WAIT       => integer_value,
              CRC_END_OF_PKT            => string_value,
              CRC_START_OF_PKT          => string_value,
              REF_CLK_V_SEL             => integer_value,
              RX_CRC_USE                => boolean_value,
              RX_LOSS_OF_SYNC_FSM       => boolean_value,
              RX_LOS_INVALID_INCR       => integer_value,
```

```
            RX_LOS_THRESHOLD          => integer_value,
            SERDES_10B                => boolean_value,
            TERMINATION_IMP           => integer_value,
            TX_CRC_FORCE_VALUE        => bit_value,
            TX_CRC_USE                => boolean_value,
            TX_DIFF_CTRL              => integer_value,
            TX_PREEMPHASIS            => integer_value);
-- synthesis translate_on
  port map(CHBONDDONE      => user_CHBONDDONE,
           CHBONDO         => user_CHBONDO,
           CONFIGOUT       => user_CONFIGOUT,
           RXBUFSTATUS     => user_RXBUFSTATUS,
           RXCHARISCOMMA   => user_RXCHARISCOMMA,
           RXCHARISK       => user_RXCHARISK,
           RXCHECKINGCRC   => user_RXCHECKINGCRC,
           RXCLKCORCNT     => user_RXCLKCORCNT,
           RXCOMMADET      => user_RXCOMMADET,
           RXCRCERR        => user_RXCRCERR,
           RXDATA          => user_RXDATA,
           RXDISPERR       => user_RXDISPERR,
           RXLOSSOFSYNC    => user_RXLOSSOFSYNC,
           RXNOTINTABLE    => user_RXNOTINTABLE,
           RXREALIGN       => user_RXREALIGN,
           RXRECCLK        => user_RXRECCLK,
           RXRUNDISP       => user_RXRUNDISP,
           TXBUFERR        => user_TXBUFERR,
           TXKERR          => user_TXKERR,
           TXN             => user_TXN,
           TXP             => user_TXP,
           TXRUNDISP       => user_TXRUNDISP,
           BREFCLK         => user_BREFCLK,
           BREFCLK         => user_BREFCLK2,
           CHBONDI         => user_CHBONDI,
           CONFIGENABLE    => user_CONFIGENABLE,
           CONFIGIN        => user_CONFIGIN,
           ENCHANSYNC      => user_ENCHANSYNC,
           ENMCOMMAALIGN   => user_ENMCOMMAALIGN,
           ENPCOMMAALIGN   => user_ENPCOMMAALIGN,
           LOOPBACK        => user_LOOPBACK,
           POWERDOWN       => user_POWERDOWN,
           REFCLK          => user_REFCLK,
           REFCLK2         => user_REFCLK2,
           REFCLKSEL       => user_REFCLKSEL,
           RXN             => user_RXN,
           RXP             => user_RXP,
           RXPOLARITY      => user_RXPOLARITY,
           RXRESET         => user_RXRESET,
           RXUSRCLK        => user_RXUSRCLK,
           RXUSRCLK2       => user_RXUSRCLK2,
           TXBYPASS8B10B   => user_TXBYPASS8B10B,
           TXCHARDISPMODE  => user_TXCHARDISPMODE,
           TXCHARDISPVAL   => user_TXCHARDISPVAL,
           TXCHARISK       => user_TXCHARISK,
           TXDATA          => user_TXDATA,
```

```
              TXFORCECRCERR   => user_TXFORCECRCERR,
              TXINHIBIT       => user_TXINHIBIT,
              TXPOLARITY      => user_TXPOLARITY,
              TXRESET         => user_TXRESET,
              TXUSRCLK        => user_TXUSRCLK,
              TXUSRCLK2       => user_TXUSRCLK2);
```

Verilog Instantiation Template

```
GT_AURORA_n GT_AURORA_n_instance_name (.CHBONDDONE (user_CHBONDDONE),
                                       .CHBONDO (user_CHBONDO),
                                       .CONFIGOUT (user_CONFIGOUT),
                                       .RXBUFSTATUS (user_RXBUFSTATUS),
                                       .RXCHARISCOMMA (user_RXCHARISCOMMA),
                                       .RXCHARISK (user_RXCHARISK),
                                       .RXCHECKINGCRC (user_RXCHECKINGCRC),
                                       .RXCLKCORCNT (user_RXCLKCORCNT),
                                       .RXCOMMADET (user_RXCOMMADET),
                                       .RXCRCERR (user_RXCRCERR),
                                       .RXDATA (user_RXDATA),
                                       .RXDISPERR (user_RXDISPERR),
                                       .RXLOSSOFSYNC (user_RXLOSSOFSYNC),
                                       .RXNOTINTABLE (user_RXNOTINTABLE),
                                       .RXREALIGN (user_RXREALIGN),
                                       .RXRECCLK (user_RXRECCLK),
                                       .RXRUNDISP (user_RXRUNDISP),
                                       .TXBUFERR (user_TXBUFERR),
                                       .TXKERR (user_TXKERR),
                                       .TXN(user_TXN),
                                       .TXP (user_TXP),
                                       .TXRUNDISP (user_TXRUNDISP),
                                       .CHBONDI (user_CHBONDI),
                                       .CONFIGENABLE (user_CONFIGENABLE),
                                       .CONFIGIN (user_CONFIGIN),
                                       .ENCHANSYNC (user_ENCHANSYNC),
                                       .ENMCOMMAALIGN (user_ENMCOMMAALIGN),
                                       .ENPCOMMAALIGN (user_ENPCOMMAALIGN),
                                       .LOOPBACK (user_LOOPBACK),
                                       .POWERDOWN (user_POWERDOWN),
                                       .REFCLK (user_REFCLK),
                                       .REFCLK2 (user_REFCLK2),
                                       .REFCLKSEL (user_REFCLKSEL),
                                       .RXN (user_RXN),
                                       .RXP (user_RXP),
                                       .RXPOLARITY (user_RXPOLARITY),
                                       .RXRESET (user_RXRESET),
                                       .RXUSRCLK (user_RXUSRCLK),
                                       .RXUSRCLK2 (user_RXUSRCLK2),
                                       .TXBYPASS8B10B (user_TXBYPASS8B10B),
                                       .TXCHARDISPMODE => user_TXCHARDISPMODE),
                                       .TXCHARDISPVAL (user_TXCHARDISPVAL),
                                       .TXCHARISK (user_TXCHARISK),
                                       .TXDATA (user_TXDATA),
                                       .TXFORCECRCERR (user_TXFORCECRCERR),
```

```
                              .TXINHIBIT (user_TXINHIBIT),
                              .TXPOLARITY (user_TXPOLARITY),
                              .TXRESET (user_TXRESET),
                              .TXUSRCLK (user_TXUSRCLK),
                              .TXUSRCLK2 (user_TXUSRCLK2));

defparam user_instance_name.CHAN_BOND_MODE = "OFF";
defparam user_instance_name.CHAN_BOND_ONE_SHOT = "FALSE";
defparam user_instance_name.CLK_COR_INSERT_IDLE_FLAG = "FALSE";
defparam user_instance_name.CLK_COR_KEEP_IDLE = "FALSE";
defparam user_instance_name.CLK_COR_REPEAT_WAIT = 1;
defparam user_instance_name.CRC_END_OF_PKT = "K29_7";
defparam user_instance_name.CRC_START_OF_PKT = "K27_7";
defparam user_instance_name.RX_CRC_USE = "FALSE";
defparam user_instance_name.RX_LOSS_OF_SYNC_FSM = "FALSE";
defparam user_instance_name.RX_LOS_INVALID_INCR = 1;
defparam user_instance_name.RX_LOS_THRESHOLD = 4;
defparam user_instance_name.SERDES_10B = "FALSE";
defparam user_instance_name.TERMINATION_IMP = 50;
defparam user_instance_name.TX_CRC_FORCE_VALUE = bit_value;
defparam user_instance_name.TX_CRC_USE = "FALSE";
defparam user_instance_name.TX_DIFF_CTRL = 500;
defparam user_instance_name.TX_PREEMPHASIS = 0;
```

Commonly Used Constraints

None

GT_CUSTOM

Gigabit Transceiver for High-Speed I/O

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II Pro*	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A

*Supported for Virtex-II Pro but not for Virtex-II

This gigabit transceiver is fully customizable. You can set attributes for the primitives. See Table 1-8 of the *Rocket I/O Transceiver User Guide* for a description of these attributes. See Table 1-9 of the *Rocket I/O Transceiver User Guide* for the default attribute values.

The following table lists the input and output ports. For a description of each of the ports, see Table 1-7 in the *Rocket I/O Transceiver User Guide*.

Inputs	Outputs
BREFCLK	
BREFCLK2	
CHBONDI [3:0]	CHBONDDONE
CONFIGENABLE	CHBONDO [3:0]
CONFIGIN	CONFIGOUT
ENCHANSYNC	RXBUFSTATUS [1:0]
ENMCOMMAALIGN	RXCHARISCOMMA [3:0]
ENPCOMMAALIGN	RXCHARISK [3:0]
LOOPBACK [1:0]	RXCHECKINGCRC
POWERDOWN	RXCLKCORCNT [2:0]
REFCLK	RXCOMMADET
REFCLK2	RXCRCERR
REFCLKSEL	RXDATA [31:0]
RXN	RXDISPERR [3:0]
RXP	RXLOSSOFSYNC [1:0]
RXPOLARITY	RXNOTINTABLE [3:0]
RXRESET	RXREALIGN
RXUSRCLK	RXRECCLK
RXUSRCLK2	RXRUNDISP [3:0]
TXBYPASS8B10B [3:0]	TXBUFERR
TXCHARDISPMODE [3:0]	TXKERR [3:0]
TXCHARDISPVAL [3:0]	TXN
TXCHARISK [3:0]	TXP
TXDATA [31:0]	TXRUNDISP [3:0]
TXFORCECRCERR	
TXINHIBIT	
TXPOLARITY	

Inputs	Outputs
TXRESET	
TXUSRCLK	
TXUSRCLK2	

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

```
-- Component Declaration for GT_CUSTOM should be placed
-- after architecture statement but before begin keyword

component GT_CUSTOM
  -- synthesis translate_off
  generic (ALIGN_COMMA_MSB        : boolean := FALSE;
           CHAN_BOND_LIMIT        : integer := 16;
           CHAN_BOND_MODE         : string := "OFF";
           CHAN_BOND_OFFSET       : integer := 8;
           CHAN_BOND_ONE_SHOT     : boolean : = FALSE;
           CHAN_BOND_SEQ_1_1      : bit_vector (10 downto 0):= "0000000000";
           CHAN_BOND_SEQ_1_2      : bit_vector (10 downto 0):= "0000000000";
           CHAN_BOND_SEQ_1_3      : bit_vector (10 downto 0):= "0000000000";
           CHAN_BOND_SEQ_1_4      : bit_vector (10 downto 0):= "0000000000";
           CHAN_BOND_SEQ_2_1      : bit_vector (10 downto 0):= "0000000000";
           CHAN_BOND_SEQ_2_2      : bit_vector (10 downto 0):= "0000000000";
           CHAN_BOND_SEQ_2_3      : bit_vector (10 downto 0):= "0000000000";
           CHAN_BOND_SEQ_2_4      : bit_vector (10 downto 0):= "0000000000";
           CHAN_BOND_SEQ_2_USE    : boolean := FALSE;
           CHAN_BOND_SEQ_LEN      : integer := 1;
           CHAN_BOND_WAIT         : integer := 8;
           CLK_CORRECT_USE        : boolean := TRUE;
           CLK_COR_INSERT_IDLE_FLAG  : boolean := FALSE;
           CLK_COR_KEEP_IDLE      : boolean := FALSE;
           CLK_COR_REPEAT_WAIT    : integer := 1;
           CLK_COR_SEQ_1_1        : bit_vector (10 downto 0):= "0000000000";
           CLK_COR_SEQ_1_2        : bit_vector (10 downto 0):= "0000000000";
           CLK_COR_SEQ_1_3        : bit_vector (10 downto 0):= "0000000000";
           CLK_COR_SEQ_1_4        : bit_vector (10 downto 0):= "0000000000";
           CLK_COR_SEQ_2_1        : bit_vector (10 downto 0):= "0000000000";
           CLK_COR_SEQ_2_2        : bit_vector (10 downto 0):= "0000000000";
           CLK_COR_SEQ_2_3        : bit_vector (10 downto 0):= "0000000000";
           CLK_COR_SEQ_2_4        : bit_vector (10 downto 0):= "0000000000";
           CLK_COR_SEQ_2_USE      : string := "FALSE";
           CLK_COR_SEQ_LEN        : real:= 1;
           COMMA_10B_MASK         : bit_vector := X"0";
           CRC_END_OF_PKT         : string := "K29_7";
           CRC_FORMAT             : string := "USER_MODE";
           CRC_START_OF_PKT       : string := "K27_7";
           DEC_MCOMMA_DETECT      : string := "TRUE";
           DEC_PCOMMA_DETECT      : string := "TRUE";
           DEC_VALID_COMMA_ONLY   : string := "TRUE";
```

```
        MCOMMA_10B_VALUE        : bit_vector := X"0";
        MCOMMA_DETECT           : string := "TRUE";
        PCOMMA_10B_VALUE        : bit_vector := X"0";
        PCOMMA_DETECT           : string := "TRUE";
        RX_BUFFER_USE           : string := "TRUE";
        RX_CRC_USE              : string := "FALSE";
        RX_DATA_WIDTH           : real: = 2;
        RX_DECODE_USE           : string := "TRUE";
        RX_LOSS_OF_SYNC_FSM     : string := "TRUE";
        RX_LOS_INVALID_INCR     : real:= 1;
        RX_LOS_THRESHOLD        : real:= 4;
        SERDES_10B              : string := "FALSE";
        TERMINATION_IMP         : real:= 50;
        TX_BUFFER_USE           : string := "TRUE";
        TX_CRC_FORCE_VALUE      : bit_vector := X"0";
        TX_CRC_USE              : string := "FALSE";
        TX_DATA_WIDTH           : real:= 2;
        TX_DIFF_CTRL            : real:= 500;
        TX_PREEMPHASIS          : real:= 0);
-- synthesis translate_on
port(CHBONDDONE    : out STD_ULOGIC;
     CHBONDO       : out STD_LOGIC_VECTOR (3 downto 0);
     CONFIGOUT     : out STD_ULOGIC;
     RXBUFSTATUS   : out STD_LOGIC_VECTOR (1 downto 0);
     RXCHARISCOMMA : out STD_LOGIC_VECTOR (3 downto 0);
     RXCHARISK     : out STD_LOGIC_VECTOR (3 downto 0);
     RXCHECKINGCRC : out STD_ULOGIC;
     RXCLKCORCNT   : out STD_LOGIC_VECTOR (2 downto 0);
     RXCOMMADET    : out STD_ULOGIC;
     RXCRCERR      : out STD_ULOGIC;
     RXDATA        : out STD_LOGIC_VECTOR (31 downto 0);
     RXDISPERR     : out STD_LOGIC_VECTOR (3 downto 0);
     RXLOSSOFSYNC  : out STD_LOGIC_VECTOR (1 downto 0);
     RXNOTINTABLE  : out STD_LOGIC_VECTOR (3 downto 0);
     RXREALIGN     : out STD_ULOGIC;
     RXRECCLK      : out STD_ULOGIC;
     RXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0);
     TXBUFERR      : out STD_ULOGIC;
     TXKERR        : out STD_LOGIC_VECTOR (3 downto 0);
     TXN           : out STD_ULOGIC;
     TXP           : out STD_ULOGIC;
     TXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0);
     CHBONDI       : in STD_LOGIC_VECTOR (3 downto 0);
     CONFIGENABLE  : in STD_ULOGIC;
     CONFIGIN      : in STD_ULOGIC;
     ENCHANSYNC    : in STD_ULOGIC;
     ENMCOMMAALIGN : in STD_ULOGIC;
     ENPCOMMAALIGN : in STD_ULOGIC;
     LOOPBACK      : in STD_LOGIC_VECTOR (1 downto 0);
     POWERDOWN     : in STD_ULOGIC;
     REFCLK        : in STD_ULOGIC;
     REFCLK2       : in STD_ULOGIC;
     REFCLKSEL     : in STD_ULOGIC;
     RXN           : in STD_ULOGIC;
```

```
       RXP           : in STD_ULOGIC;
       RXPOLARITY    : in STD_ULOGIC;
       RXRESET       : in STD_ULOGIC;
       RXUSRCLK      : in STD_ULOGIC;
       RXUSRCLK2     : in STD_ULOGIC;
       TXBYPASS8B10B : in STD_LOGIC_VECTOR (3 downto 0);
       TXCHARDISPMODE: in STD_LOGIC_VECTOR (3 downto 0);
       TXCHARDISPVAL : in STD_LOGIC_VECTOR (3 downto 0);
       TXCHARISK     : in STD_LOGIC_VECTOR (3 downto 0);
       TXDATA        : in STD_LOGIC_VECTOR (31 downto 0);
       TXFORCECRCERR : in STD_ULOGIC;
       TXINHIBIT     : in STD_ULOGIC;
       TXPOLARITY    : in STD_ULOGIC;
       TXRESET       : in STD_ULOGIC;
       TXUSRCLK      : in STD_ULOGIC;
       TXUSRCLK2     : in STD_ULOGIC);
end component;

-- Component Attribute specification for GT_CUSTOM
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here


-- Component Instantiation for GT_CUSTOM should be placed
-- in architecture after the begin keyword

GT_CUSTOM_INSTANCE_NAME :GT_CUSTOM
  -- synthesis translate_off
  generic map(ALIGN_COMMA_MSB     => string_value,
          CHAN_BOND_LIMIT         => real_value,
          CHAN_BOND_MODE          => string_value,
          CHAN_BOND_OFFSET        => real_value,
          CHAN_BOND_ONE_SHOT      => string_value,
          CHAN_BOND_SEQ_1_1       => bit_value,
          CHAN_BOND_SEQ_1_2       => bit_value,
          CHAN_BOND_SEQ_1_3       => bit_value,
          CHAN_BOND_SEQ_1_4       => bit_value,
          CHAN_BOND_SEQ_2_1       => bit_value,
          CHAN_BOND_SEQ_2_2       => bit_value,
          CHAN_BOND_SEQ_2_3       => bit_value,
          CHAN_BOND_SEQ_2_4       => bit_value,
          CHAN_BOND_SEQ_2_USE     => string_value,
          CHAN_BOND_SEQ_LEN       => real_value,
          CHAN_BOND_WAIT          => real_value,
          CLK_CORRECT_USE         => string_value,
          CLK_COR_INSERT_IDLE_FLAG  => string_value,
          CLK_COR_KEEP_IDLE       => string_value,
          CLK_COR_REPEAT_WAIT     => real_value,
          CLK_COR_SEQ_1_1         => bit_value,
          CLK_COR_SEQ_1_2         => bit_value,
          CLK_COR_SEQ_1_3         => bit_value,
          CLK_COR_SEQ_1_4         => bit_value,
```

```
        CLK_COR_SEQ_2_1         => bit_value,
        CLK_COR_SEQ_2_2         => bit_value,
        CLK_COR_SEQ_2_3         => bit_value,
        CLK_COR_SEQ_2_4         => bit_value,
        CLK_COR_SEQ_2_USE       => string_value,
        CLK_COR_SEQ_LEN         => real_value,
        COMMA_10B_MASK          => bit_value,
        CRC_END_OF_PKT          => string_value,
        CRC_FORMAT              => string_value,
        CRC_START_OF_PKT        => string_value,
        DEC_MCOMMA_DETECT       => string_value,
        DEC_PCOMMA_DETECT       => string_value,
        DEC_VALID_COMMA_ONLY    => string_value,
        MCOMMA_10B_VALUE        => bit_value,
        MCOMMA_DETECT           => string_value,
        PCOMMA_10B_VALUE        => bit_value,
        PCOMMA_DETECT           => string_value,
        RX_BUFFER_USE           => string_value,
        RX_CRC_USE              => string_value,
        RX_DATA_WIDTH           => real_value,
        RX_DECODE_USE           => string_value,
        RX_LOSS_OF_SYNC_FSM     => string_value,
        RX_LOS_INVALID_INCR     => real_value,
        RX_LOS_THRESHOLD        => real_value,
        SERDES_10B              => string_value,
        TERMINATION_IMP         => real_value,
        TX_BUFFER_USE           => string_value,
        TX_CRC_FORCE_VALUE      => bit_value,
        TX_CRC_USE              => string_value,
        TX_DATA_WIDTH           => real_value,
        TX_DIFF_CTRL            => real_value,
        TX_PREEMPHASIS          => real_value);
-- synthesis translate_on
  port map(CHBONDDONE      => user_CHBONDDONE,
           CHBONDO         => user_CHBONDO,
           CONFIGOUT       => user_CONFIGOUT,
           RXBUFSTATUS     => user_RXBUFSTATUS,
           RXCHARISCOMMA   => user_RXCHARISCOMMA,
           RXCHARISK       => user_RXCHARISK,
           RXCHECKINGCRC   => user_RXCHECKINGCRC,
           RXCLKCORCNT     => user_RXCLKCORCNT,
           RXCOMMADET      => user_RXCOMMADET,
           RXCRCERR        => user_RXCRCERR,
           RXDATA          => user_RXDATA,
           RXDISPERR       => user_RXDISPERR,
           RXLOSSOFSYNC    => user_RXLOSSOFSYNC,
           RXNOTINTABLE    => user_RXNOTINTABLE,
           RXREALIGN       => user_RXREALIGN,
           RXRECCLK        => user_RXRECCLK,
           RXRUNDISP       => user_RXRUNDISP,
           TXBUFERR        => user_TXBUFERR,
           TXKERR          => user_TXKERR,
           TXN             => user_TXN,
           TXP             => user_TXP,
```

```
               TXRUNDISP       => user_TXRUNDISP,
               CHBONDI         => user_CHBONDI,
               CONFIGENABLE    => user_CONFIGENABLE,
               CONFIGIN        => user_CONFIGIN,
               ENCHANSYNC      => user_ENCHANSYNC,
               ENMCOMMAALIGN   => user_ENMCOMMAALIGN,
               ENPCOMMAALIGN   => user_ENPCOMMAALIGN,
               LOOPBACK        => user_LOOPBACK,
               POWERDOWN       => user_POWERDOWN,
               REFCLK          => user_REFCLK,
               REFCLK2         => user_REFCLK2,
               REFCLKSEL       => user_REFCLKSEL,
               RXN             => user_RXN,
               RXP             => user_RXP,
               RXPOLARITY      => user_RXPOLARITY,
               RXRESET         => user_RXRESET,
               RXUSRCLK        => user_RXUSRCLK,
               RXUSRCLK2       => user_RXUSRCLK2,
               TXBYPASS8B10B   => user_TXBYPASS8B10B,
               TXCHARDISPMODE  => user_TXCHARDISPMODE,
               TXCHARDISPVAL   => user_TXCHARDISPVAL,
               TXCHARISK       => user_TXCHARISK,
               TXDATA          => user_TXDATA,
               TXFORCECRCERR   => user_TXFORCECRCERR,
               TXINHIBIT       => user_TXINHIBIT,
               TXPOLARITY      => user_TXPOLARITY,
               TXRESET         => user_TXRESET,
               TXUSRCLK        => user_TXUSRCLK,
               TXUSRCLK2       => user_TXUSRCLK2);
```

Verilog Instantiation Template

```
GT_CUSTOM GT_CUSTOM_instance_name (.CHBONDDONE (user_CHBONDDONE),
                                   .CHBONDO (user_CHBONDO),
                                   .CONFIGOUT (user_CONFIGOUT),
                                   .RXBUFSTATUS (user_RXBUFSTATUS),
                                   .RXCHARISCOMMA (user_RXCHARISCOMMA),
                                   .RXCHARISK (user_RXCHARISK),
                                   .RXCHECKINGCRC (user_RXCHECKINGCRC),
                                   .RXCLKCORCNT (user_RXCLKCORCNT),
                                   .RXCOMMADET (user_RXCOMMADET),
                                   .RXCRCERR (user_RXCRCERR),
                                   .RXDATA (user_RXDATA),
                                   .RXDISPERR (user_RXDISPERR),
                                   .RXLOSSOFSYNC (user_RXLOSSOFSYNC),
                                   .RXNOTINTABLE (user_RXNOTINTABLE),
                                   .RXREALIGN (user_RXREALIGN),
                                   .RXRECCLK (user_RXRECCLK),
                                   .RXRUNDISP (user_RXRUNDISP),
                                   .TXBUFERR (user_TXBUFERR),
                                   .TXKERR (user_TXKERR),
                                   .TXN(user_TXN),
                                   .TXP (user_TXP),
                                   .TXRUNDISP (user_TXRUNDISP),
```

```
                                .CHBONDI (user_CHBONDI),
                                 .CONFIGENABLE (user_CONFIGENABLE),
                                 .CONFIGIN (user_CONFIGIN),
                                 .ENCHANSYNC (user_ENCHANSYNC),
                                 .ENMCOMMAALIGN (user_ENMCOMMAALIGN),
                                 .ENPCOMMAALIGN (user_ENPCOMMAALIGN),
                                 .LOOPBACK (user_LOOPBACK),
                                 .POWERDOWN (user_POWERDOWN),
                                 .REFCLK (user_REFCLK),
                                 .REFCLK2 (user_REFCLK2),
                                 .REFCLKSEL (user_REFCLKSEL),
                                 .RXN (user_RXN),
                                 .RXP (user_RXP),
                                 .RXPOLARITY (user_RXPOLARITY),
                                 .RXRESET (user_RXRESET),
                                 .RXUSRCLK (user_RXUSRCLK),
                                 .RXUSRCLK2 (user_RXUSRCLK2),
                                 .TXBYPASS8B10B (user_TXBYPASS8B10B),
                                 .TXCHARDISPMODE  => user_TXCHARDISPMODE),
                                 .TXCHARDISPVAL (user_TXCHARDISPVAL),
                                 .TXCHARISK (user_TXCHARISK),
                                 .TXDATA (user_TXDATA),
                                 .TXFORCECRCERR (user_TXFORCECRCERR),
                                 .TXINHIBIT (user_TXINHIBIT),
                                 .TXPOLARITY (user_TXPOLARITY),
                                 .TXRESET (user_TXRESET),
                                 .TXUSRCLK (user_TXUSRCLK),
                                 .TXUSRCLK2 (user_TXUSRCLK2));

        defparam user_instance_name.ALIGN_COMMA_MSB = "FALSE";
        defparam user_instance_name.CHAN_BOND_LIMIT = 16;
        defparam user_instance_name.CHAN_BOND_MODE  : string := "OFF";
        defparam user_instance_name.CHAN_BOND_OFFSET  : real:= 8;
        defparam user_instance_name.CHAN_BOND_ONE_SHOT  : string : = "FALSE";
        defparam user_instance_name.CHAN_BOND_SEQ_1_1 = 11'b00000000000
        defparam user_instance_name.CHAN_BOND_SEQ_1_2 = 11'b00000000000;
        defparam user_instance_name.CHAN_BOND_SEQ_1_3 = 11'b00000000000;
        defparam user_instance_name.CHAN_BOND_SEQ_1_4 = 11'b00000000000;
        defparam user_instance_name.CHAN_BOND_SEQ_2_1 = 11'b00000000000;
        defparam user_instance_name.CHAN_BOND_SEQ_2_2 = 11'b00000000000;
        defparam user_instance_name.CHAN_BOND_SEQ_2_3 = 11'b00000000000;
        defparam user_instance_name.CHAN_BOND_SEQ_2_4 = 11'b00000000000;
        defparam user_instance_name.CHAN_BOND_SEQ_2_USE = "FALSE";
        defparam user_instance_name.CHAN_BOND_SEQ_LEN = 1;
        defparam user_instance_name.CHAN_BOND_WAIT = 8;
        defparam user_instance_name.CLK_CORRECT_USE = "TRUE";
        defparam user_instance_name.CLK_COR_INSERT_IDLE_FLAG = "FALSE";
        defparam user_instance_name.CLK_COR_KEEP_IDLE = "FALSE";
        defparam user_instance_name.CLK_COR_REPEAT_WAIT = 1;
        defparam user_instance_name.CLK_COR_SEQ_1_1 = 11'b00000000000;
        defparam user_instance_name.CLK_COR_SEQ_1_2 = 11'b00000000000;
        defparam user_instance_name.CLK_COR_SEQ_1_3 = 11'b00000000000;
        defparam user_instance_name.CLK_COR_SEQ_1_4 = 11'b00000000000;
        defparam user_instance_name.CLK_COR_SEQ_2_1 = 11'b00000000000;
```

```
defparam user_instance_name.CLK_COR_SEQ_2_2 = 11'b00000000000;
defparam user_instance_name.CLK_COR_SEQ_2_3 = 11'b00000000000;
defparam user_instance_name.CLK_COR_SEQ_2_4 = 11'b00000000000;
defparam user_instance_name.CLK_COR_SEQ_2_USE = "FALSE";
defparam user_instance_name.CLK_COR_SEQ_LEN = 1;
defparam user_instance_name.COMMA_10B_MASK = 10'b1111111000;
defparam user_instance_name.CRC_END_OF_PKT = "K29_7";
defparam user_instance_name.CRC_FORMAT = "USER_MODE";
defparam user_instance_name.CRC_START_OF_PKT = "K27_7";
defparam user_instance_name.DEC_MCOMMA_DETECT = "TRUE";
defparam user_instance_name.DEC_PCOMMA_DETECT = "TRUE";
defparam user_instance_name.DEC_VALID_COMMA_ONLY = "TRUE";
defparam user_instance_name.MCOMMA_10B_VALUE = 10'b1100000000;
defparam user_instance_name.MCOMMA_DETECT = "TRUE";
defparam user_instance_name.PCOMMA_10B_VALUE = 10'b0011111000;
defparam user_instance_name.PCOMMA_DETECT = "TRUE";
defparam user_instance_name.RX_BUFFER_USE = "TRUE";
defparam user_instance_name.RX_CRC_USE = "FALSE";
defparam user_instance_name.RX_DATA_WIDTH = 2;
defparam user_instance_name.RX_DECODE_USE = "TRUE";
defparam user_instance_name.RX_LOSS_OF_SYNC_FSM = "TRUE";
defparam user_instance_name.RX_LOS_INVALID_INCR = 1;
defparam user_instance_name.RX_LOS_THRESHOLD = 4;
defparam user_instance_name.SERDES_10B = "FALSE";
defparam user_instance_name.TERMINATION_IMP = 50;
defparam user_instance_name.TX_BUFFER_USE = "TRUE";
defparam user_instance_name.TX_CRC_FORCE_VALUE = 8'b11010110;
defparam user_instance_name.TX_CRC_USE = "FALSE";
defparam user_instance_name.TX_DATA_WIDTH = 2;
defparam user_instance_name.TX_DIFF_CTRL = 500;
defparam user_instance_name.TX_PREEMPHASIS = 0
```

Commonly Used Constraints

None

GT_ETHERNET_*n*

Gigabit Transceiver for High-Speed I/O

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II Pro*	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A

*Supported for Virtex-II Pro but not for Virtex-II

This Ethernet gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

You can also set attributes for the primitives. See Table 1-8 of the *Rocket I/O Transceiver User Guide* for a description of these attributes. See Table 1-9 of the *Rocket I/O Transceiver User Guide* for the default attribute values.

The following table lists the input and output ports for all values of *n*. For a description of each of the ports, see Table 1-7 in the *Rocket I/O Transceiver User Guide.*

Inputs	Outputs
BREFCLK	
BREFCLK2	
CONFIGENABLE	CONFIGOUT
CONFIGIN	RXBUFSTATUS [1:0]
ENMCOMMAALIGN	RXCHARISCOMMA [0:0] (GT_ETHERNET_1) RXCHARISCOMMA [1:0] (GT_ETHERNET_2) RXCHARISCOMMA [3:0] (GT_ETHERNET_4)
ENPCOMMAALIGN	RXCHARISK [0:0] (GT_ETHERNET_1) RXCHARISK [1:0] (GT_ETHERNET_2) RXCHARISK [3:0] (GT_ETHERNET_4)
LOOPBACK [1:0]	RXCHECKINGCRC
POWERDOWN	RXCLKCORCNT [2:0]
REFCLK	RXCOMMADET
REFCLK2	RXCRCERR
REFCLKSEL	RXDATA [7:0] (GT_ETHERNET_1) RXDATA [15:0] (GT_ETHERNET_2) RXDATA [31:0] (GT_ETHERNET_4)
RXN	RXDISPERR [0:0] (GT_ETHERNET_1) RXDISPERR [1:0] (GT_ETHERNET_2) RXDISPERR [3:0] (GT_ETHERNET_4
RXP	RXLOSSOFSYNC [1:0]
RXPOLARITY	RXNOTINTABLE [0:0] (GT_ETHERNET_1) RXNOTINTABLE [1:0] (GT_ETHERNET_2) RXNOTINTABLE [3:0] (GT_ETHERNET_4)
RXRESET	RXREALIGN
RXUSRCLK	RXRECCLK

Inputs	Outputs
RXUSRCLK2	RXRUNDISP [0:0] (GT_ETHERNET_1) RXRUNDISP [1:0] (GT_ETHERNET_2) RXRUNDISP [3:0] (GT_ETHERNET_4)
TXBYPASS8B10B [0:0] (GT_ETHERNET_1) TXBYPASS8B10B [1:0] (GT_ETHERNET_2) TXBYPASS8B10B [3:0] (GT_ETHERNET_4)	TXBUFERR
TXCHARDISPMODE [0:0] (GT_ETHERNET_1) TXCHARDISPMODE [1:0] (GT_ETHERNET_2) TXCHARDISPMODE [3:0] (GT_ETHERNET_4)	TXKERR [0:0] (GT_ETHERNET_1) TXKERR [1:0] (GT_ETHERNET_2) TXKERR [3:0] (GT_ETHERNET_4)
TXCHARDISPVAL [0:0] (GT_ETHERNET_1) TXCHARDISPVAL [1:0] (GT_ETHERNET_2) TXCHARDISPVAL [3:0] (GT_ETHERNET_4)	TXN
TXCHARISK [0:0] (GT_ETHERNET_1) TXCHARISK [1:0] (GT_ETHERNET_2) TXCHARISK [3:0] (GT_ETHERNET_4)	TXP
TXDATA [7:0] (GT_ETHERNET_1) TXDATA [15:0] (GT_ETHERNET_2) TXDATA [31:0] (GT_ETHERNET_4)	TXRUNDISP [0:0] (GT_ETHERNET_1) TXRUNDISP [1:0] (GT_ETHERNET_2) TXRUNDISP [3:0] (GT_ETHERNET_4)
TXFORCECRCERR	
TXINHIBIT	
TXPOLARITY	
TXRESET	
TXUSRCLK	
TXUSRCLK2	

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

```
-- Component Declaration for GT_ETHERNET_n should be placed
-- after architecture statement but before begin keyword

component GT_ETHERNET_n
  -- synthesis translate_off
  generic (CLK_COR_INSERT_IDLE_FLAG  : boolean := FALSE;
           CLK_COR_KEEP_IDLE         : boolean := FALSE;
           CLK_COR_REPEAT_WAIT       : integer := 1;
           REF_CLK_V_SEL             : integer := 0;
           RX_CRC_USE                : boolean := FALSE;
           RX_LOSS_OF_SYNC_FSM       : boolean := FALSE;
           RX_LOS_INVALID_INCR       : integer := 1;
           RX_LOS_THRESHOLD          : integer := 4;
           SERDES_10B                : boolean := FALSE;
           TERMINATION_IMP           : real := 50;
           TX_CRC_FORCE_VALUE        : bit_vector  := "11010110";
           TX_CRC_USE                : boolean := FALSE;
           TX_DIFF_CTRL              : integer := 500;
```

```
          TX_PREEMPHASIS           : integer := 0);
  -- synthesis translate_on
  port(CONFIGOUT     : out STD_ULOGIC;
       RXBUFSTATUS   : out STD_LOGIC_VECTOR (1 downto 0);
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
       RXCHARISK     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
       RXCHARISK     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
       RXCHARISK     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
       RXCHECKINGCRC : out STD_ULOGIC;
       RXCLKCORCNT   : out STD_LOGIC_VECTOR (2 downto 0);
       RXCOMMADET    : out STD_ULOGIC;
       RXCRCERR      : out STD_ULOGIC;
       RXDATA        : out STD_LOGIC_VECTOR (7 downto 0); -- for GT_ETHERNET_1
       RXDATA        : out STD_LOGIC_VECTOR (15 downto 0); -- for GT_ETHERNET_2
       RXDATA        : out STD_LOGIC_VECTOR (31 downto 0); -- for GT_ETHERNET_4
       RXDISPERR     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
       RXDISPERR     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
       RXDISPERR     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
       RXLOSSOFSYNC  : out STD_LOGIC_VECTOR (1 downto 0);
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
       RXREALIGN     : out STD_ULOGIC;
       RXRECCLK      : out STD_ULOGIC;
       RXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
       RXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
       RXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
       TXBUFERR      : out STD_ULOGIC;
       TXKERR        : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
       TXKERR        : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
       TXKERR        : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
       TXN           : out STD_ULOGIC;
       TXP           : out STD_ULOGIC;
       TXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
       TXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
       TXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
       BREFCLK       : in STD_ULOGIC;
       BREFCLK2      : in STD_ULOGIC;
       CONFIGENABLE  : in STD_ULOGIC;
       CONFIGIN      : in STD_ULOGIC;
       ENMCOMMAALIGN : in STD_ULOGIC;
       ENPCOMMAALIGN : in STD_ULOGIC;
       LOOPBACK      : in STD_LOGIC_VECTOR (1 downto 0);
       POWERDOWN     : in STD_ULOGIC;
       REFCLK        : in STD_ULOGIC;
       REFCLK2       : in STD_ULOGIC;
       REFCLKSEL     : in STD_ULOGIC;
       RXN           : in STD_ULOGIC;
       RXP           : in STD_ULOGIC;
       RXPOLARITY    : in STD_ULOGIC;
       RXRESET       : in STD_ULOGIC;
       RXUSRCLK      : in STD_ULOGIC;
```

```
      RXUSRCLK2     : in STD_ULOGIC;
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
      TXCHARISK     : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_ETHERNET_1
      TXCHARISK     : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_ETHERNET_2
      TXCHARISK     : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_ETHERNET_4
      TXDATA        : in STD_LOGIC_VECTOR (7 downto 0); -- for GT_ETHERNET_1
      TXDATA        : in STD_LOGIC_VECTOR (15 downto 0); -- for GT_ETHERNET_2
      TXDATA        : in STD_LOGIC_VECTOR (31 downto 0); -- for GT_ETHERNET_4
      TXFORCECRCERR : in STD_ULOGIC;
      TXINHIBIT     : in STD_ULOGIC;
      TXPOLARITY    : in STD_ULOGIC;
      TXRESET       : in STD_ULOGIC;
      TXUSRCLK      : in STD_ULOGIC;
      TXUSRCLK2     : in STD_ULOGIC);
end component;

-- Component Attribute specification for GT_ETHERNET_n
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here


-- Component Instantiation for GT_ETHERNET_n should be placed
-- in architecture after the begin keyword

GT_ETHERNET_n_INSTANCE_NAME : GT_ETHERNET_n
  -- synthesis translate_off
  generic map(CLK_COR_INSERT_IDLE_FLAG => boolean_value,
              CLK_COR_KEEP_IDLE         => boolean_value,
              CLK_COR_REPEAT_WAIT       => integer_value,
              REF_CLK_V_SEL             => integer_value,
              RX_CRC_USE                => boolean_value,
              RX_LOSS_OF_SYNC_FSM       => boolean_value,
              RX_LOS_INVALID_INCR       => integer_value,
              RX_LOS_THRESHOLD          => integer_value,
              SERDES_10B                => boolean_value,
              TERMINATION_IMP           => integer_value,
              TX_CRC_FORCE_VALUE        => bit_value,
              TX_CRC_USE                => boolean_value,
              TX_DIFF_CTRL              => integer_value,
              TX_PREEMPHASIS            => integer_value);
  -- synthesis translate_on
    port map(CONFIGOUT       => user_CONFIGOUT,
             RXBUFSTATUS     => user_RXBUFSTATUS,
             RXCHARISCOMMA   => user_RXCHARISCOMMA,
```

```
RXCHARISK       => user_RXCHARISK,
RXCHECKINGCRC   => user_RXCHECKINGCRC,
RXCLKCORCNT     => user_RXCLKCORCNT,
RXCOMMADET      => user_RXCOMMADET,
RXCRCERR        => user_RXCRCERR,
RXDATA          => user_RXDATA,
RXDISPERR       => user_RXDISPERR,
RXLOSSOFSYNC    => user_RXLOSSOFSYNC,
RXNOTINTABLE    => user_RXNOTINTABLE,
RXREALIGN       => user_RXREALIGN,
RXRECCLK        => user_RXRECCLK,
RXRUNDISP       => user_RXRUNDISP,
TXBUFERR        => user_TXBUFERR,
TXKERR          => user_TXKERR,
TXN             => user_TXN,
TXP             => user_TXP,
TXRUNDISP       => user_TXRUNDISP,
BREFCLK         => user_BREFCLK,
BREFCLK2        => user_BREFCLK2,
CHBONDI         => user_CHBONDI,
CONFIGENABLE    => user_CONFIGENABLE,
CONFIGIN        => user_CONFIGIN,
ENMCOMMAALIGN   => user_ENMCOMMAALIGN,
ENPCOMMAALIGN   => user_ENPCOMMAALIGN,
LOOPBACK        => user_LOOPBACK,
POWERDOWN       => user_POWERDOWN,
REFCLK          => user_REFCLK,
REFCLK2         => user_REFCLK2,
REFCLKSEL       => user_REFCLKSEL,
RXN             => user_RXN,
RXP             => user_RXP,
RXPOLARITY      => user_RXPOLARITY,
RXRESET         => user_RXRESET,
RXUSRCLK        => user_RXUSRCLK,
RXUSRCLK2       => user_RXUSRCLK2,
TXBYPASS8B10B   => user_TXBYPASS8B10B,
TXCHARDISPMODE  => user_TXCHARDISPMODE,
TXCHARDISPVAL   => user_TXCHARDISPVAL,
TXCHARISK       => user_TXCHARISK,
TXDATA          => user_TXDATA,
TXFORCECRCERR   => user_TXFORCECRCERR,
TXINHIBIT       => user_TXINHIBIT,
TXPOLARITY      => user_TXPOLARITY,
TXRESET         => user_TXRESET,
TXUSRCLK        => user_TXUSRCLK,
TXUSRCLK2       => user_TXUSRCLK2);
```

Verilog Instantiation Template

```
GT_ETHERNET_n GT_ETHERNET_n_instance_name (.CONFIGOUT (user_CONFIGOUT),
                                           .RXBUFSTATUS (user_RXBUFSTATUS),
                                           .RXCHARISCOMMA (user_RXCHARISCOMMA),
                                           .RXCHARISK (user_RXCHARISK),
                                           .RXCHECKINGCRC (user_RXCHECKINGCRC),
                                           .RXCLKCORCNT (user_RXCLKCORCNT),
                                           .RXCOMMADET (user_RXCOMMADET),
                                           .RXCRCERR (user_RXCRCERR),
                                           .RXDATA (user_RXDATA),
                                           .RXDISPERR (user_RXDISPERR),
                                           .RXLOSSOFSYNC (user_RXLOSSOFSYNC),
                                           .RXNOTINTABLE (user_RXNOTINTABLE),
                                           .RXREALIGN (user_RXREALIGN),
                                           .RXRECCLK (user_RXRECCLK),
                                           .RXRUNDISP (user_RXRUNDISP),
                                           .TXBUFERR (user_TXBUFERR),
                                           .TXKERR (user_TXKERR),
                                           .TXN(user_TXN),
                                           .TXP (user_TXP),
                                           .TXRUNDISP (user_TXRUNDISP),
                                           .CONFIGENABLE (user_CONFIGENABLE),
                                           .CONFIGIN (user_CONFIGIN),
                                           .ENMCOMMAALIGN (user_ENMCOMMAALIGN),
                                           .ENPCOMMAALIGN (user_ENPCOMMAALIGN),
                                           .LOOPBACK (user_LOOPBACK),
                                           .POWERDOWN (user_POWERDOWN),
                                           .REFCLK (user_REFCLK),
                                           .REFCLK2 (user_REFCLK2),
                                           .REFCLKSEL (user_REFCLKSEL),
                                           .RXN (user_RXN),
                                           .RXP (user_RXP),
                                           .RXPOLARITY (user_RXPOLARITY),
                                           .RXRESET (user_RXRESET),
                                           .RXUSRCLK (user_RXUSRCLK),
                                           .RXUSRCLK2 (user_RXUSRCLK2),
                                           .TXBYPASS8B10B (user_TXBYPASS8B10B),
                                           .TXCHARDISPMODE => user_TXCHARDISPMODE),
                                           .TXCHARDISPVAL (user_TXCHARDISPVAL),
                                           .TXCHARISK (user_TXCHARISK),
                                           .TXDATA (user_TXDATA),
                                           .TXFORCECRCERR (user_TXFORCECRCERR),
                                           .TXINHIBIT (user_TXINHIBIT),
                                           .TXPOLARITY (user_TXPOLARITY),
                                           .TXRESET (user_TXRESET),
                                           .TXUSRCLK (user_TXUSRCLK),
                                           .TXUSRCLK2 (user_TXUSRCLK2));

          defparam user_instance_name.CLK_COR_INSERT_IDLE_FLAG = "FALSE";
          defparam user_instance_name.CLK_COR_KEEP_IDLE = "FALSE";
          defparam user_instance_name.CLK_COR_REPEAT_WAIT = 1;
          defparam user_instance_name.RX_CRC_USE = "FALSE";
          defparam user_instance_name.RX_LOSS_OF_SYNC_FSM = "TRUE";
```

```
defparam user_instance_name.RX_LOS_INVALID_INCR = 1;
defparam user_instance_name.RX_LOS_THRESHOLD = 4;
defparam user_instance_name.SERDES_10B = "FALSE";
defparam user_instance_name.TERMINATION_IMP = 50;
defparam user_instance_name.TX_CRC_FORCE_VALUE = bit_value;
defparam user_instance_name.TX_CRC_USE = "FALSE";
defparam user_instance_name.TX_DIFF_CTRL = 500;
defparam user_instance_name.TX_PREEMPHASIS = 0;
```

Commonly Used Constraints

None

GT_FIBRE_CHAN_*n*

Gigabit Transceiver for High-Speed I/O

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II Pro*	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A

*Supported for Virtex-II Pro but not for Virtex-II

This Fibre Channel gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

You can also set attributes for the primitives. See Table 1-8 of the *Rocket I/O Transceiver User Guide* for a description of these attributes. See Table 1-10 of the *Rocket I/O Transceiver User Guide* for the default attribute values.

The following table lists the input and output ports for all values of *n*. For a description of each of the ports, see Table 1-7 in the *Rocket I/O Transceiver User Guide.*

Inputs	Outputs
BREFCLK	
BREFCLK2	
CONFIGENABLE	CONFIGOUT
CONFIGIN	RXBUFSTATUS [1:0]
ENMCOMMAALIGN	RXCHARISCOMMA [0:0] (GT_FIBRE_CHAN_1) RXCHARISCOMMA [1:0] (GT_FIBRE_CHAN_2) RXCHARISCOMMA [3:0] (GT_FIBRE_CHAN_4)
ENPCOMMAALIGN	RXCHARISK [0:0] (GT_FIBRE_CHAN_1) RXCHARISK [1:0] (GT_FIBRE_CHAN_2) RXCHARISK [3:0] (GT_FIBRE_CHAN_4)
LOOPBACK [1:0]	RXCHECKINGCRC
POWERDOWN	RXCLKCORCNT [2:0]
REFCLK	RXCOMMADET
REFCLK2	RXCRCERR
REFCLKSEL	RXDATA [7:0] (GT_FIBRE_CHAN_1) RXDATA [15:0] (GT_FIBRE_CHAN_2) RXDATA [31:0] (GT_FIBRE_CHAN_4)
RXN	RXDISPERR [0:0] (GT_FIBRE_CHAN_1) RXDISPERR [1:0] (GT_FIBRE_CHAN_2) RXDISPERR [3:0] (GT_FIBRE_CHAN_4
RXP	RXLOSSOFSYNC [1:0]
RXPOLARITY	RXNOTINTABLE [0:0] (GT_FIBRE_CHAN_1) RXNOTINTABLE [1:0] (GT_FIBRE_CHAN_2) RXNOTINTABLE [3:0] (GT_FIBRE_CHAN_4)
RXRESET	RXREALIGN
RXUSRCLK	RXRECCLK
RXUSRCLK2	RXRUNDISP [0:0] (GT_FIBRE_CHAN_1) RXRUNDISP [1:0] (GT_FIBRE_CHAN_2) RXRUNDISP [3:0] (GT_FIBRE_CHAN_4)

Inputs	Outputs
TXBYPASS8B10B [0:0] (GT_FIBRE_CHAN_1) TXBYPASS8B10B [1:0] (GT_FIBRE_CHAN_2) TXBYPASS8B10B [3:0] (GT_FIBRE_CHAN_4)	TXBUFERR
TXCHARDISPMODE [0:0] (GT_FIBRE_CHAN_1) TXCHARDISPMODE [1:0] (GT_FIBRE_CHAN_2) TXCHARDISPMODE [3:0] (GT_FIBRE_CHAN_4)	TXKERR [0:0] (GT_FIBRE_CHAN_1) TXKERR [1:0] (GT_FIBRE_CHAN_2) TXKERR [3:0] (GT_FIBRE_CHAN_4)
TXCHARDISPVAL [0:0] (GT_FIBRE_CHAN_1) TXCHARDISPVAL [1:0] (GT_FIBRE_CHAN_2) TXCHARDISPVAL [3:0] (GT_FIBRE_CHAN_4)	TXN
TXCHARISK [0:0] (GT_FIBRE_CHAN_1) TXCHARISK [1:0] (GT_FIBRE_CHAN_2) TXCHARISK [3:0] (GT_FIBRE_CHAN_4)	TXP
TXDATA [7:0] (GT_FIBRE_CHAN_1) TXDATA [15:0] (GT_FIBRE_CHAN_2) TXDATA [31:0] (GT_FIBRE_CHAN_4)	TXRUNDISP [0:0] (GT_FIBRE_CHAN_1) TXRUNDISP [1:0] (GT_FIBRE_CHAN_2) TXRUNDISP [3:0] (GT_FIBRE_CHAN_4)
TXFORCECRCERR	
TXINHIBIT	
TXPOLARITY	
TXRESET	
TXUSRCLK	
TXUSRCLK2	

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

```
-- Component Declaration for GT_FIBRE_CHAN_n should be placed
-- after architecture statement but before begin keyword

component GT_FIBRE_CHAN_n
  -- synthesis translate_off
  generic (CLK_COR_INSERT_IDLE_FLAG  : boolean:= FALSE;
           CLK_COR_KEEP_IDLE         : boolean:= FALSE;
           CLK_COR_REPEAT_WAIT       : integer:= 2;
           REF_CLK_V_SEL             : integer := 0;
           RX_CRC_USE                : boolean:= FALSE;
           RX_LOSS_OF_SYNC_FSM       : boolean:= FALSE;
           RX_LOS_INVALID_INCR       : integer := 1;
           RX_LOS_THRESHOLD          : integer:= 4;
           SERDES_10B                : boolean:= FALSE;
           TERMINATION_IMP           : integer:= 50;
           TX_CRC_FORCE_VALUE        : bit_vector  := "11010110";
           TX_CRC_USE                : integer:= FALSE;
           TX_DIFF_CTRL              : integer:= 500;
           TX_PREEMPHASIS            : integer:= 0);
  -- synthesis translate_on
  port(CONFIGOUT     : out STD_ULOGIC;
       RXBUFSTATUS   : out STD_LOGIC_VECTOR (1 downto 0);
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
```

```
RXCHARISCOMMA : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
RXCHARISCOMMA : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
RXCHARISK     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
RXCHARISK     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
RXCHARISK     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
RXCHECKINGCRC : out STD_ULOGIC;
RXCLKCORCNT   : out STD_LOGIC_VECTOR (2 downto 0);
RXCOMMADET    : out STD_ULOGIC;
RXCRCERR      : out STD_ULOGIC;
RXDATA        : out STD_LOGIC_VECTOR (7 downto 0); -- for GT_FIBRE_CHAN_1
RXDATA        : out STD_LOGIC_VECTOR (15 downto 0); -- for GT_FIBRE_CHAN_2
RXDATA        : out STD_LOGIC_VECTOR (31 downto 0); -- for GT_FIBRE_CHAN_4
RXDISPERR     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
RXDISPERR     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
RXDISPERR     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
RXLOSSOFSYNC  : out STD_LOGIC_VECTOR (1 downto 0);
RXNOTINTABLE  : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
RXNOTINTABLE  : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
RXNOTINTABLE  : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
RXREALIGN     : out STD_ULOGIC;
RXRECCLK      : out STD_ULOGIC;
RXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
RXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
RXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
TXBUFERR      : out STD_ULOGIC;
TXKERR        : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
TXKERR        : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
TXKERR        : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
TXN           : out STD_ULOGIC;
TXP           : out STD_ULOGIC;
TXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
TXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
TXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
BREFCLK       : in STD_ULOGIC;
BREFCLK2      : in STD_ULOGIC;
CONFIGENABLE  : in STD_ULOGIC;
CONFIGIN      : in STD_ULOGIC;
ENMCOMMAALIGN : in STD_ULOGIC;
ENPCOMMAALIGN : in STD_ULOGIC;
LOOPBACK      : in STD_LOGIC_VECTOR (1 downto 0);
POWERDOWN     : in STD_ULOGIC;
REFCLK        : in STD_ULOGIC;
REFCLK2       : in STD_ULOGIC;
REFCLKSEL     : in STD_ULOGIC;
RXN           : in STD_ULOGIC;
RXP           : in STD_ULOGIC;
RXPOLARITY    : in STD_ULOGIC;
RXRESET       : in STD_ULOGIC;
RXUSRCLK      : in STD_ULOGIC;
RXUSRCLK2     : in STD_ULOGIC;
TXBYPASS8B10B : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
TXBYPASS8B10B : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
TXBYPASS8B10B : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
TXCHARDISPMODE: in STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
```

```
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
      TXCHARISK     : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_FIBRE_CHAN_1
      TXCHARISK     : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_FIBRE_CHAN_2
      TXCHARISK     : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_FIBRE_CHAN_4
      TXDATA        : in STD_LOGIC_VECTOR (7 downto 0); -- for GT_FIBRE_CHAN_1
      TXDATA        : in STD_LOGIC_VECTOR (15 downto 0); -- for GT_FIBRE_CHAN_2
      TXDATA        : in STD_LOGIC_VECTOR (31 downto 0); -- for GT_FIBRE_CHAN_4
      TXFORCECRCERR : in STD_ULOGIC;
      TXINHIBIT     : in STD_ULOGIC;
      TXPOLARITY    : in STD_ULOGIC;
      TXRESET       : in STD_ULOGIC;
      TXUSRCLK      : in STD_ULOGIC;
      TXUSRCLK2     : in STD_ULOGIC);
end component;

-- Component Attribute specification for GT_FIBRE_CHAN_n
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here


-- Component Instantiation for GT_FIBRE_CHAN_n should be placed
-- in architecture after the begin keyword

GT_FIBRE_CHAN_n_INSTANCE_NAME : GT_FIBRE_CHAN_n
  -- synthesis translate_off
  generic map(CLK_COR_INSERT_IDLE_FLAG => boolean_value,
              CLK_COR_KEEP_IDLE        => boolean_value,
              CLK_COR_REPEAT_WAIT      => integer_value,
              RX_CRC_USE               => boolean_value,
              RX_LOSS_OF_SYNC_FSM      => boolean_value,
              RX_LOS_INVALID_INCR      => integer_value,
              RX_LOS_THRESHOLD         => integer_value,
              SERDES_10B               => boolean_value,
              TERMINATION_IMP          => integer_value,
              TX_CRC_FORCE_VALUE       => bit_value,
              TX_CRC_USE               => boolean_value,
              TX_DIFF_CTRL             => integer_value,
              TX_PREEMPHASIS           => integer_value);
  -- synthesis translate_on
    port map(CONFIGOUT       => user_CONFIGOUT,
             RXBUFSTATUS     => user_RXBUFSTATUS,
             RXCHARISCOMMA   => user_RXCHARISCOMMA,
             RXCHARISK       => user_RXCHARISK,
             RXCHECKINGCRC   => user_RXCHECKINGCRC,
             RXCLKCORCNT     => user_RXCLKCORCNT,
             RXCOMMADET      => user_RXCOMMADET,
             RXCRCERR        => user_RXCRCERR,
             RXDATA          => user_RXDATA,
```

```
RXDISPERR       => user_RXDISPERR,
RXLOSSOFSYNC    => user_RXLOSSOFSYNC,
RXNOTINTABLE    => user_RXNOTINTABLE,
RXREALIGN       => user_RXREALIGN,
RXRECCLK        => user_RXRECCLK,
RXRUNDISP       => user_RXRUNDISP,
TXBUFERR        => user_TXBUFERR,
TXKERR          => user_TXKERR,
TXN             => user_TXN,
TXP             => user_TXP,
TXRUNDISP       => user_TXRUNDISP,
BREFCLK         => user_BREFCLK,
BREFCLK         => user_BREFCLK,
CHBONDI         => user_CHBONDI,
CONFIGENABLE    => user_CONFIGENABLE,
CONFIGIN        => user_CONFIGIN,
ENMCOMMAALIGN   => user_ENMCOMMAALIGN,
ENPCOMMAALIGN   => user_ENPCOMMAALIGN,
LOOPBACK        => user_LOOPBACK,
POWERDOWN       => user_POWERDOWN,
REFCLK          => user_REFCLK,
REFCLK2         => user_REFCLK2,
REFCLKSEL       => user_REFCLKSEL,
RXN             => user_RXN,
RXP             => user_RXP,
RXPOLARITY      => user_RXPOLARITY,
RXRESET         => user_RXRESET,
RXUSRCLK        => user_RXUSRCLK,
RXUSRCLK2       => user_RXUSRCLK2,
TXBYPASS8B10B   => user_TXBYPASS8B10B,
TXCHARDISPMODE  => user_TXCHARDISPMODE,
TXCHARDISPVAL   => user_TXCHARDISPVAL,
TXCHARISK       => user_TXCHARISK,
TXDATA          => user_TXDATA,
TXFORCECRCERR   => user_TXFORCECRCERR,
TXINHIBIT       => user_TXINHIBIT,
TXPOLARITY      => user_TXPOLARITY,
TXRESET         => user_TXRESET,
TXUSRCLK        => user_TXUSRCLK,
TXUSRCLK2       => user_TXUSRCLK2);
```

Verilog Instantiation Template

```
GT_FIBRE_CHAN_n GT_FIBRE_CHAN_n_instance_name (.CONFIGOUT (user_CONFIGOUT),
                                               .RXBUFSTATUS (user_RXBUFSTATUS),
                                               .RXCHARISCOMMA (user_RXCHARISCOMMA),
                                               .RXCHARISK (user_RXCHARISK),
                                               .RXCHECKINGCRC (user_RXCHECKINGCRC),
                                               .RXCLKCORCNT (user_RXCLKCORCNT),
                                               .RXCOMMADET (user_RXCOMMADET),
                                               .RXCRCERR (user_RXCRCERR),
                                               .RXDATA (user_RXDATA),
                                               .RXDISPERR (user_RXDISPERR),
                                               .RXLOSSOFSYNC (user_RXLOSSOFSYNC),
                                               .RXNOTINTABLE (user_RXNOTINTABLE),
                                               .RXREALIGN (user_RXREALIGN),
                                               .RXRECCLK (user_RXRECCLK),
                                               .RXRUNDISP (user_RXRUNDISP),
                                               .TXBUFERR (user_TXBUFERR),
                                               .TXKERR (user_TXKERR),
                                               .TXN(user_TXN),
                                               .TXP (user_TXP),
                                               .TXRUNDISP (user_TXRUNDISP),
                                               .CONFIGENABLE (user_CONFIGENABLE),
                                               .CONFIGIN (user_CONFIGIN),
                                               .ENMCOMMAALIGN (user_ENMCOMMAALIGN),
                                               .ENPCOMMAALIGN (user_ENPCOMMAALIGN),
                                               .LOOPBACK (user_LOOPBACK),
                                               .POWERDOWN (user_POWERDOWN),
                                               .REFCLK (user_REFCLK),
                                               .REFCLK2 (user_REFCLK2),
                                               .REFCLKSEL (user_REFCLKSEL),
                                               .RXN (user_RXN),
                                               .RXP (user_RXP),
                                               .RXPOLARITY (user_RXPOLARITY),
                                               .RXRESET (user_RXRESET),
                                               .RXUSRCLK (user_RXUSRCLK),
                                               .RXUSRCLK2 (user_RXUSRCLK2),
                                               .TXBYPASS8B10B (user_TXBYPASS8B10B),
                                               .TXCHARDISPMODE(user_TXCHARDISPMODE),
                                               .TXCHARDISPVAL (user_TXCHARDISPVAL),
                                               .TXCHARISK (user_TXCHARISK),
                                               .TXDATA (user_TXDATA),
                                               .TXFORCECRCERR (user_TXFORCECRCERR),
                                               .TXINHIBIT (user_TXINHIBIT),
                                               .TXPOLARITY (user_TXPOLARITY),
                                               .TXRESET (user_TXRESET),
                                               .TXUSRCLK (user_TXUSRCLK),
                                               .TXUSRCLK2 (user_TXUSRCLK2));

          defparam user_instance_name.CLK_COR_INSERT_IDLE_FLAG = "FALSE";
          defparam user_instance_name.CLK_COR_KEEP_IDLE = "FALSE";
          defparam user_instance_name.CLK_COR_REPEAT_WAIT = 2;
          defparam user_instance_name.RX_CRC_USE = "FALSE";
          defparam user_instance_name.RX_LOSS_OF_SYNC_FSM = "FALSE";
```

```
defparam user_instance_name.RX_LOS_INVALID_INCR = 1;
defparam user_instance_name.RX_LOS_THRESHOLD = 4;
defparam user_instance_name.SERDES_10B = "FALSE";
defparam user_instance_name.TERMINATION_IMP = 50;
defparam user_instance_name.TX_CRC_FORCE_VALUE = bit_value;
defparam user_instance_name.TX_CRC_USE = "FALSE";
defparam user_instance_name.TX_DIFF_CTRL = 500;
defparam user_instance_name.TX_PREEMPHASIS = 0;
```

Commonly Used Constraints

None

GT_INFINIBAND_*n*

Gigabit Transceiver for High-Speed I/O

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II Pro*	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A

*Supported for Virtex-II Pro but not for Virtex-II

This Infiniband gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

You can also set attributes for the primitives. See Table 1-8 of the *Rocket I/O Transceiver User Guide* for a description of these attributes. See Table 1-10 of the *Rocket I/O Transceiver User Guide* for the default attribute values.

The following table lists the input and output ports for all values of *n*. For a description of each of the ports, see Table 1-7 in the *Rocket I/O Transceiver User Guide.*

Inputs	Outputs
BREFCLK	
BREFCLK2	
CHBONDI [3:0]	CHBONDDONE
CONFIGENABLE	CHBONDO [3:0]
CONFIGIN	CONFIGOUT
ENCHANSYNC	RXBUFSTATUS [1:0]
ENMCOMMAALIGN	RXCHARISCOMMA [0:0] (GT_INFINIBAND_1) RXCHARISCOMMA [1:0] (GT_INFINIBAND_2) RXCHARISCOMMA [3:0] (GT_INFINIBAND_4)
ENPCOMMAALIGN	RXCHARISK [0:0] (GT_INFINIBAND_1) RXCHARISK [1:0] (GT_INFINIBAND_2) RXCHARISK [3:0] (GT_INFINIBAND_4)
LOOPBACK [1:0]	RXCHECKINGCRC
POWERDOWN	RXCLKCORCNT [2:0]
REFCLK	RXCOMMADET
REFCLK2	RXCRCERR
REFCLKSEL	RXDATA [7:0] (GT_INFINIBAND_1) RXDATA [15:0] (GT_INFINIBAND_2) RXDATA [31:0] (GT_INFINIBAND_4)
RXN	RXDISPERR [0:0] (GT_INFINIBAND_1) RXDISPERR [1:0] (GT_INFINIBAND_2) RXDISPERR [3:0] (GT_INFINIBAND_4
RXP	RXLOSSOFSYNC [1:0]
RXPOLARITY	RXNOTINTABLE [0:0] (GT_INFINIBAND_1) RXNOTINTABLE [1:0] (GT_INFINIBAND_2) RXNOTINTABLE [3:0] (GT_INFINIBAND_4)
RXRESET	RXREALIGN
RXUSRCLK	RXRECCLK

Inputs	Outputs
RXUSRCLK2	RXRUNDISP [0:0] (GT_INFINIBAND_1) RXRUNDISP [1:0] (GT_INFINIBAND_2) RXRUNDISP [3:0] (GT_INFINIBAND_4)
TXBYPASS8B10B [0:0] (GT_INFINIBAND_1) TXBYPASS8B10B [1:0] (GT_INFINIBAND_2) TXBYPASS8B10B [3:0] (GT_INFINIBAND_4)	TXBUFERR
TXCHARDISPMODE [0:0] (GT_INFINIBAND_1) TXCHARDISPMODE [1:0] (GT_INFINIBAND_2) TXCHARDISPMODE [3:0] (GT_INFINIBAND_4)	TXKERR [0:0] (GT_INFINIBAND_1) TXKERR [1:0] (GT_INFINIBAND_2) TXKERR [3:0] (GT_INFINIBAND_4)
TXCHARDISPVAL [0:0] (GT_INFINIBAND_1) TXCHARDISPVAL [1:0] (GT_INFINIBAND_2) TXCHARDISPVAL [3:0] (GT_INFINIBAND_4)	TXN
TXCHARISK [0:0] (GT_INFINIBAND_1) TXCHARISK [1:0] (GT_INFINIBAND_2) TXCHARISK [3:0] (GT_INFINIBAND_4)	TXP
TXDATA [7:0] (GT_INFINIBAND_1) TXDATA [15:0] (GT_INFINIBAND_2) TXDATA [31:0] (GT_INFINIBAND_4)	TXRUNDISP [0:0] (GT_INFINIBAND_1) TXRUNDISP [1:0] (GT_INFINIBAND_2) TXRUNDISP [3:0] (GT_INFINIBAND_4)
TXFORCECRCERR	
TXINHIBIT	
TXPOLARITY	
TXRESET	
TXUSRCLK	
TXUSRCLK2	

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

```
-- Component Declaration for GT_INFINIBAND_n should be placed
-- after architecture statement but before begin keyword

component GT_INFINIBAND_n
  -- synthesis translate_off
  generic (CHAN_BOND_MODE            : string := "OFF";
           CHAN_BOND_ONE_SHOT        : boolean := FALSE;
           CLK_COR_INSERT_IDLE_FLAG  : boolean := FALSE;
           CLK_COR_KEEP_IDLE         : boolean := FALSE;
           CLK_COR_REPEAT_WAIT       : integer := 1;
           LANE_ID                   : bit_vector  := X"0";
           REF_CLK_V_SEL             : integer := 0;
           RX_CRC_USE                : boolean := FALSE;
           RX_LOSS_OF_SYNC_FSM       : boolean := FALSE;
           RX_LOS_INVALID_INCR       : integer := 1;
           RX_LOS_THRESHOLD          : integer := 4;
           SERDES_10B                : boolean := FALSE;
           TERMINATION_IMP           : integer := 50;
           TX_CRC_FORCE_VALUE        : bit_vector  := "11010110";
           TX_CRC_USE                : boolean := FALSE;
```

```
         TX_DIFF_CTRL              : integer := 500;
         TX_PREEMPHASIS            : integer := 0);
  -- synthesis translate_on
  port(CHBONDDONE    : out STD_ULOGIC;
       CHBONDO       : out STD_LOGIC_VECTOR (3 downto 0);
       CONFIGOUT     : out STD_ULOGIC;
       RXBUFSTATUS   : out STD_LOGIC_VECTOR (1 downto 0);
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
       RXCHARISCOMMA : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
       RXCHARISK     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
       RXCHARISK     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
       RXCHARISK     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
       RXCHECKINGCRC : out STD_ULOGIC;
       RXCLKCORCNT   : out STD_LOGIC_VECTOR (2 downto 0);
       RXCOMMADET    : out STD_ULOGIC;
       RXCRCERR      : out STD_ULOGIC;
       RXDATA        : out STD_LOGIC_VECTOR (7 downto 0); -- for GT_INFINIBAND_1
       RXDATA        : out STD_LOGIC_VECTOR (15 downto 0); -- for GT_INFINIBAND_2
       RXDATA        : out STD_LOGIC_VECTOR (31 downto 0); -- for GT_INFINIBAND_4
       RXDISPERR     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
       RXDISPERR     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
       RXDISPERR     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
       RXLOSSOFSYNC  : out STD_LOGIC_VECTOR (1 downto 0);
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
       RXNOTINTABLE  : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
       RXREALIGN     : out STD_ULOGIC;
       RXRECCLK      : out STD_ULOGIC;
       RXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
       RXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
       RXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
       TXBUFERR      : out STD_ULOGIC;
       TXKERR        : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
       TXKERR        : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
       TXKERR        : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
       TXN           : out STD_ULOGIC;
       TXP           : out STD_ULOGIC;
       TXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
       TXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
       TXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
       BREFCLK       : in STD_ULOGIC;
       BREFCLK2      : in STD_ULOGIC;
       CHBONDI       : in STD_LOGIC_VECTOR (3 downto 0);
       CONFIGENABLE  : in STD_ULOGIC;
       CONFIGIN      : in STD_ULOGIC;
       ENCHANSYNC    : in STD_ULOGIC;
       ENMCOMMAALIGN : in STD_ULOGIC;
       ENPCOMMAALIGN : in STD_ULOGIC;
       LOOPBACK      : in STD_LOGIC_VECTOR (1 downto 0);
       POWERDOWN     : in STD_ULOGIC;
       REFCLK        : in STD_ULOGIC;
       REFCLK2       : in STD_ULOGIC;
       REFCLKSEL     : in STD_ULOGIC;
```

```
      RXN           : in STD_ULOGIC;
      RXP           : in STD_ULOGIC;
      RXPOLARITY    : in STD_ULOGIC;
      RXRESET       : in STD_ULOGIC;
      RXUSRCLK      : in STD_ULOGIC;
      RXUSRCLK2     : in STD_ULOGIC;
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
      TXCHARISK     : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_INFINIBAND_1
      TXCHARISK     : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_INFINIBAND_2
      TXCHARISK     : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_INFINIBAND_4
      TXDATA        : in STD_LOGIC_VECTOR (7 downto 0); -- for GT_INFINIBAND_1
      TXDATA        : in STD_LOGIC_VECTOR (15 downto 0); -- for GT_INFINIBAND_2
      TXDATA        : in STD_LOGIC_VECTOR (31 downto 0); -- for GT_INFINIBAND_4
      TXFORCECRCERR : in STD_ULOGIC;
      TXINHIBIT     : in STD_ULOGIC;
      TXPOLARITY    : in STD_ULOGIC;
      TXRESET       : in STD_ULOGIC;
      TXUSRCLK      : in STD_ULOGIC;
      TXUSRCLK2     : in STD_ULOGIC);
end component;

-- Component Attribute specification for GT_INFINIBAND_n
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here


-- Component Instantiation for GT_INFINIBAND_n should be placed
-- in architecture after the begin keyword

GT_INFINIBAND_n_INSTANCE_NAME : GT_INFINIBAND_n
  -- synthesis translate_off
  generic map(CHAN_BOND_MODE        => string_value,
              CHAN_BOND_ONE_SHOT        => boolean_value,
              CLK_COR_INSERT_IDLE_FLAG => boolean_value,
              CLK_COR_KEEP_IDLE         => boolean_value,
              CLK_COR_REPEAT_WAIT       => integer_value,
              LANE_ID                   => bit_value,
              REF_CLK_V_SEL             => integer_value,
              RX_CRC_USE                => boolean_value,
              RX_LOSS_OF_SYNC_FSM       => boolean_value,
              RX_LOS_INVALID_INCR       => integer_value,
              RX_LOS_THRESHOLD          => integer_value,
              SERDES_10B                => boolean_value,
              TERMINATION_IMP           => integer_value,
```

```
            TX_CRC_FORCE_VALUE        => bit_value,
            TX_CRC_USE                => boolean_value,
            TX_DIFF_CTRL              => integer_value,
            TX_PREEMPHASIS            => integer_value);
-- synthesis translate_on
  port map(CHBONDDONE      => user_CHBONDDONE,
           CHBONDO         => user_CHBONDO,
           CONFIGOUT       => user_CONFIGOUT,
           RXBUFSTATUS     => user_RXBUFSTATUS,
           RXCHARISCOMMA   => user_RXCHARISCOMMA,
           RXCHARISK       => user_RXCHARISK,
           RXCHECKINGCRC   => user_RXCHECKINGCRC,
           RXCLKCORCNT     => user_RXCLKCORCNT,
           RXCOMMADET      => user_RXCOMMADET,
           RXCRCERR        => user_RXCRCERR,
           RXDATA          => user_RXDATA,
           RXDISPERR       => user_RXDISPERR,
           RXLOSSOFSYNC    => user_RXLOSSOFSYNC,
           RXNOTINTABLE    => user_RXNOTINTABLE,
           RXREALIGN       => user_RXREALIGN,
           RXRECCLK        => user_RXRECCLK,
           RXRUNDISP       => user_RXRUNDISP,
           TXBUFERR        => user_TXBUFERR,
           TXKERR          => user_TXKERR,
           TXN             => user_TXN,
           TXP             => user_TXP,
           TXRUNDISP       => user_TXRUNDISP,
           BREFCLK         => user_BREFCLK,
           BREFCLK2        => user_BREFCLK2,
           CHBONDI         => user_CHBONDI,
           CONFIGENABLE    => user_CONFIGENABLE,
           CONFIGIN        => user_CONFIGIN,
           ENCHANSYNC      => user_ENCHANSYNC,
           ENMCOMMAALIGN   => user_ENMCOMMAALIGN,
           ENPCOMMAALIGN   => user_ENPCOMMAALIGN,
           LOOPBACK        => user_LOOPBACK,
           POWERDOWN       => user_POWERDOWN,
           REFCLK          => user_REFCLK,
           REFCLK2         => user_REFCLK2,
           REFCLKSEL       => user_REFCLKSEL,
           RXN             => user_RXN,
           RXP             => user_RXP,
           RXPOLARITY      => user_RXPOLARITY,
           RXRESET         => user_RXRESET,
           RXUSRCLK        => user_RXUSRCLK,
           RXUSRCLK2       => user_RXUSRCLK2,
           TXBYPASS8B10B   => user_TXBYPASS8B10B,
           TXCHARDISPMODE  => user_TXCHARDISPMODE,
           TXCHARDISPVAL   => user_TXCHARDISPVAL,
           TXCHARISK       => user_TXCHARISK,
           TXDATA          => user_TXDATA,
           TXFORCECRCERR   => user_TXFORCECRCERR,
           TXINHIBIT       => user_TXINHIBIT,
           TXPOLARITY      => user_TXPOLARITY,
```

```
            TXRESET         => user_TXRESET,
            TXUSRCLK        => user_TXUSRCLK,
            TXUSRCLK2       => user_TXUSRCLK2);
```

Verilog Instantiation Template

```
GT_INFINIBAND_n GT_INFINIBAND_n_instance_name (.CHBONDDONE (user_CHBONDDONE),
                                               .CHBONDO (user_CHBONDO),
                                               .CONFIGOUT (user_CONFIGOUT),
                                               .RXBUFSTATUS (user_RXBUFSTATUS),
                                               .RXCHARISCOMMA (user_RXCHARISCOMMA),
                                               .RXCHARISK (user_RXCHARISK),
                                               .RXCHECKINGCRC (user_RXCHECKINGCRC),
                                               .RXCLKCORCNT (user_RXCLKCORCNT),
                                               .RXCOMMADET (user_RXCOMMADET),
                                               .RXCRCERR (user_RXCRCERR),
                                               .RXDATA (user_RXDATA),
                                               .RXDISPERR (user_RXDISPERR),
                                               .RXLOSSOFSYNC (user_RXLOSSOFSYNC),
                                               .RXNOTINTABLE (user_RXNOTINTABLE),
                                               .RXREALIGN (user_RXREALIGN),
                                               .RXRECCLK (user_RXRECCLK),
                                               .RXRUNDISP (user_RXRUNDISP),
                                               .TXBUFERR (user_TXBUFERR),
                                               .TXKERR (user_TXKERR),
                                               .TXN(user_TXN),
                                               .TXP (user_TXP),
                                               .TXRUNDISP (user_TXRUNDISP),
                                               .CHBONDI (user_CHBONDI),
                                               .CONFIGENABLE (user_CONFIGENABLE),
                                               .CONFIGIN (user_CONFIGIN),
                                               .ENCHANSYNC (user_ENCHANSYNC),
                                               .ENMCOMMAALIGN (user_ENMCOMMAALIGN),
                                               .ENPCOMMAALIGN (user_ENPCOMMAALIGN),
                                               .LOOPBACK (user_LOOPBACK),
                                               .POWERDOWN (user_POWERDOWN),
                                               .REFCLK (user_REFCLK),
                                               .REFCLK2 (user_REFCLK2),
                                               .REFCLKSEL (user_REFCLKSEL),
                                               .RXN (user_RXN),
                                               .RXP (user_RXP),
                                               .RXPOLARITY (user_RXPOLARITY),
                                               .RXRESET (user_RXRESET),
                                               .RXUSRCLK (user_RXUSRCLK),
                                               .RXUSRCLK2 (user_RXUSRCLK2),
                                               .TXBYPASS8B10B (user_TXBYPASS8B10B),
                                               .TXCHARDISPMODE(user_TXCHARDISPMODE),
                                               .TXCHARDISPVAL (user_TXCHARDISPVAL),
                                               .TXCHARISK (user_TXCHARISK),
                                               .TXDATA (user_TXDATA),
                                               .TXFORCECRCERR (user_TXFORCECRCERR),
                                               .TXINHIBIT (user_TXINHIBIT),
                                               .TXPOLARITY (user_TXPOLARITY),
                                               .TXRESET (user_TXRESET),
```

```
                                        .TXUSRCLK (user_TXUSRCLK),
                                        .TXUSRCLK2 (user_TXUSRCLK2));

defparam user_instance_name.CHAN_BOND_MODE = "OFF";
defparam user_instance_name.CHAN_BOND_ONE_SHOT = "FALSE";
defparam user_instance_name.CLK_COR_INSERT_IDLE_FLAG = "FALSE";
defparam user_instance_name.CLK_COR_KEEP_IDLE = "FALSE";
defparam user_instance_name.CLK_COR_REPEAT_WAIT = 1;
defparam user_instance_name.LANE_ID = "bit_value";
defparam user_instance_name.RX_CRC_USE = "FALSE";
defparam user_instance_name.RX_LOSS_OF_SYNC_FSM = "FALSE";
defparam user_instance_name.RX_LOS_INVALID_INCR = 1;
defparam user_instance_name.RX_LOS_THRESHOLD = 4;
defparam user_instance_name.SERDES_10B = "FALSE";
defparam user_instance_name.TERMINATION_IMP = 50;
defparam user_instance_name.TX_CRC_FORCE_VALUE = bit_value;
defparam user_instance_name.TX_CRC_USE = "FALSE";
defparam user_instance_name.TX_DIFF_CTRL = 500;
defparam user_instance_name.TX_PREEMPHASIS = 0;
```

Commonly Used Constraints

None

GT_XAUI_*n*

Gigabit Transceiver for High-Speed I/O

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II Pro*	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A

*Supported for Virtex-II Pro but not for Virtex-II

This XAUI gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

You can also set attributes for the primitives. See Table 1-8 of the *Rocket I/O Transceiver User Guide* for a description of these attributes. See Table 1-10 of the *Rocket I/O Transceiver User Guide* for the default attribute values.

The following table lists the input and output ports for all values of *n*. For a description of each of the ports, see Table 1-7 in the *Rocket I/O Transceiver User Guide.*

Inputs	Outputs
BREFCLK	
BREFCLK2	
CHBONDI [3:0]	CHBONDDONE
CONFIGENABLE	CHBONDO [3:0]
CONFIGIN	CONFIGOUT
ENCHANSYNC	RXBUFSTATUS [1:0]
ENMCOMMAALIGN	RXCHARISCOMMA [0:0] (GT_XAUI_1) RXCHARISCOMMA [1:0] (GT_XAUI_2) RXCHARISCOMMA [3:0] (GT_XAUI_4)
ENPCOMMAALIGN	RXCHARISK [0:0] (GT_XAUI_1) RXCHARISK [1:0] (GT_XAUI_2) RXCHARISK [3:0] (GT_XAUI_4)
LOOPBACK [1:0]	RXCHECKINGCRC
POWERDOWN	RXCLKCORCNT [2:0]
REFCLK	RXCOMMADET
REFCLK2	RXCRCERR
REFCLKSEL	RXDATA [7:0] (GT_XAUI_1) RXDATA [15:0] (GT_XAUI_2) RXDATA [31:0] (GT_XAUI_4)
RXN	RXDISPERR [0:0] (GT_XAUI_1) RXDISPERR [1:0] (GT_XAUI_2) RXDISPERR [3:0] (GT_XAUI_4
RXP	RXLOSSOFSYNC [1:0]
RXPOLARITY	RXNOTINTABLE [0:0] (GT_XAUI_1) RXNOTINTABLE [1:0] (GT_XAUI_2) RXNOTINTABLE [3:0] (GT_XAUI_4)
RXRESET	RXREALIGN
RXUSRCLK	RXRECCLK

Inputs	Outputs
RXUSRCLK2	RXRUNDISP [0:0] (GT_XAUI_1) RXRUNDISP [1:0] (GT_XAUI_2) RXRUNDISP [3:0] (GT_XAUI_4)
TXBYPASS8B10B [0:0] (GT_XAUI_1) TXBYPASS8B10B [1:0] (GT_XAUI_2) TXBYPASS8B10B [3:0] (GT_XAUI_4)	TXBUFERR
TXCHARDISPMODE [0:0] (GT_XAUI_1) TXCHARDISPMODE [1:0] (GT_XAUI_2) TXCHARDISPMODE [3:0] (GT_XAUI_4)	TXKERR [0:0] (GT_XAUI_1) TXKERR [1:0] (GT_XAUI_2) TXKERR [3:0] (GT_XAUI_4)
TXCHARDISPVAL [0:0] (GT_XAUI_1) TXCHARDISPVAL [1:0] (GT_XAUI_2) TXCHARDISPVAL [3:0] (GT_XAUI_4)	TXN
TXCHARISK [0:0] (GT_XAUI_1) TXCHARISK [1:0] (GT_XAUI_2) TXCHARISK [3:0] (GT_XAUI_4)	TXP
TXDATA [7:0] (GT_XAUI_1) TXDATA [15:0] (GT_XAUI_2) TXDATA [31:0] (GT_XAUI_4)	TXRUNDISP [0:0] (GT_XAUI_1) TXRUNDISP [1:0] (GT_XAUI_2) TXRUNDISP [3:0] (GT_XAUI_4)
TXFORCECRCERR	
TXINHIBIT	
TXPOLARITY	
TXRESET	
TXUSRCLK	
TXUSRCLK2	

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

```
-- Component Declaration for GT_XAUI_n should be placed
-- after architecture statement but before begin keyword

component GT_XAUI_n
  -- synthesis translate_off
  generic (CHAN_BOND_MODE            : string := "OFF";
           CHAN_BOND_ONE_SHOT        : boolean := FALSE;
           CLK_COR_INSERT_IDLE_FLAG  : boolean := FALSE;
           CLK_COR_KEEP_IDLE         : boolean := FALSE;
           CLK_COR_REPEAT_WAIT       : integer := 1;
           CRC_END_OF_PKT            : string := "K29_7";
           CRC_FORMAT                : string := "USER_MODE";
           CRC_START_OF_PKT          : string := "K27_7";
           REF_CLK_V_SEL             : integer := 0;
           RX_CRC_USE                : boolean := FALSE;
           RX_LOSS_OF_SYNC_FSM       : boolean := TRUE;
           RX_LOS_INVALID_INCR       : integer := 1;
           RX_LOS_THRESHOLD          : integer := 4;
```

```
        SERDES_10B               : boolean := FALSE;
        TERMINATION_IMP          : integer := 50;
        TX_CRC_FORCE_VALUE       : bit_vector  := "11010110";
        TX_CRC_USE               : boolean := FALSE;
        TX_DIFF_CTRL             : integer := 500;
        TX_PREEMPHASIS           : integer := 0);
-- synthesis translate_on
port(CHBONDDONE    : out STD_ULOGIC;
     CHBONDO       : out STD_LOGIC_VECTOR (3 downto 0);
     CONFIGOUT     : out STD_ULOGIC;
     RXBUFSTATUS   : out STD_LOGIC_VECTOR (1 downto 0);
     RXCHARISCOMMA : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
     RXCHARISCOMMA : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
     RXCHARISCOMMA : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
     RXCHARISK     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
     RXCHARISK     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
     RXCHARISK     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
     RXCHECKINGCRC : out STD_ULOGIC;
     RXCLKCORCNT   : out STD_LOGIC_VECTOR (2 downto 0);
     RXCOMMADET    : out STD_ULOGIC;
     RXCRCERR      : out STD_ULOGIC;
     RXDATA        : out STD_LOGIC_VECTOR (7 downto 0); -- for GT_XAUI_1
     RXDATA        : out STD_LOGIC_VECTOR (15 downto 0); -- for GT_XAUI_2
     RXDATA        : out STD_LOGIC_VECTOR (31 downto 0); -- for GT_XAUI_4
     RXDISPERR     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
     RXDISPERR     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
     RXDISPERR     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
     RXLOSSOFSYNC  : out STD_LOGIC_VECTOR (1 downto 0);
     RXNOTINTABLE  : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
     RXNOTINTABLE  : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
     RXNOTINTABLE  : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
     RXREALIGN     : out STD_ULOGIC;
     RXRECCLK      : out STD_ULOGIC;
     RXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
     RXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
     RXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
     TXN           : out STD_ULOGIC;
     TXP           : out STD_ULOGIC;
     TXBUFERR      : out STD_ULOGIC;
     TXKERR        : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
     TXKERR        : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
     TXKERR        : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
     TXRUNDISP     : out STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
     TXRUNDISP     : out STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
     TXRUNDISP     : out STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
     CHBONDI       : in STD_LOGIC_VECTOR (3 downto 0);
     BREFCLK       : in STD_ULOGIC;
     BREFCLK2      : in STD_ULOGIC;
     CONFIGENABLE  : in STD_ULOGIC;
     CONFIGIN      : in STD_ULOGIC;
     ENCHANSYNC    : in STD_ULOGIC;
     ENMCOMMAALIGN : in STD_ULOGIC;
     ENPCOMMAALIGN : in STD_ULOGIC;
     LOOPBACK      : in STD_LOGIC_VECTOR (1 downto 0);
```

```
      POWERDOWN     : in STD_ULOGIC;
      REFCLK        : in STD_ULOGIC;
      REFCLK2       : in STD_ULOGIC;
      REFCLKSEL     : in STD_ULOGIC;
      RXN           : in STD_ULOGIC;
      RXP           : in STD_ULOGIC;
      RXPOLARITY    : in STD_ULOGIC;
      RXRESET       : in STD_ULOGIC;
      RXUSRCLK      : in STD_ULOGIC;
      RXUSRCLK2     : in STD_ULOGIC;
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
      TXBYPASS8B10B : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
      TXCHARDISPMODE: in STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
      TXCHARDISPVAL : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
      TXCHARISK     : in STD_LOGIC_VECTOR (0 downto 0); -- for GT_XAUI_1
      TXCHARISK     : in STD_LOGIC_VECTOR (1 downto 0); -- for GT_XAUI_2
      TXCHARISK     : in STD_LOGIC_VECTOR (3 downto 0); -- for GT_XAUI_4
      TXDATA        : in STD_LOGIC_VECTOR (7 downto 0); -- for GT_XAUI_1
      TXDATA        : in STD_LOGIC_VECTOR (15 downto 0); -- for GT_XAUI_2
      TXDATA        : in STD_LOGIC_VECTOR (31 downto 0); -- for GT_XAUI_4
      TXFORCECRCERR : in STD_ULOGIC;
      TXINHIBIT     : in STD_ULOGIC;
      TXPOLARITY    : in STD_ULOGIC;
      TXRESET       : in STD_ULOGIC;
      TXUSRCLK      : in STD_ULOGIC;
      TXUSRCLK2     : in STD_ULOGIC);
end component;

-- Component Attribute specification for GT_XAUI_n
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here


-- Component Instantiation for GT_XAUI_n should be placed
-- in architecture after the begin keyword

GT_XAUI_n_INSTANCE_NAME : GT_XAUI_n
  -- synthesis translate_off
  generic map(CHAN_BOND_MODE        => string_value,
              CHAN_BOND_ONE_SHOT        => boolean_value,
              CLK_COR_INSERT_IDLE_FLAG => boolean_value,
              CLK_COR_KEEP_IDLE         => boolean_value,
              CLK_COR_REPEAT_WAIT       => integer _value,
              CRC_END_OF_PKT            => string_value,
              CRC_FORMAT                => string_value,
              CRC_START_OF_PKT          => string_value,
              REF_CLK_V_SEL             => integer_value,
```

```
            RX_CRC_USE                => boolean_value,
            RX_LOSS_OF_SYNC_FSM       => boolean_value,
            RX_LOS_INVALID_INCR       => integer_value,
            RX_LOS_THRESHOLD          => integer_value,
            SERDES_10B                => boolean_value,
            TERMINATION_IMP           => integer_value,
            TX_CRC_FORCE_VALUE        => bit_value,
            TX_CRC_USE                => boolean_value,
            TX_DIFF_CTRL              => integer_value,
            TX_PREEMPHASIS            => integer_value);
-- synthesis translate_on
  port map(CHBONDDONE      => user_CHBONDDONE,
           CHBONDO         => user_CHBONDO,
           CONFIGOUT       => user_CONFIGOUT,
           RXBUFSTATUS     => user_RXBUFSTATUS,
           RXCHARISCOMMA   => user_RXCHARISCOMMA,
           RXCHARISK       => user_RXCHARISK,
           RXCHECKINGCRC   => user_RXCHECKINGCRC,
           RXCLKCORCNT     => user_RXCLKCORCNT,
           RXCOMMADET      => user_RXCOMMADET,
           RXCRCERR        => user_RXCRCERR,
           RXDATA          => user_RXDATA,
           RXDISPERR       => user_RXDISPERR,
           RXLOSSOFSYNC    => user_RXLOSSOFSYNC,
           RXNOTINTABLE    => user_RXNOTINTABLE,
           RXREALIGN       => user_RXREALIGN,
           RXRECCLK        => user_RXRECCLK,
           RXRUNDISP       => user_RXRUNDISP,
           TXBUFERR        => user_TXBUFERR,
           TXKERR          => user_TXKERR,
           TXN             => user_TXN,
           TXP             => user_TXP,
           TXRUNDISP       => user_TXRUNDISP,
           BREFCLK         => user_BREFCLK,
           BREFCLK2        => user_BREFCLK2,
           CHBONDI         => user_CHBONDI,
           CONFIGENABLE    => user_CONFIGENABLE,
           CONFIGIN        => user_CONFIGIN,
           ENCHANSYNC      => user_ENCHANSYNC,
           ENMCOMMAALIGN   => user_ENMCOMMAALIGN,
           ENPCOMMAALIGN   => user_ENPCOMMAALIGN,
           LOOPBACK        => user_LOOPBACK,
           POWERDOWN       => user_POWERDOWN,
           REFCLK          => user_REFCLK,
           REFCLK2         => user_REFCLK2,
           REFCLKSEL       => user_REFCLKSEL,
           RXN             => user_RXN,
           RXP             => user_RXP,
           RXPOLARITY      => user_RXPOLARITY,
           RXRESET         => user_RXRESET,
           RXUSRCLK        => user_RXUSRCLK,
           RXUSRCLK2       => user_RXUSRCLK2,
           TXBYPASS8B10B   => user_TXBYPASS8B10B,
           TXCHARDISPMODE  => user_TXCHARDISPMODE,
```

```
            TXCHARDISPVAL   => user_TXCHARDISPVAL,
            TXCHARISK       => user_TXCHARISK,
            TXDATA          => user_TXDATA,
            TXFORCECRCERR   => user_TXFORCECRCERR,
            TXINHIBIT       => user_TXINHIBIT,
            TXPOLARITY      => user_TXPOLARITY,
            TXRESET         => user_TXRESET,
            TXUSRCLK        => user_TXUSRCLK,
            TXUSRCLK2       => user_TXUSRCLK2);
```

Verilog Instantiation Template

```
GT_XAUI_n GT_XAUI_n_instance_name (.CHBONDDONE (user_CHBONDDONE),
                                   .CHBONDO (user_CHBONDO),
                                   .CONFIGOUT (user_CONFIGOUT),
                                   .RXBUFSTATUS (user_RXBUFSTATUS),
                                   .RXCHARISCOMMA (user_RXCHARISCOMMA),
                                   .RXCHARISK (user_RXCHARISK),
                                   .RXCHECKINGCRC (user_RXCHECKINGCRC),
                                   .RXCLKCORCNT (user_RXCLKCORCNT),
                                   .RXCOMMADET (user_RXCOMMADET),
                                   .RXCRCERR (user_RXCRCERR),
                                   .RXDATA (user_RXDATA),
                                   .RXDISPERR (user_RXDISPERR),
                                   .RXLOSSOFSYNC (user_RXLOSSOFSYNC),
                                   .RXNOTINTABLE (user_RXNOTINTABLE),
                                   .RXREALIGN (user_RXREALIGN),
                                   .RXRECCLK (user_RXRECCLK),
                                   .RXRUNDISP (user_RXRUNDISP),
                                   .TXBUFERR (user_TXBUFERR),
                                   .TXKERR (user_TXKERR),
                                   .TXN(user_TXN),
                                   .TXP (user_TXP),
                                   .TXRUNDISP (user_TXRUNDISP),
                                   .CHBONDI (user_CHBONDI),
                                   .CONFIGENABLE (user_CONFIGENABLE),
                                   .CONFIGIN (user_CONFIGIN),
                                   .ENCHANSYNC (user_ENCHANSYNC),
                                   .ENMCOMMAALIGN (user_ENMCOMMAALIGN),
                                   .ENPCOMMAALIGN (user_ENPCOMMAALIGN),
                                   .LOOPBACK (user_LOOPBACK),
                                   .POWERDOWN (user_POWERDOWN),
                                   .REFCLK (user_REFCLK),
                                   .REFCLK2 (user_REFCLK2),
                                   .REFCLKSEL (user_REFCLKSEL),
                                   .RXN (user_RXN),
                                   .RXP (user_RXP),
                                   .RXPOLARITY (user_RXPOLARITY),
                                   .RXRESET (user_RXRESET),
                                   .RXUSRCLK (user_RXUSRCLK),
                                   .RXUSRCLK2 (user_RXUSRCLK2),
                                   .TXBYPASS8B10B (user_TXBYPASS8B10B),
                                   .TXCHARDISPMODE => user_TXCHARDISPMODE),
                                   .TXCHARDISPVAL (user_TXCHARDISPVAL),
```

```
                            .TXCHARISK (user_TXCHARISK),
                            .TXDATA (user_TXDATA),
                            .TXFORCECRCERR (user_TXFORCECRCERR),
                            .TXINHIBIT (user_TXINHIBIT),
                            .TXPOLARITY (user_TXPOLARITY),
                            .TXRESET (user_TXRESET),
                            .TXUSRCLK (user_TXUSRCLK),
                            .TXUSRCLK2 (user_TXUSRCLK2));

defparam user_instance_name.CHAN_BOND_MODE = "OFF";
defparam user_instance_name.CHAN_BOND_ONE_SHOT = "FALSE";
defparam user_instance_name.CLK_COR_INSERT_IDLE_FLAG = "FALSE";
defparam user_instance_name.CLK_COR_KEEP_IDLE = "FALSE";
defparam user_instance_name.CLK_COR_REPEAT_WAIT = 1;
defparam user_instance_name.CRC_END_OF_PKT = "K29_7";
defparam user_instance_name.CRC_FORMAT = "USER_MODE";
defparam user_instance_name.CRC_START_OF_PKT = "K27_7";
defparam user_instance_name.RX_CRC_USE = "FALSE";
defparam user_instance_name.RX_LOSS_OF_SYNC_FSM = "TRUE";
defparam user_instance_name.RX_LOS_INVALID_INCR = 1;
defparam user_instance_name.RX_LOS_THRESHOLD = 4;
defparam user_instance_name.SERDES_10B = "FALSE";
defparam user_instance_name.TERMINATION_IMP = 50;
defparam user_instance_name.TX_CRC_FORCE_VALUE = bit_value;
defparam user_instance_name.TX_CRC_USE = "FALSE";
defparam user_instance_name.TX_DIFF_CTRL = 500;
defparam user_instance_name.TX_PREEMPHASIS = 0;
```

Commonly Used Constraints

None

IBUF, 4, 8, 16

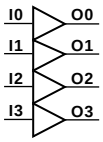
Single- and Multiple-Input Buffers

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
IBUF	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IBUF4, IBUF8, IBUF16	Macro	Macro	Macro	Macro	Macro	Macro

IBUF

X9442

IBUF4



X9443

IBUF8

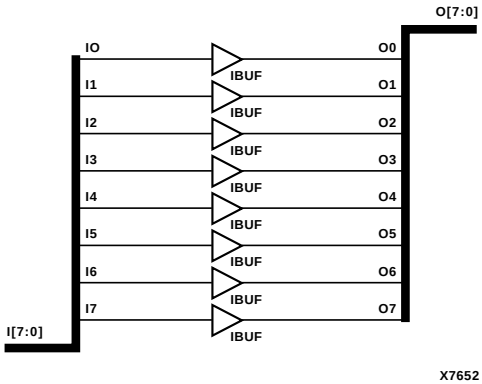
X3803

IBUF16

X3815

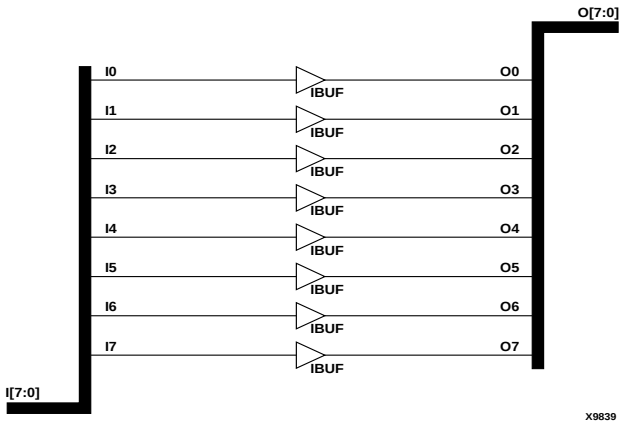
IBUF, IBUF4, IBUF8, and IBUF16 are single- and multiple-input buffers. An IBUF isolates the internal circuit from the signals coming into a chip. IBUFs are contained in input/output blocks (IOBs). IBUF inputs (I) are connected to an IPAD or an IOPAD. IBUF outputs (O) are connected to the internal circuit.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, see the "IBUF_selectIO" section for information on IBUF variants with selectable I/O interfaces.



X7652

IBUF8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-II E, Virtex, Virtex-E



X9839

IBUF8 Implementation Virtex-II, Virtex-II Pro

Usage

IBUFs are typically inferred for all top level input ports, but they can also be instantiated if necessary.

VHDL Instantiation Template

```
-- Component Declaration for IBUF should be placed
-- after architecture statement but before begin keyword

component IBUF
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for IBUF
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IBUF should be placed
-- in architecture after the begin keyword

IBUF_INSTANCE_NAME : IBUF
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
IBUF instance_name (.O (user_O),
                    .I (user_I));
```

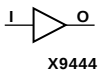
Commonly Used Constraints

BUFG (CPLDs), IOSTANDARD, IOBDELAY

IBUF_*selectIO*

Single Input Buffer with Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



X9444

For Spartan-II, Spartan-IIE, Virtex, and Virtex-E, IBUF and its selectIO variants (listed in the "Components" column in Table 6-1) are single input buffers whose I/O interface corresponds to a specific I/O standard. The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. For example, IBUF_SSTL3_II is a single input buffer that uses the SSTL3_II I/O-signaling standard. You can attach an IOSTANDARD attribute to an IBUF instance instead of using an IBUF_*selectIO* component. Check marks (√) in the "Spartan-II, Virtex" and "Spartan-IIE, Virtex-E" columns indicate the components and IOSTANDARD attribute values available for those architectures.

An IBUF isolates the internal circuit from the signals coming into a chip. For Spartan-II, Spartan-IIE, Virtex, and Virtex-E, the dedicated GCLKIOB pad is input only. IBUF inputs (I) are connected to an IPAD or IOPAD. IBUF outputs (O) are connected to the internal circuit.

The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffers. See the "SelectI/O Usage Rules" section below for information on using these components and IOSTANDARD attributes.

Spartan-II, Spartan-IIE, Virtex, and Virtex-E IBUF_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Input VCCO	VREF
IBUF	√	√	(defaults to LVTTL)	3.3	N/A
IBUF_AGP	√	√	AGP	N/A	1.32
IBUF_CTT	√	√	CTT	N/A	1.50
IBUF_GTL	√	√	GTL	N/A	0.80
IBUF_GTLP	√	√	GTLP	N/A	1.00
IBUF_HSTL_I	√	√	HSTL_I	N/A	0.75
IBUF_HSTL_III	√	√	HSTL_III	1.5	0.90
IBUF_HSTL_IV	√	√	HSTL_IV	N/A	0.90
IBUF_LVCMOS2	√	√	LVCMOS2	2.5	N/A
IBUF_LVCMOS18		√	LVCMOS18	1.8	N/A
IBUF_LVDS		√	LVDS	N/A	N/A
IBUF_LVPECL		√	LVPECL	N/A	N/A
IBUF_PCI33_3	√	√	PCI33_3	3.3	N/A
IBUF_PCI33_5	√		PCI33_5	N/A	N/A
IBUF_PCI66_3	√	√	PCI66_3	3.3	N/A
IBUF_PCIX66_3		√	PCIX66_3	3.3	N/A

Spartan-II, Spartan-IIE, Virtex, and Virtex-E IBUF_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Input VCCO	VREF
IBUF_SSTL2_I	√	√	SSTL2_I	N / A	1.25
IBUF_SSTL2_II	√	√	SSTL2_II	N / A	1.25
IBUF_SSTL3_I	√	√	SSTL3_I	N / A	1.50
IBUF_SSTL3_II	√	√	SSTL3_II	N / A	1.50

The Virtex-II and Virtex-II Pro library includes some IBUF_*selectIO* components for compatibility with older, existing designs and other architectures. For new Virtex-II and Virtex-II Pro designs, however, the recommended method for using IBUF selectIO buffers is to attach an IOSTANDARD attribute to an IBUF component. For example, attach IOSTANDARD=GTLP to an IBUF instead of using the IBUF_GTLP component for new Virtex-II and Virtex-II Pro designs. The IOSTANDARD attributes that can be attached to an IBUF component are listed in the "IOSTANDARD (Attribute Value)" column in Table 6-2. See the "SelectI/O Usage Rules" section for information on using these IOSTANDARD attributes.

Virtex-II, Virtex-II Pro IBUF_*selectIO* IOSTANDARD Attributes

Component[a]	Virtex-II	Virtex-II Pro	IOSTANDARD (Attribute Value)	Input VCCO	VREF	Terminate Type (Virtex-II only)
IBUF	√		AGP	N / A	1.32	None
IBUF	√	√	GTL	N / A	0.80	None
IBUF	√	√	GTL_DCI	1.2	0.80	Single
IBUF	√	√	GTLP	N / A	1.00	None
IBUF	√	√	GTLP_DCI	1.5	1.00	Single
IBUF	√	√	HSTL_I	N / A	0.75	None
IBUF	√	√	HSTL_I_18	1.8	0.75	None
IBUF	√	√	HSTL_I_DCI	1.5	0.75	Split
IBUF	√	√	HSTL_I_DCI_18	1.8	0.75	Split
IBUF	√	√	HSTL_II	N / A	0.75	None
IBUF	√	√	HSTL_II_18	1.8	0.75	None
IBUF	√	√	HSTL_II_DCI	1.5	0.75	Split
IBUF	√	√	HSTL_II_DCI_18	1.8	0.75	Split
IBUF	√	√	HSTL_III	1.5	0.90	None
IBUF	√	√	HSTL_III_18	1.8	0.90	None
IBUF	√	√	HSTL_III_DCI	1.5	0.90	Split
IBUF	√	√	HSTL_III_DCI_18	1.8	0.90	Split
IBUF	√	√	HSTL_IV	N / A	0.90	None
IBUF	√	√	HSTL_IV_18	1.8	0.90	None
IBUF	√	√	HSTL_IV_DCI	1.5	0.90	Split
IBUF	√	√	HSTL_IV_DCI_18	1.8	0.90	Split
IBUF	√	√	LVCMOS15	1.5	N / A	None

Virtex-II, Virtex-II Pro IBUF_*selectIO* IOSTANDARD Attributes

Component[a]	Virtex-II	Virtex-II Pro	IOSTANDARD (Attribute Value)	Input VCCO	VREF	Terminate Type (Virtex-II only)
IBUF	√	√	LVCMOS18	1.8	N/A	None
IBUF	√	√	LVCMOS2	2.0	N/A	None
IBUF	√	√	LVCMOS25	2.5	N/A	None
IBUF	√		LVCMOS33	3.3	N/A	None
IBUF	√	√	LVDCI_15	N/A	N/A	Driver
IBUF	√	√	LVDCI_18	N/A	N/A	Driver
IBUF	√	√	LVDCI_25	N/A	N/A	Driver
IBUF	√	√	LVDCI_33	N/A	N/A	Driver
IBUF	√	√	LVDCI_DV2_15	N/A	N/A	Driver
IBUF	√	√	LVDCI_DV2_18	N/A	N/A	Driver
IBUF	√	√	LVDCI_DV2_25	N/A	N/A	Driver
IBUF	√		LVDCI_DV2_33	N/A	N/A	Driver
IBUF	√		LVTTL (default)	3.3	N/A	None
IBUF	√	√	PCI33_3	3.3	N/A	None
IBUF	√	√	PCI66_3	3.3	N/A	None
IBUF	√		PCIX	3.3	N/A	None
IBUF	√	√	SSTL18_I	1.8	0.9	None
IBUF	√	√	SSTL18_I_DCI	1.8	0.9	Split
IBUF	√	√	SSTL18_II	1.8	0.9	None
IBUF	√	√	SSTL18_II_DCI	1.8	0.9	Split
IBUF	√	√	SSTL2_I	N/A	1.25	None
IBUF	√	√	SSTL2_I_DCI	2.5	1.25	Split
IBUF	√	√	SSTL2_II	N/A	1.25	None
IBUF	√	√	SSTL2_II_DCI	2.5	1.25	Split
IBUF	√		SSTL3_I	N/A	1.50	None
IBUF	√		SSTL3_I_DCI	3.3	1.25	Split
IBUF	√		SSTL3_II	N/A	1.50	None
IBUF	√		SSTL3_II_DCI	3.3	1.25	Split

[a]Attach an IOSTANDARD attribute to an IBUF and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the input for the I/O standard associated with that value.

SelectI/O Usage Rules

The Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro architectures include a versatile SelectI/O interface to multiple voltage and drive standards. To select an I/O standard, you must choose the appropriate component from the library or add an IOSTANDARD attribute to the appropriate buffer component. For example, for an input buffer that uses the GTL standard, you would choose the IBUF_GTL component or choose the IBUF component and attach the IOSTANDARD=GTL attribute to it.

See the following sections for information on the various input/output buffer components and attributes available to implement the desired standard: "IBUF_selectIO", "IBUFG, IBUFG_selectIO", "IOBUF, IOBUF_selectIO", "OBUF_selectIO", and "OBUFT_selectIO" sections.

The hardware implementation of the various I/O standards requires that certain usage rules be followed. Each I/O standard has voltage source requirements for input reference (VREF), output drive (VCCO), or both. In addition, Virtex-II and Virtex-II Pro have terminate type requirements. Each Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro device has eight banks (two on each edge). Each bank has voltage sources shared by all I/O in the bank. Therefore, in a particular bank, the voltage source (for either input or output) must be of the same type.

For Spartan-II, Spartan-IIE, Virtex, and Virtex-E, see the "Virtex, Virtex-E, Spartan-II, and Spartan-IIE Banking Rules" section. Virtex-E follows the same banking rules as Virtex with a few additions. See "Additional Banking Rules for Virtex-E and Spartan-IIE" section for the additional Virtex-E rules. Virtex-II and Virtex-II Pro have their own set of banking rules. See the "Virtex-II, Virtex-II Pro Banking Rules" section for Virtex-II and Virtex-II Pro rules.

Virtex, Virtex-E, Spartan-II, and Spartan-IIE Banking Rules

The hardware implementation of the various I/O standards requires that certain usage rules be followed. As shown in the following table, each I/O standard has voltage source requirements for input reference (VREF), output drive (VCCO), or both. Each Spartan-II, Spartan-IIE, Virtex, and Virtex-E device has eight banks (two on each edge). Each bank has voltage sources shared by all I/O in the bank. Therefore, in a particular bank, the voltage source (for either input or output) must be of the same type. The Input Banking (VREF) Rules section and the Output Banking (VCCO) Rules section below summarize the SelectI/O component usage rules based on the hardware implementation.

I/O Standards Supported in Virtex, Virtex-E, Spartan-II, and Spartan-IIE

I/O Standard	Application	Description	Output VCCO	Input VCCO	VREF
AGP	Graphics	Advanced graphics port	3.3	N / A	1.32
CTT	Memory	Center tap terminated	3.3	N / A	1.50
LVTTL	General Purpose	Low voltage transistor-transistor logic	3.3	3.3*	N / A
LVCMOS2	General Purpose	Low voltage complementary metal-oxide semiconductor	2.5	2.5*	N / A
PCI33_3	PCI	Peripheral component interface (33MHz 3.3V)	3.3	3.3*	N / A
PCI33_5	PCI	Peripheral component interface (33MHZ 5.0V) PCI33_5 is not supported for Virtex-E or Spartan-IIE.	3.3	N / A	N / A
PCI66_3	PCI	Peripheral component interface (66MHz 3.3V)	3.3	3.3*	N / A
GTL	Backplane	Gunning transceiver logic interface (to processors or backplane driver)	N / A	N / A	0.80
GTL+ (GTLP)	Backplane	Gunning transceiver logic interface Plus	N / A	N / A	1.00
HSTL_I	Hitachi SRAM	High Speed transceiver logic	1.5	N / A	0.75
HSTL_III	Hitachi SRAM	High Speed transceiver logic	1.5	N / A	0.90
HSTL_IV	Hitachi SRAM	High Speed transceiver logic	1.5	N / A	0.90
SSTL2_I	Synchronous DRAM	Stub-series terminated logic interface for SDRAM	2.5	N / A	1.25
SSTL2_II	Synchronous DRAM	Stub-series terminated logic interface for SDRAM	2.5	N / A	1.25
SSTL3_I	Synchronous DRAM	Stub-series terminated logic interface for SDRAM	3.3	N / A	1.50
SSTL3_II	Synchronous DRAM	Stub-series terminated logic interface for SDRAM	3.3	N / A	1.50

*Only LVTTL, LVCMOS, and PCI need Input VCCO in Virtex-E and Spartan-IIE parts.

Input Banking (VREF) Rules

The low-voltage I/O standards that have a differential amplifier input require a voltage reference input (VREF). The VREF voltage source is provided as an external signal to the chip.

- Any input buffer component that does not require a VREF source (LVTTL, LVCMOS2, PCI) can be placed in any bank.
- All input buffer components that require a VREF source (GTL*, HSTL*, SSTL*, CTT, AGP) must be of the same I/O standard in a particular bank. For example, IBUF_SSTL2_I and IBUFG_SSTL2_I are compatible since they are the same I/O standard (SSTL2_I).
- If the bank contains any input buffer component that requires a VREF source, the following conditions apply.
  - One or more VREF sources must be connected to the bank via an IOB.
  - The number of VREF sources is dependent on the device and package.
  - The locations of the VREF sources are fixed for each device/package.
  - All VREF sources must be used in that bank.
- If the bank contains no input buffer component that requires a VREF source, the IOBs for VREF sources can be used for general I/O.
- Output buffer components of any type can be placed in the bank.

Output Banking (VCCO) Rules

Because Virtex, Virtex-E, Spartan-II, and Spartan-IIE have multiple low-voltage standards, some control is required over the distribution of VCCO, the drive source voltage for output pins. To provide for maximum flexibility, the output pins are banked. In comparison to the VREF sources described above, the VCCO voltage sources are dedicated pins on the device and do not consume valuable IOBs.

- Any output buffer component that does not require a VCCO source (GTL, GTL+) can be placed in any bank.
- To be placed in a particular bank, all output buffer components that require VCCO must have the same supply voltage (VCCO). For example, OBUF_SSTL3_I and OBUF_PCI33_3 are compatible in the same output bank since VCCO=3.3 for both.
- Input buffer components of any type can be placed in the bank.
- The configuration pins on a Virtex, Virtex-E, Spartan-II, and Spartan-IIE device are on the right side of the chip. When configuring the device through a serial PROM, the user is required to use a VCCO of 3.3V in the two banks on the right hand side of the chip. If the user is not configuring the device through a serial PROM, the VCCO requirement is dependent upon the configuration source.

Banking Rules for OBUFT_*selectIO* with KEEPER

If a KEEPER symbol is attached to an OBUFT_*selectIO* component (3-state output buffer) for an I/O standard that requires a VREF (for example, OBUFT_GTL, OBUFT_SSTL3_I), then the OBUFT_*selectIO* component follows the same rules as an IOBUF_*selectIO* component for the same standard. It must follow both the input banking and output banking rules. The KEEPER element requires that the VREF be properly driven.

Additional Banking Rules for Virtex-E and Spartan-IIE

The Virtex-E and Spartan-IIE architectures requires the same banking rules as described in the “Virtex, Virtex-E, Spartan-II, and Spartan-IIE Banking Rules” section.

Additional I/O standards are supported as indicated in the following table.

Additional I/O Standards Supported in Virtex-E and Spartan-IIE

I/O Standard	Application	Description	Output VCCO	Input VCCO	VREF
Single Ended:					
LVCMOS18	General Purpose	Low voltage complementary metal-oxide semiconductor	1.8	1.8	N/A
Differential Signaling:					
LVDS	Point-to-point or multi-drop backplanes, high noise immunity	Low voltage differential signal	2.5	2.5	N/A
LVPECL	High performance clocking, backplanes, differential 100MHz+ clocking, optical transceiver, high speed networking and mixed-signal interfacing	Low voltage positive emitter couple logic	2.5	2.5	N/A

Virtex-II, Virtex-II Pro Banking Rules

The hardware implementation of the various I/O standards requires that certain usage rules be followed. Virtex-II and Virtex-II Pro devices have eight banks (two on each edge), numbered from 0 through 7. Each bank has voltage sources shared by all I/O in the bank.

As shown in Table 6-5, for the I/O standards supported in Virtex-II and Virtex-II Pro, each I/O standard has voltage source requirements for input reference (VREF), output drive (VCCO), and Terminate Type.

I/O Standards Supported In Virtex-II and Virtex-II Pro

I/O Standard	Application	Description	Output VCCO	Input VCCO	VREF	Terminate Type
Single-Ended:						
AGP	Graphics	Advanced graphics port	3.3	N / A	1.32	None
GTL	Memory	Gunning transceiver logic interface (to processors or backplace driver)	N / A	N / A	0.80	None
GTL_DCI	Memory	Gunning transceiver logic interface with on-chip Digital Controlled Impedance	1.2	1.2	0.80	Single
GTL+ (GTLP)	Memory	Gunning transceiver logic interface Plus	N / A	N / A	1.00	None
GTLP_DCI	Memory	Gunning transceiver logic interface with on-chip Digital Controlled Impedance	1.5	1.5	1.00	Single
HSTL_I	Hitachi SRAM	High Speed transceiver logic	1.5	N / A	0.75	None
HSTL_I__DCI	Hitachi SRAM	High Speed transceiver logic interface with on-chip Digital Controlled Impedance	1.5	1.5	0.75	Split
HSTL_II	Hitachi SRAM	High Speed transceiver logic	1.5	N / A	0.75	None
HSTL_II__DCI	Hitachi SRAM	High Speed transceiver logic interface with on-chip Digital Controlled Impedance	1.5	1.5	0.75	Split
HSTL_III	Hitachi SRAM	High Speed transceiver logic	1.5	1.5	0.90	None
HSTL_III__DCI	Hitachi SRAM	High Speed transceiver logic interface with on-chip Digital Controlled Impedance	1.5	1.5	0.90	Split
HSTL_IV	Hitachi SRAM	High Speed transceiver logic	1.5	N / A	0.90	None
HSTL_IV__DCI	Hitachi SRAM	High Speed transceiver logic interface with on-chip Digital Controlled Impedance	1.5	1.5	0.90	Split
LVCMOS15[a]	General Purpose	Low voltage complementary metal-oxide semiconductor	1.5	1.5	N / A	None
LVCMOS18[a]	General Purpose	Low voltage complementary metal-oxide semiconductor	1.8	1.8	N / A	None
LVCMOS25[a]	General Purpose	Low voltage complementary metal-oxide semiconductor	2.5	2.5	N / A	None
LVCMOS33[a]	General Purpose	Low voltage complementary metal-oxide semiconductor	3.3	3.3	N / A	None
LVDCI_15	General Purpose	Low voltage complementary metal-oxide semiconductor with on-chip Digital Controlled Impedance	1.5	N / A	N / A	Driver
LVDCI_18	General Purpose	Low voltage complementary metal-oxide semiconductor with on-chip Digital Controlled Impedance	1.8	N / A	N / A	Driver

I/O Standards Supported In Virtex-II and Virtex-II Pro

I/O Standard	Application	Description	Output VCCO	Input VCCO	VREF	Terminate Type
LVDCI_25	General Purpose	Low voltage complementary metal-oxide semiconductor with on-chip Digital Controlled Impedance	2.5	N / A	N / A	Driver
LVDCI_33	General Purpose	Low voltage complementary metal-oxide semiconductor with on-chip Digital Controlled Impedance	3.3	N / A	N / A	Driver
LVDCI_DV2_15	General Purpose	Low voltage complementary metal-oxide semiconductor with on-chip Digital Controlled Impedance	1.5	N / A	N / A	Driver
LVDCI_DV2_18	General Purpose	Low voltage complementary metal-oxide semiconductor with on-chip Digital Controlled Impedance	1.8	N / A	N / A	Driver
LVDCI_DV2_25	General Purpose	Low voltage complementary metal-oxide semiconductor with on-chip Digital Controlled Impedance	2.5	N / A	N / A	Driver
LVDCI_DV2_33	General Purpose	Low voltage complementary metal-oxide semiconductor with on-chip Digital Controlled Impedance	3.3	N / A	N / A	Driver
LVTTL[a]	General Purpose	Low voltage transistor-transistor logic	3.3	3.3	N / A	None
LVTTL_33[a]	General Purpose	Low voltage transistor-transistor logic	3.3	3.3	N / A	None
PCI33_3	PCI	Peripheral component interface (33MHz 3.3V)	3.3	3.3	N / A	None
PCI66_3	PCI	Peripheral component interface (66MHz 3.3V)	3.3	3.3	N / A	None
PCIX	PCI	Peripheral component interface	3.3	3.3	N / A	None
SSTL2_I	Synchronous DRAM	Stub-series terminated logic interface for SDRAM	2.5	N / A	1.25	None
SSTL2_I_DCI	Synchronous DRAM	Stub-series terminated logic interface for SDRAM with on-chip Digital Controlled Impedance	2.5	2.5	1.25	Split
SSTL2_II	Synchronous DRAM	Stub-series terminated logic interface for SDRAM	2.5	N / A	1.25	None
SSTL2_II_DCI	Synchronous DRAM	Stub-series terminated logic interface for SDRAM with on-chip Digital Controlled Impedance	2.5	2.5	1.25	Split
SSTL3_I	Synchronous DRAM	Stub-series terminated logic interface for SDRAM	3.3	N / A	1.50	None

I/O Standards Supported In Virtex-II and Virtex-II Pro

I/O Standard	Application	Description	Output VCCO	Input VCCO	VREF	Terminate Type
SSTL3_I_DCI	Synchronous DRAM	Stub-series terminated logic interface for SDRAM with on-chip Digital Controlled Impedance	3.3	3.3	1.5	Split
SSTL3_II	Synchronous DRAM	Stub-series terminated logic interface for SDRAM	3.3	N/A	1.50	None
SSTL3_II_DCI	Synchronous DRAM	Stub-series terminated logic interface for SDRAM with on-chip Digital Controlled Impedance	3.3	3.3	1.5	Split

Notes:
[a]LVTTL, LVCMOS15, LVCMOS18, and LVCMOS25 also require DRIVE and SLEW (FAST or Slow) attributes.

The following rules apply for using the various IO standards with Virtex-II or Virtex-II Pro:

- In any particular Virtex-II or Virtex-II Pro I/O bank, the voltage sources (both input and output) must be compatible. That is, they must have either the same voltage or an undefined (N/A) voltage.
- VREF, VCCO input, and VCCO output must be compatible within an I/O bank.
- In addition, to VREF and VCCO compatibility, the terminate type I/O standards must be compatible within the bank.

  For terminate type compatibility, the following rules apply:

  - Only one I/O buffer with terminate type of SINGLE can be in a particular bank.
  - Only one I/O buffer with terminate type of SPLIT can be in a particular bank.
  - Multiple I/O buffers with NONE and DRIVER terminate types can be in a particular bank.
  - SPLIT and SINGLE can co-exist in the same bank.
  - NONE and DRIVER types can co-exist with SPLIT and SINGLE types.
- The bottom edge of a Virtex-II or Virtex-II Pro device is set for 3.3V. Therefore, on the bottom edge of the device, VCCO output and VCCO input must be 3.3V or N/A.
- To place an I/O buffer that requires a VREF in a bank, the reserved VREF sites in that bank must be empty.
- To place an I/O buffer that has a terminate type of SINGLE, SPLIT, or DRIVER in a bank, the reserved VREF sites in that bank must be empty.
- For Virtex-II and Virtex-II Pro, differential signaling standards apply to IBUFDS, IBUFGDS, OBUFDS, and OBUFTDS only (not IBUF or OBUF).

The following table summarizes the values that you need to check for compatibility for each combination of I/O buffer programming (input, output, or bidirectional buffer). For example, the table shows that if you configure an output buffer as LVCMOS25, which has an output voltage of 2.5V, and an input buffer as LVCMOS15, which as an input voltage of 1.5V, the Out/In Voltage is checked. Because they have different voltages, this combination would not be allowed in a particular I/O bank.

IOB Programming Combinations		VREF	Output VCCO	Input VCCO	Out/In Voltage
Input	Input	Check		Check	
Input	Output				Check
Input	Bidirectional	Check		Check	Check
Output	Input				Check
Output	Output		Check		
Output	Bidirectional		Check		Check
Bidirectional	Input	Check		Check	Check
Bidirectional	Output		Check		Check
Bidirectional	Bidirectional	Check	Check	Check	Check

Usage

The recommended usage for IBUF_selectIO is to allow the IBUFs be inferred and apply the IOSTANDARD constraint to the input in either the UCF or in the HDL code. IBUF_selectIO can also be instantiated if necessary.

VHDL Instantiation Template

```
-- Component Declaration for IBUF_selectIO should be placed
-- after architecture statement but before begin keyword

component IBUF_selectIO
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for IBUF_selectIO
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IBUF_selectIO should be placed
-- in architecture after the begin keyword

IBUF_selectIO_INSTANCE_NAME : IBUF_selectIO
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
IBUF_selectIO instance_name (.O (user_O),
                             .I (user_I));
```

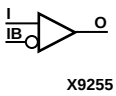
Commonly Used Constraints

IOBDELAY, IOSTANDARD

IBUFDS

Differential Signaling Input Buffer with Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



X9255

IBUFDS is an input buffer that supports low-voltage, differential signaling. In IBUFDS, a design level interface signal is represented as two distinct ports (I and IB), one deemed the "master" and the other the "slave." The master and the slave are opposite phases of the same logical signal (for example, MYNET and MYNETB).

Inputs		Outputs
I	IB	O
0	0	X
0	1	0
1	0	1
1	1	X

The IOSTANDARD attribute values listed in the following table can be applied to an IBUFDS component to provide selectIO interface capability.

Component	IOSTANDARD (Attribute Value)	Virtex-II	Virtex-II Pro	Input VCCO	VREF	Terminate Type
IBUFDS[a]	BLVDS_25	√	√	2.5	N/A	None
IBUFDS[a]	LDT_25	√	√	2.5	N/A	None
IBUFDS[a]	LDT_25_DCI	√	√	2.5	N/A	None
IBUFDS[a]	LVDS_25 (default)	√	√	N/A	N/A	None
IBUFDS[a]	LVDS_25_DCI	√	√	N/A	N/A	None
IBUFDS[a]	LVDS_33	√	√	N/A	N/A	None
IBUFDS[a]	LVDS_33_DCI	√	√	N/A	N/A	None
IBUFDS[a]	LVDSEXT_25	√	√	N/A	N/A	None
IBUFDS[a]	LVDSEXT_25_DCI	√	√	N/A	N/A	None
IBUFDS[a]	LVDSEXT_33	√	√	N/A	N/A	None
IBUFDS[a]	LVDSEXT_33_DCI	√	√	N/A	N/A	None
IBUFDS[a]	LVPECL_25		√	N/A	N/A	None
IBUFDS[a]	LVPECL_33	√		N/A	N/A	None
IBUFDS[a]	ULVDS_25	√	√	N/A	N/A	None
IBUFDS[a]	ULVDS_25_DCI	√	√	N/A	N/A	None

[a] A separate SelectI/O component is not provided. Attach an IOSTANDARD attribute to an IBUFDS and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the inputs for the I/O standard associated with that value.

Usage

For HDL, this design element is supported for instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for IBUFDS should be placed
-- after architecture statement but before begin keyword

component IBUFDS
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC;
         IB : in STD_ULOGIC);
end component;

-- Component Attribute specification for IBUFDS
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IBUFDS should be placed
-- in architecture after the begin keyword

IBUFDS_INSTANCE_NAME : IBUFDS
      port map  (O => user_O,
                 I => user_I,
                 IB => user_IB);
```

Verilog Instantiation Template

```
IBUFDS instance_name (.O (user_O),
                      .I (user_I),
                      .IB (user_IB));
```

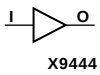
Commonly Used Constraints

IOSTANDARD, IOBDELAY

IBUFG, IBUFG_*selectIO*

Dedicated Input Buffer with Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



For Virtex, Virtex-E, Spartan-II, and Spartan-IIE, IBUFG and its selectIO variants (listed in the "Components" column in the table below) are dedicated input buffers for connecting to the clock buffer BUFG or CLKDLL. The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. For example, IBUFG_SSTL3_II is a single input buffer that uses the SSTL3_II I/O-signaling standard. You can attach an IOSTANDARD attribute to an IBUFG instance instead of using an IBUFG_*selectIO* component. Check marks (√) in the "Spartan-II, Virtex" and "Spartan-IIE, Virtex-E" columns indicate the components and IOSTANDARD attribute values available for those architectures.

The Xilinx implementation software converts each BUFG to an appropriate type of global buffer for the target PLD device. The IBUFG input can only be driven by the global clock pins. The IBUFG output can drive CLKIN of a DLL/DCM, BUFG, or user logic. IBUFG can be routed to user logic and does not have to be routed to a DLL. The IBUFG can only be driven by an IPAD.

The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffers. See the "SelectI/O Usage Rules" section included in the IBUF_*selectIO* section for information on using these components and the IOSTANDARD attributes.

Spartan-II, Spartan-IIE, Virtex, and Virtex-E IBUFG_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Input VCCO	VREF
IBUFG	√	√	(defaults to LVTTL)	3.3	N/A
IBUFG_AGP	√	√	AGP	N/A	1.32
IBUFG_CTT	√	√	CTT	N/A	1.50
IBUFG_GTL	√	√	GTL	N/A	0.80
IBUFG_GTLP	√	√	GTLP	N/A	1.00
IBUFG_HSTL_I	√	√	HSTL_I	N/A	0.75
IBUFG_HSTL_III	√	√	HSTL_III	1.5	0.90
IBUFG_HSTL_IV	√	√	HSTL_IV	N/A	0.90
IBUFG_LVCMOS2	√	√	LVCMOS2	2.5	N/A
IBUFG_LVCMOS18		√	LVCMOS18	1.8	N/A
IBUFG_LVDS		√	LVDS	N/A	N/A
IBUFG_LVPECL		√	LVPECL	N/A	N/A
IBUFG_PCI33_3	√	√	PCI33_3	3.3	N/A
IBUFG_PCI33_5	√	√	PCI33_5	N/A	N/A
IBUFG_PCI66_3	√	√	PCI66_3	3.3	N/A

Spartan-II, Spartan-IIE, Virtex, and Virtex-E IBUFG_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Input VCCO	VREF
IBUFG_PCIX66_3		√	PCIX66_3	3.3	N/A
IBUFG_SSTL2_I	√	√	SSTL2_I	N/A	1.25
IBUFG_SSTL2_II	√	√	SSTL2_II	N/A	1.25
IBUFG_SSTL3_I	√	√	SSTL3_I	N/A	1.50
IBUFG_SSTL3_II	√	√	SSTL3_II	N/A	1.50

The Virtex-II and Virtex-II Pro library includes some IBUFG_*selectIO* components for compatibility with older, existing designs and other architectures. For new Virtex-II and Virtex-II Pro designs, however, the recommended method for using IBUFG selectI/O buffers is to attach an IOSTANDARD attribute to an IBUFG component. For example, attach IOSTANDARD=GTLP to an IBUFG instead of using the IBUFG_GTLP component for new Virtex-II and Virtex-II Pro designs. The IOSTANDARD attributes that can be attached to an IBUFG component are listed in the "IOSTANDARD (Attribute Value)" column in the following table. See the "SelectI/O Usage Rules" section for information on using these IOSTANDARD attributes.

Virtex-II, Virtex-II Pro IBUFG_*selectIO* IOSTANDARD Attributes

Component[a]	Virtex-II	Virtex-II Pro	IOSTANDARD (Attribute Value)	Input VCCO	VREF	Terminate Type
IBUFG	√		AGP	N/A	1.32	None
IBUFG	√	√	GTL	N/A	0.80	None
IBUFG	√	√	GTL_DCI	1.2	0.80	Single
IBUFG	√	√	GTLP	N/A	1.00	None
IBUFG	√	√	GTLP_DCI	1.5	1.00	Single
IBUFG	√	√	HSTL_I	N/A	0.75	None
IBUFG	√	√	HSTL_I_18	1.8	0.75	None
IBUFG	√	√	HSTL_I_DCI	1.5	0.75	Split
IBUFG	√	√	HSTL_I_DCI_18	1.8	0.75	Split
IBUFG	√	√	HSTL_II	N/A	0.75	None
IBUFG	√	√	HSTL_II_18	1.8	0.75	None
IBUFG	√	√	HSTL_II_DCI	1.5	0.75	Split
IBUFG	√	√	HSTL_II_DCI_18	1.8	0.75	Split
IBUFG	√	√	HSTL_III	1.5	0.90	None
IBUFG	√	√	HSTL_III_18	1.8	0.90	None
IBUFG	√	√	HSTL_III_DCI	1.5	0.90	Split
IBUFG	√	√	HSTL_III_DCI_18	1.8	0.90	Split
IBUFG	√	√	HSTL_IV	N/A	0.90	None
IBUFG	√	√	HSTL_IV_18	1.8	0.90	None
IBUFG	√	√	HSTL_IV_DCI	1.5	0.90	Split
IBUFG	√	√	HSTL_IV_DCI_18	1.8	0.90	Split
IBUFG	√	√	LVCMOS15	1.5	N/A	None

Virtex-II, Virtex-II Pro IBUFG_*selectIO* IOSTANDARD Attributes

Component[a]	Virtex-II	Virtex-II Pro	IOSTANDARD (Attribute Value)	Input VCCO	VREF	Terminate Type
IBUFG	√	√	LVCMOS18	1.8	N/A	None
IBUFG	√	√	LVCMOS2	2.0	N/A	None
IBUFG	√	√	LVCMOS25	2.5	N/A	None
IBUFG	√		LVCMOS33	3.3	N/A	None
IBUFG	√	√	LVDCI_15	N/A	N/A	Driver
IBUFG	√	√	LVDCI_18	N/A	N/A	Driver
IBUFG	√	√	LVDCI_25	N/A	N/A	Driver
IBUFG	√	√	LVDCI_33	N/A	N/A	Driver
IBUFG	√	√	LVDCI_DV2_15	N/A	N/A	Driver
IBUFG	√	√	LVDCI_DV2_18	N/A	N/A	Driver
IBUFG	√	√	LVDCI_DV2_25	N/A	N/A	Driver
IBUFG	√		LVDCI_DV2_33	N/A	N/A	Driver
IBUFG	√		LVTTL (default)	3.3	N/A	None
IBUFG	√	√	PCI33_3	3.3	N/A	None
IBUFG	√	√	PCI66_3	3.3	N/A	None
IBUFG	√		PCIX	3.3	N/A	None
IBUFG	√	√	SSTL18_I	1.8	0.9	None
IBUFG	√	√	SSTL18_I_DCI	1.8	0.9	Split
IBUFG	√	√	SSTL18_II	1.8	0.9	None
IBUFG	√	√	SSTL18_II_DCI	1.8	0.9	Split
IBUFG	√	√	SSTL2_I	N/A	1.25	None
IBUFG	√	√	SSTL2_I_DCI	2.5	1.25	Split
IBUFG	√	√	SSTL2_II	N/A	1.25	None
IBUFG	√	√	SSTL2_II_DCI	2.5	1.25	Split
IBUFG	√		SSTL3_I	N/A	1.50	None
IBUFG	√		SSTL3_I_DCI	3.3	1.25	Split
IBUFG	√		SSTL3_II	N/A	1.50	None
IBUFG	√		SSTL3_II_DCI	3.3	1.25	Split

[a]Attach an IOSTANDARD attribute to an IBUFG and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the input for the I/O standard associated with that value.

Usage

This design element is supported for schematic and instantiation. Synthesis tools usually infer a BUFGP on any clock net. If there are more clock nets than BUFGPs, the synthesis tool usually instantiates BUFGPs for the clocks that are most utilized. The BUFGP contains both a BUFG and an IBUFG.

VHDL Instantiation Template

```
-- Component Declaration for IBUFG should be placed
-- after architecture statement but before begin keyword

component IBUFG
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for IBUFG
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IBUFG should be placed
-- in architecture after the begin keyword

IBUFG_INSTANCE_NAME : IBUFG
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
IBUFG instance_name (.O (user_O),
                     .I (user_I));
```

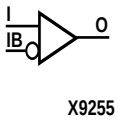
Commonly Used Constraints

IOSTANDARD, IOBDELAY

IBUFGDS

Dedicated Differential Signaling Input Buffer with Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



X9255

IBUFGDS is a dedicated differential signaling input buffer for connection to the clock buffer (BUFG) or DCM. In IBUFGDS, a design level interface signal is represented as two distinct ports (I and IB), one deemed the "master" and the other the "slave." The master and the slave are opposite phases of the same logical signal (for example, MYNET and MYNETB).

Inputs		Outputs
I	IB	O
0	0	X
0	1	0
1	0	1
1	1	X

The IOSTANDARD attribute values listed in the following table can be applied to an IBUFGDS component to provide SelectI/O interface capability. See the *Xilinx Constraints Guide* for information about using these attributes.

Component	IOSTANDARD (Attribute Value)	Virtex-II	Virtex-II Pro	Input VCCO	VREF	Terminate Type
IBUFGDS[a]	BLVDS_25	√	√	N/A	N/A	None
IBUFGDS[a]	LDT_25	√	√	N/A	N/A	None
IBUFGDS[a]	LDT_25_DCI	√	√	N/A	N/A	None
IBUFGDS[a]	LVDS_25 (default)	√	√	N/A	N/A	None
IBUFGDS[a]	LVDS_25_DCI	√	√	N/A	N/A	None
IBUFGDS[a]	LVDS_33 *	√	√	N/A	N/A	None
IBUFGDS[a]	LVDS_33_DCI *	√	√	N/A	N/A	None
IBUFGDS[a]	LVDSEXT_25	√	√	N/A	N/A	None
IBUFGDS[a]	LVDSEXT_25_DCI	√	√	N/A	N/A	None
IBUFGDS[a]	LVDSEXT_33*	√	√	N/A	N/A	None
IBUFGDS[a]	LVDSEXT_33_DCI*	√	√	N/A	N/A	None
IBUFGDS[a]	LVPECL_25*		√	N/A	N/A	None
IBUFGDS[a]	LVPECL_33*	√		N/A	N/A	None
IBUFGDS[a]	ULVDS_25	√	√	N/A	N/A	None

Component	IOSTANDARD (Attribute Value)	Virtex-II	Virtex-II Pro	Input VCCO	VREF	Terminate Type
IBUFGDS[a]	ULVDS_25_DCI	√	√	N/A	N/A	None

[a]A separate SelectI/O component is not provided. Attach an IOSTANDARD attribute to an IBUFGDS and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the inputs for the I/O standard associated with that value.

Usage

For HDL, this design element is supported for instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for IBUFGDS should be placed
-- after architecture statement but before begin keyword

component IBUFGDS
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC;
         IB : in STD_ULOGIC);
end component;

-- Component Attribute specification for IBUFGDS
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IBUFGDS should be placed
-- in architecture after the begin keyword

IBUFGDS_INSTANCE_NAME : IBUFGDS
      port map  (O => user_O,
                 I => user_I,
                 IB => user_IB);
```

Verilog Instantiation Template

```
IBUFGDS instance_name (.O (user_O),
                       .I (user_I),
                       .IB (user_IB));
```

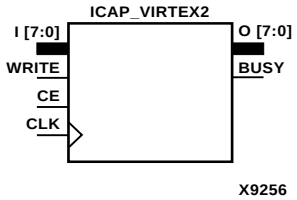
Commonly Used Constraints

IOSTANDARD, IOBDELAY

ICAP_VIRTEX2

User Interface to Virtex-II and Virtex-II Pro Internal Configuration Access Port

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



ICAP_VIRTEX2 provides user access to the Virtex-II and Virtex-II Pro internal configuration access port (ICAP).

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for ICAP_VIRTEX2 should be placed
-- after architecture statement but before begin keyword

component ICAP_VIRTEX2
   port (BUSY  : out STD_ULOGIC;
         O     : out STD_LOGIC_VECTOR (7 downto 0);
         CE    : in STD_ULOGIC;
         CLK   : in STD_ULOGIC;
         I     : in STD_LOGIC_VECTOR (7 downto 0);
         WRITE : in STD_ULOGIC);
end component;

-- Component Attribute specification for ICAP_VIRTEX2
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ICAP_VIRTEX2 should be placed
-- in architecture after the begin keyword

ICAP_VIRTEX2_INSTANCE_NAME : ICAP_VIRTEX2
      port map  (BUSY => user_BUSY,
                 O    => user_O,
                 CE   => user_CE,
                 CLK  => user_CLK,
                 I    => user_I,
                 WRITE=> user_WRITE);
```

Verilog Instantiation Template

```
ICAP_VIRTEX2 instance_name (.BUSY (user_BUSY),
                            .O (user_O),
                            .CE (user_CE),
                            .CLK (user_CLK),
                            .I (user_I),
                            .WRITE (user_WRITE));
```

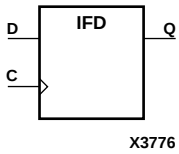
Commonly Used Constraints

None

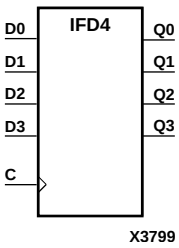
IFD, 4, 8, 16

Single- and Multiple-Input D Flip-Flops

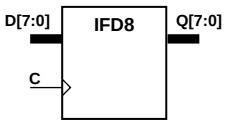
Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
IFD	Macro	Macro	Macro	Macro	Macro	Macro
IFD4, IFD8, IFD16	Macro	Macro	Macro	Macro	Macro	Macro



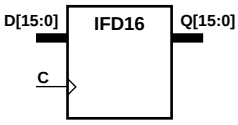
X3776



X3799



X3811



X3833

The IFD D-type flip-flop is contained in an input / output block (IOB), except for XC9500 / XV / XL, CoolRunner XPLA3. The input (D) of the flip-flop is connected to an IPAD or an IOPAD (without using an IBUF). The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

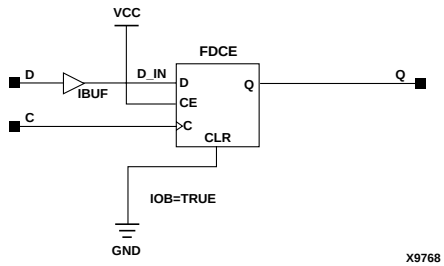
The flip-flops are asynchronously cleared with Low outputs when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

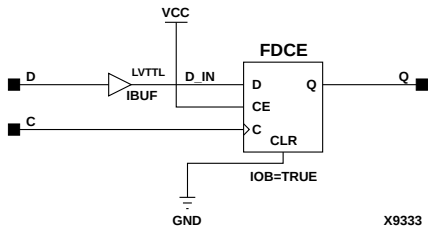
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1

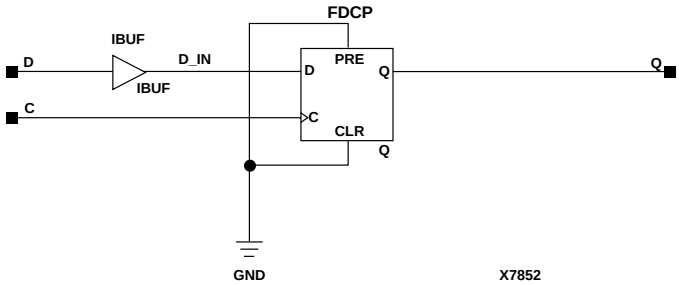


X9768

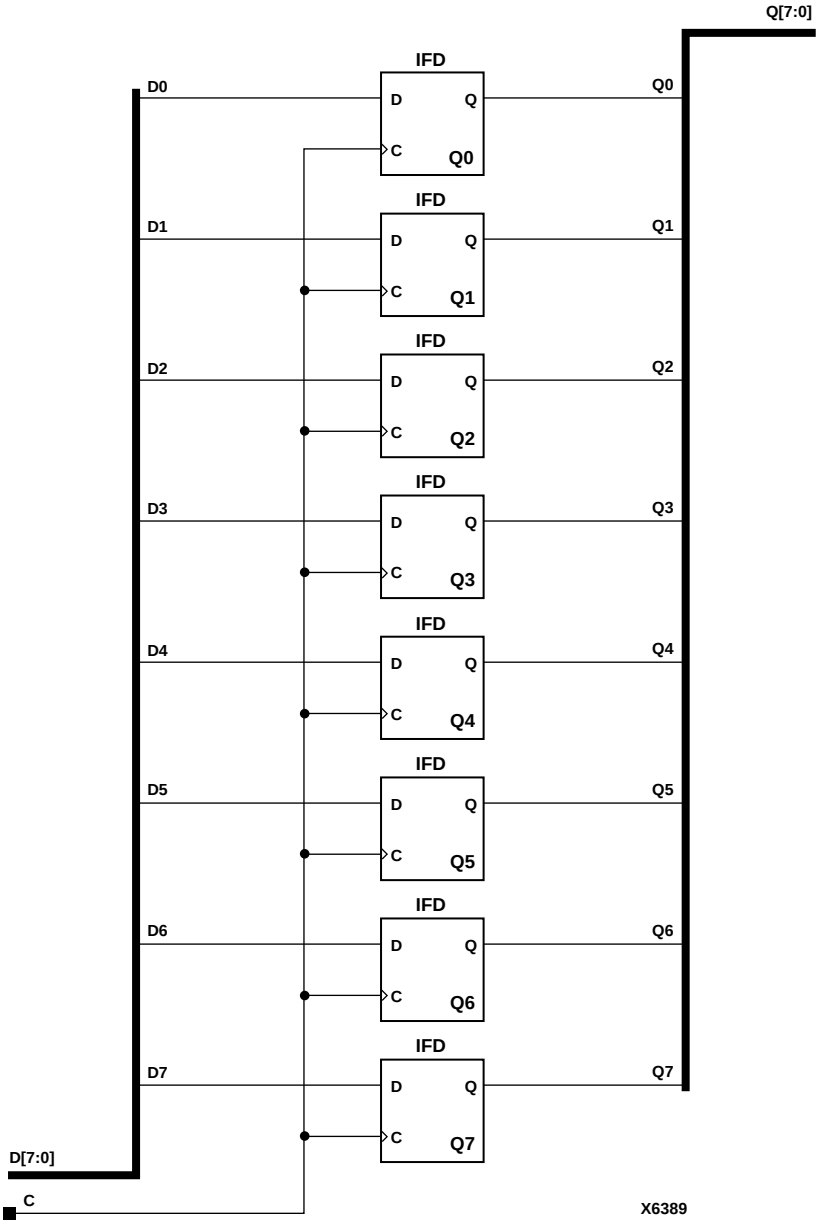
IFD Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFD Implementation Virtex-II, Virtex-II Pro



IFD Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



IFD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

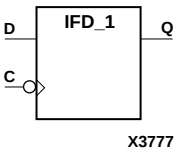
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFD, you would infer an FD and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

IFD_1

Input D Flip-Flop with Inverted Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



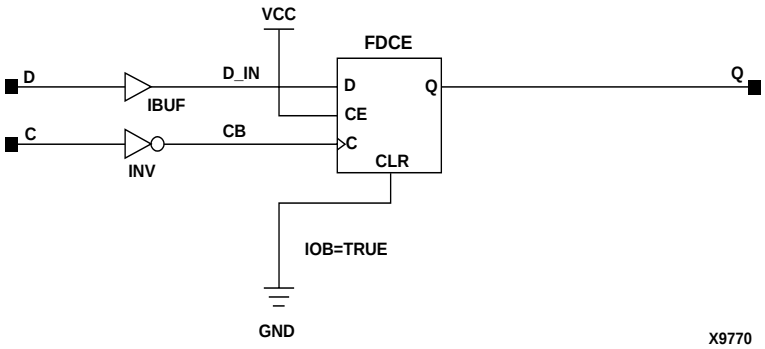
The IFD_1 D-type flip-flop is contained in an input / output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input also provides data input for the flip-flop, which synchronizes data entering the chip. The D input data is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously cleared with Low output when power is applied.

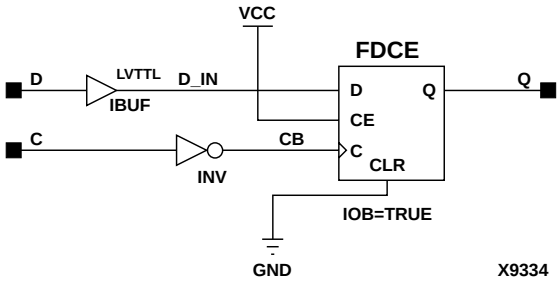
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1



IFD_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFD_1 Implementation Virtex-II, Virtex-II Pro

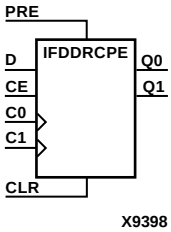
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFD_1, you would infer an FD_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

IFDDRCPE

Dual Data Rate Input D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



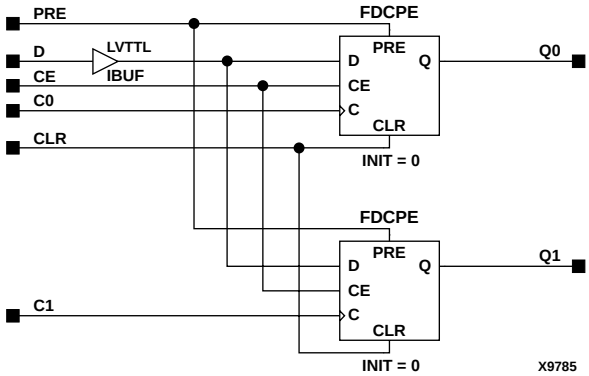
IFDDRCPE is a dual data rate (DDR) input D flip-flop with clock enable (CE) and asynchronous preset (PRE) and clear (CLR). It consists of one input buffer and two identical flip-flops (FDCPE).

When the asynchronous PRE is High and CLR is Low, both the Q0 and Q1 outputs are set High. When CLR is High, both outputs are reset Low. When PRE and CLR are Low and CE is High, data on the D input is loaded into the Q0 output on the Low-to High C0 clock transition, and into the Q1 output on the Low-to-High C1 clock transition.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

The INIT attribute does not apply to IFDDRCPE components.

Inputs						Outputs	
C0	C1	CE	D	CLR	PRE	Q0	Q1
X	X	X	X	1	0	0	0
X	X	X	X	0	1	1	1
X	X	X	X	1	1	0	0
X	X	0	X	0	0	No Chg	No Chg
↑	X	1	D	0	0	d	No Chg
X	↑	1	D	0	0	No Chg	d



IFDDRCPE Implementation Virtex-II, Virtex-II Pro

Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDDRCPE, you would infer an FDDRCPE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- Component Declaration for IFDDRCPE should be placed
-- after architecture statement but before begin keyword

component IFDDRCPE

   -- synthesis translate_on
   port (Q0   : out STD_ULOGIC;
         Q1   : out STD_ULOGIC;
         C0   : in STD_ULOGIC;
         C1   : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for IFDDRCPE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IFDDRCPE should be placed
-- in architecture after the begin keyword

IFDDRCPE_INSTANCE_NAME : IFDDRCPE
      port map  (Q0 => user_Q0,
                 Q1 => user_Q1,
                 C0 => user_C0,
                 C1 => user_C1,
                 CE => user_CE,
                 CLR=> user_CLR,
                 D  => user_D0,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
IFDDRCPE IFDDRCPE_instance_name (.Q0 (user_Q0),
                                 .Q1 (user_Q1),
                                 .C0 (user_C0),
                                 .C1 (user_C1),
                                 .CE (user_CE),
                                 .CLR (user_CLR),
                                 .D (user_D),
                                 .PRE (user_PRE));
```

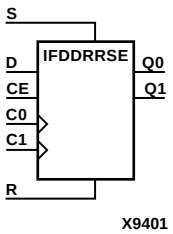
Commonly Used Constraints

LOC, RLOC, INIT

IFDDRRSE

Dual Data Rate Input D Flip-Flop with Synchronous Reset and Set and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



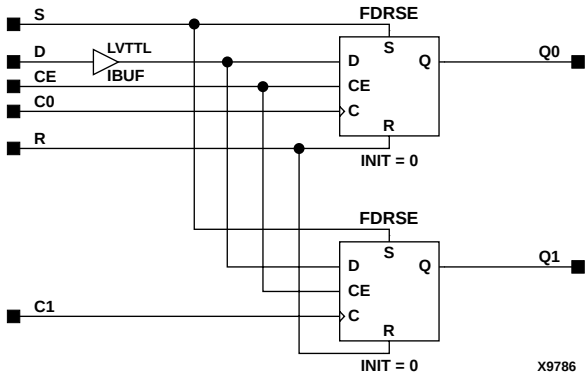
IFDDRRSE is a dual data rate (DDR) input D flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE). It consists of one input buffer and two identical flip-flops (FDRSE).

For the C0 input and Q0 output, reset (R) has precedence. The R input, when High, resets the Q0 output Low during the Low-to-High C0 clock transition. When S is High and R is Low, the Q0 output is set High during the Low-to-High C0 clock transition. For the C1 input and Q1 output, set (S) has precedence. The R input, when High, resets the Q1 output Low during the Low-to-High C1 clock transition. When S is High and R is Low, the Q0 output is set to High during the Low-to-High C1 clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

The INIT attribute does not apply to IFDDRRSE components.

Inputs						Outputs	
C0	C1	CE	D	R	S	Q0	Q1
↑	X	X	X	1	0	0	No Chg
↑	X	X	X	0	1	1	No Chg
↑	X	X	X	1	1	0	No Chg
X	↑	X	X	1	0	No Chg	0
X	↑	X	X	0	1	No Chg	1
X	↑	X	X	1	1	No Chg	0
X	X	0	X	0	0	No Chg	No Chg
↑	X	1	D	0	0	D	No Chg
X	↑	1	D	0	0	No Chg	D



IFDDRRSE Implementation Virtex-II, Virtex-II Pro

Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDDRRSE, you would infer an FDDRRSE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- Component Declaration for IFDDRRSE should be placed
-- after architecture statement but before begin keyword

component IFDDRRSE
   port (Q0   : out STD_ULOGIC;
         Q1   : out STD_ULOGIC;
         C0   : in STD_ULOGIC;
         C1   : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         R    : in STD_ULOGIC;
         S    : in STD_ULOGIC);
end component;

-- Component Attribute specification for IFDDRRSE
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes here


-- Component Instantiation for IFDDRRSE should be placed
-- in architecture after the begin keyword

IFDDRRSE_INSTANCE_NAME : IFDDRRSE
      port map  (Q0 => user_Q0,
                 Q1 => user_Q1,
                 C0 => user_C0,
```

```
C1 => user_C1,
CE => user_CE,
D  => user_D,
R  => user_R,
S  => user_S);
```

Verilog Instantiation Template

```
IFDDRRSE IFDDRRSE_instance_name (.Q0 (user_Q0),
                                 .Q1 (user_Q1),
                                 .C0 (user_C0),
                                 .C1 (user_C1),
                                 .CE (user_CE),
                                 .D (user_D),
                                 .R (user_R),
                                 .S (user_S));
```

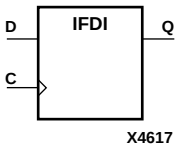
Commonly Used Constraints

LOC, RLOC, INIT

IFDI

Input D Flip-Flop (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



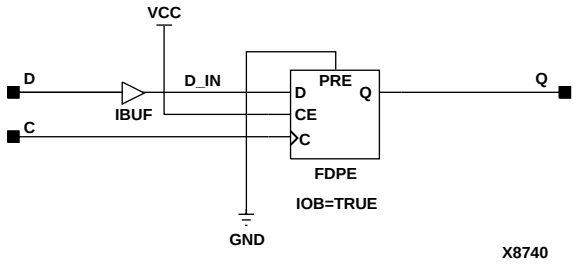
The IFDI D-type flip-flop is contained in an input / output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously preset, output High, when power is applied.

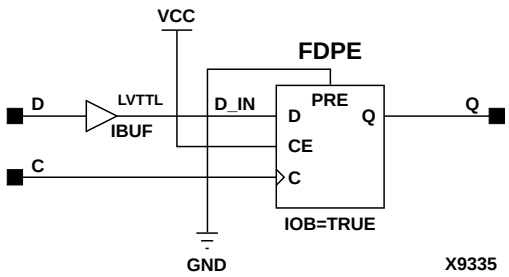
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1



IFDI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDI Implementation Virtex-II, Virtex-II Pro

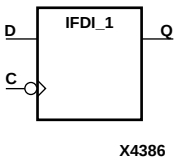
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDI, you would infer an FDP and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

IFDI_1

Input D Flip-Flop with Inverted Clock (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



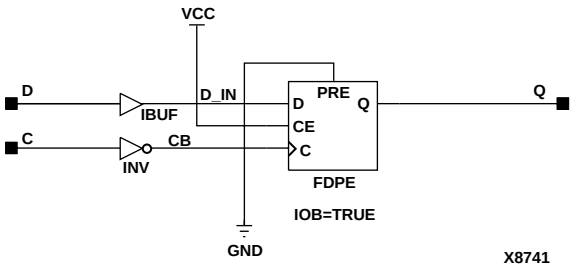
The IFDI_1 D-type flip-flop is contained in an input / output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously preset, output High, when power is applied.

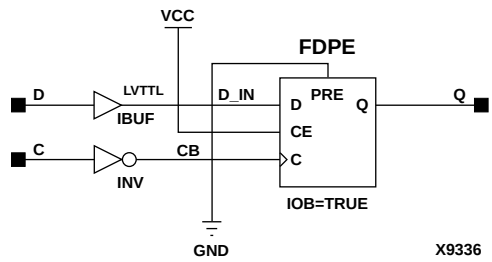
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1



IFDI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDI_1 Implementation Virtex-II, Virtex-II Pro

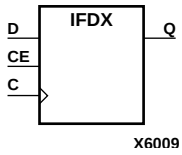
Usage

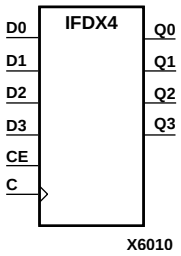
This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDI_1, you would infer an FDP_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

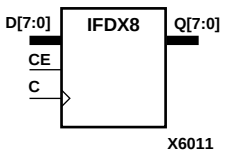
IFDX, 4, 8, 16

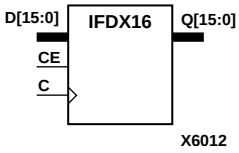
Single- and Multiple-Input D Flip-Flops with Clock Enable

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
IFDX	Macro	Macro	Macro	N/A	N/A	N/A
IFDX4, IFDX8, IFDX16	Macro	Macro	Macro	N/A	N/A	N/A









The IFDX D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD (without using an IBUF). The D input provides data input for the flip-flop, which synchronizes data entering the chip. When CE is High, the data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When CE is Low, flip-flop outputs do not change.

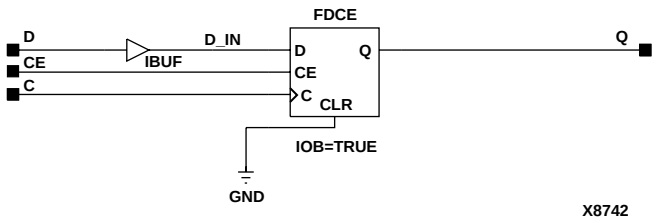
The flip-flops are asynchronously cleared with Low outputs when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

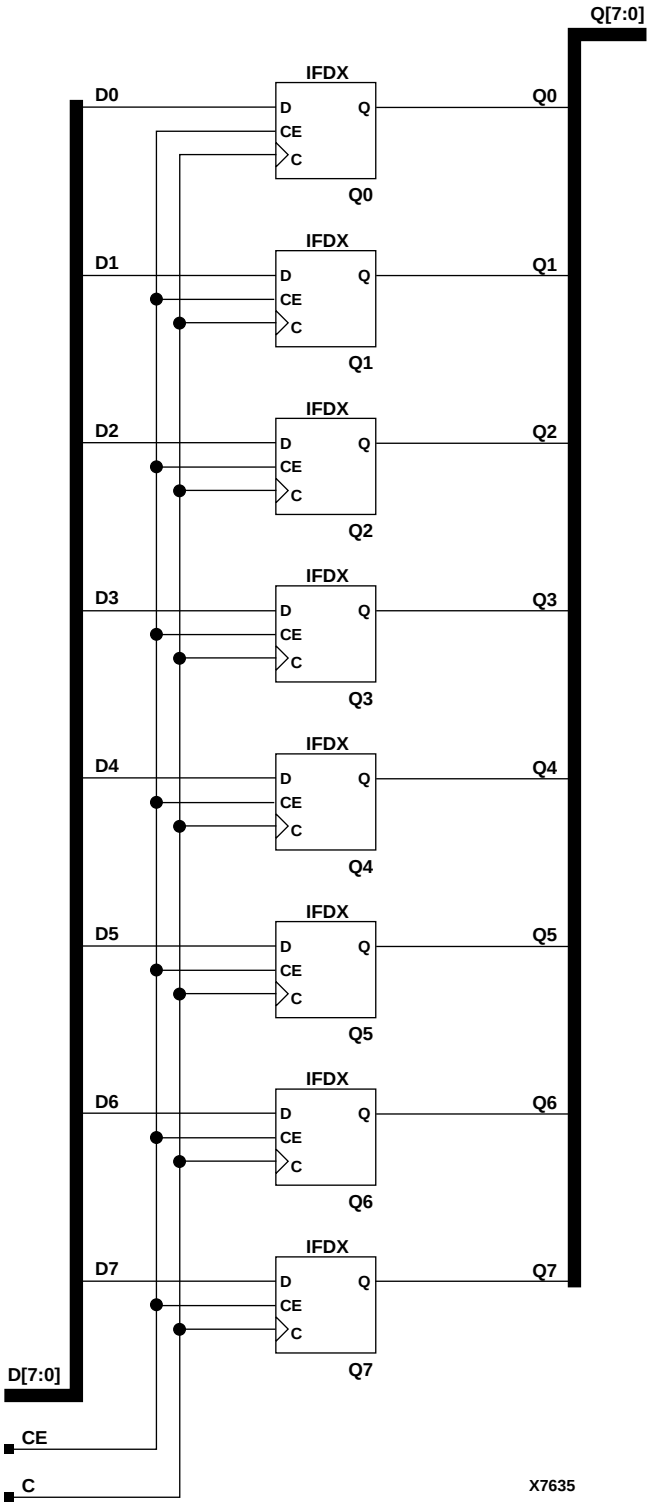
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	Dn	C	Qn
1	Dn	↑	dn
0	X	X	No Chg

dn = state of referenced input (Dn) one setup time prior to active clock transition



IFDX Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



IFDX8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDX, you would infer an FDCE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- Component Declaration for IFDX should be placed
-- after architecture statement but before begin keyword

component IFDX
   port (Q    : out STD_ULOGIC;
         C    : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for IFDX
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes here


-- Component Instantiation for IFDX should be placed
-- in architecture after the begin keyword

IFDX_INSTANCE_NAME : IFDX
      port map  (Q => user_Q,
                 C => user_C,
                 CE => user_CE,
                 D => user_D);
```

Verilog Instantiation Template

```
IFDX IFDX_instance_name (.Q (user_Q),
                         .C (user_C),
                         .CE (user_CE),
                         .D (user_D));
```

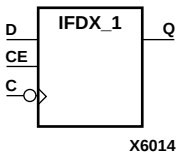
Commonly Used Constraints

IOB

IFDX_1

Input D Flip-Flop with Inverted Clock and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



The IFDX_1 D-type flip-flop is contained in an input / output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input also provides data input for the flip-flop, which synchronizes data entering the chip. When CE is High, the data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

The flip-flop is asynchronously cleared with Low output, when power is applied.

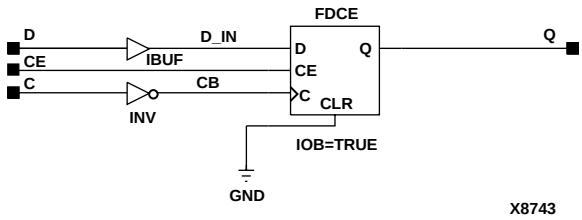
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

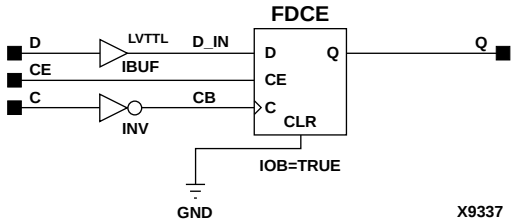
For more information on IFDX_1, see the "ILDX, 4, 8, 16" section.

Inputs			Outputs
CE	D	C	Q
1	D	↓	d
0	X	X	No Chg

d = state of D input one setup time prior to active clock transition



IFDX_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDX_1 Implementation Virtex-II, Virtex-II Pro

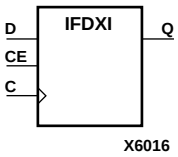
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDX_1, you would infer an FDCE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

IFDXI

Input D Flip-Flop with Clock Enable (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



The IFDXI D-type flip-flop is contained in an input / output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. When CE is High, the data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

The flip-flop is asynchronously preset with High output, when power is applied.

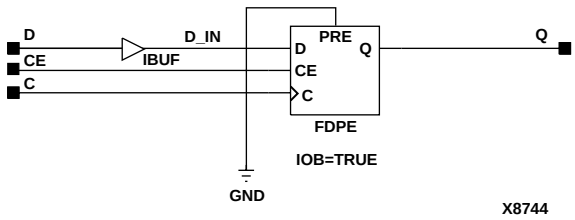
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

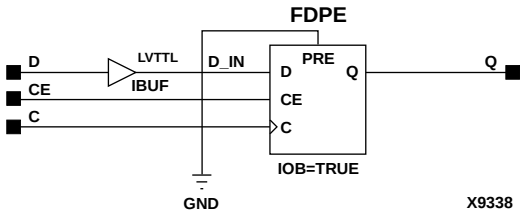
For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, see the "ILDXI" section.

Inputs			Outputs
CE	D	C	Q
1	D	↑	d
0	X	X	No Chg

d = state of D input one setup time prior to active clock transition



IFDXI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDXI Implementation Virtex-II, Virtex-II Pro

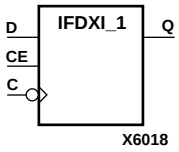
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDXI, you would infer an FDPE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

IFDXI_1

Input D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



The IFDXI_1 D-type flip-flop is contained in an input / output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. When CE is High, the data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

The flip-flop is asynchronously preset with High output when power is applied.

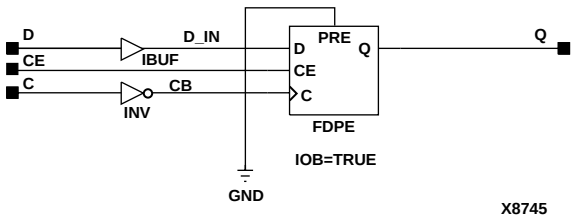
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

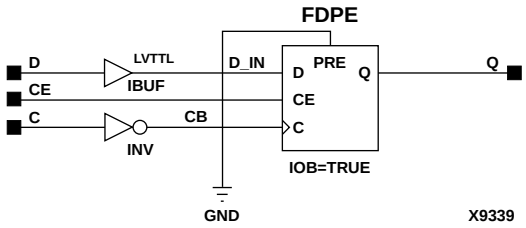
For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, see the "ILDXI" section.

Inputs			Outputs
CE	D	C	Q
1	D	↓	d
0	X	X	No Chg

d = state of D input one setup time prior to active clock transition



IFDXI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDXI_1 Implementation Virtex-II, Virtex-II Pro

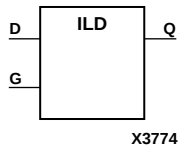
Usage

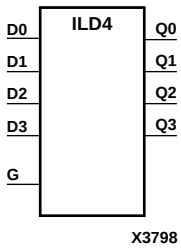
This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDXI_1, you would infer an FDPE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

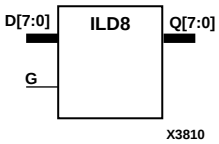
ILD, 4, 8, 16

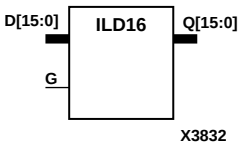
Transparent Input Data Latches

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
ILD	Macro	Macro	Macro	Macro	Macro	Macro
ILD4, ILD8, ILD16	Macro	Macro	Macro	Macro	Macro	Macro









ILD, ILD4, ILD8, and ILD16 are single or multiple transparent data latches, which can be used to hold transient data entering a chip. The ILD latch is contained in an input / output block (IOB), except for XC9500 / XV / XL. The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (G) is High, data on the inputs (D) appears on the outputs (Q). Data on the D inputs during the High-to-Low G transition is stored in the latch.

The latch is asynchronously cleared with Low output when power is applied.

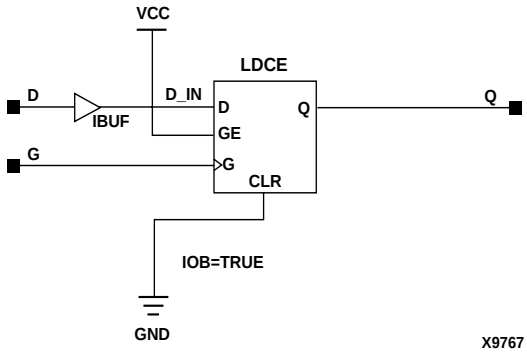
For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

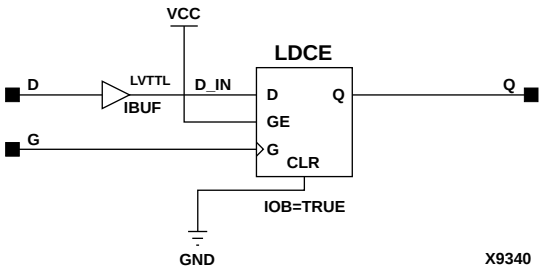
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
G	D	Q
1	1	1
1	0	0
0	X	No Chg
↓	D	d

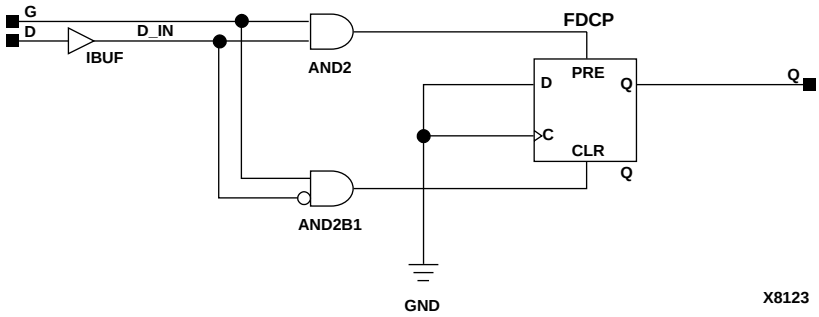
d = state of referenced input one setup time prior to active G transition



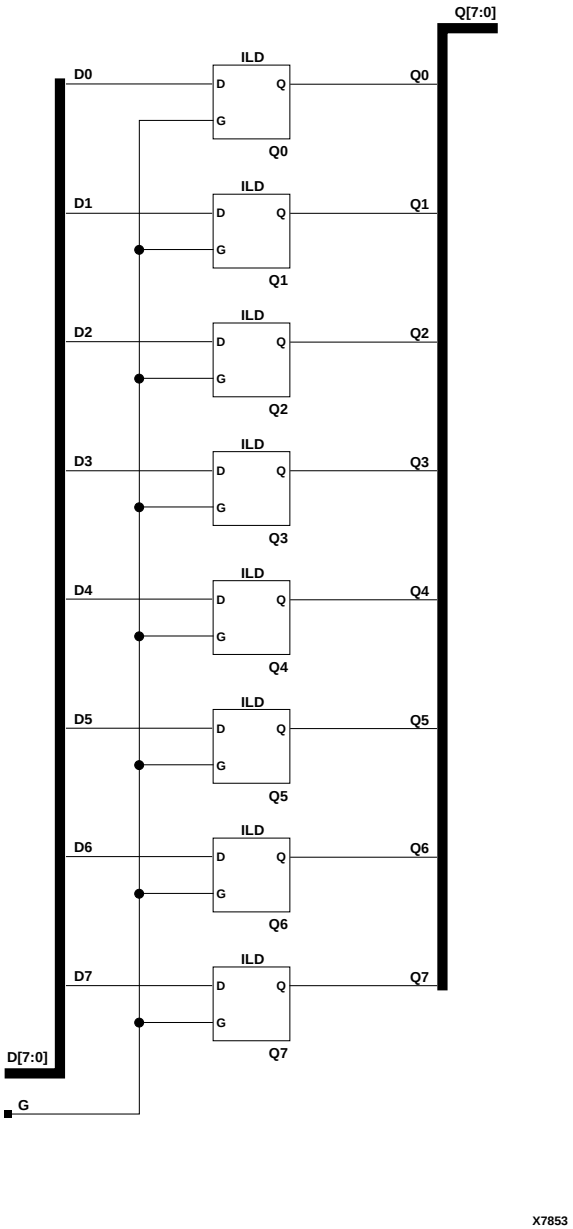
ILD Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILD Implementation Virtex-II, Virtex-II Pro



ILD Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



ILD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILD, you would infer an LD and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- Component Declaration for ILD should be placed
-- after architecture statement but before begin keyword

component ILD
      port(Q: out STD_ULOGIC;
      D     : in STD_ULOGIC;
      DG    : in STD_ULOGIC);
end component;

-- Component Attribute specification for ILD
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ILD should be placed
-- in architecture after the begin keyword

ILD_INSTANCE_NAME : ILD
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G);
```

Verilog Instantiation Template

```
ILD ILD_instance_name(.Q (user_Q),
                      .D (user_D),
                      .G (user_G));
```

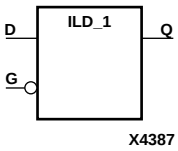
Commonly Used Constraints

INIT

ILD_1

Transparent Input Data Latch with Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



ILD_1 is a transparent data latch, which can be used to hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

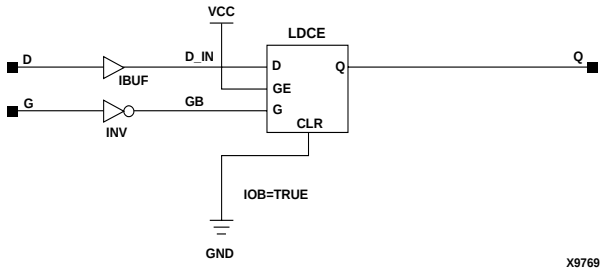
The latch is asynchronously cleared with Low output when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

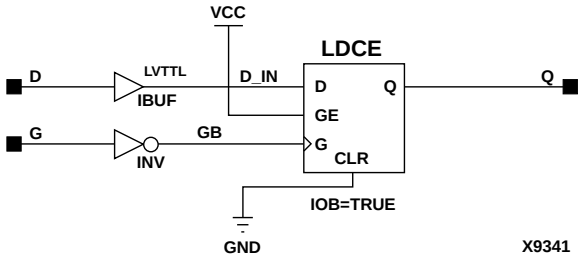
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
G	D	Q
0	1	1
0	0	0
1	X	d
↑	D	d

d = state of referenced input one setup time prior to Low-to-High gate transition



ILD_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILD_1 Implementation Virtex-II, Virtex-II Pro

Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILD_1, you would infer an LD_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- Component Declaration for ILD_1 should be placed
-- after architecture statement but before begin keyword

component ILD_1
      port(Q: out STD_ULOGIC;
      D     : in STD_ULOGIC;
      DG    : in STD_ULOGIC);
end component;

-- Component Attribute specification for ILD_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ILD_1 should be placed
-- in architecture after the begin keyword

ILD_1_INSTANCE_NAME : ILD_1
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G);
```

Verilog Instantiation Template

```
ILD_1 ILD_1_instance_name(.Q (user_Q),
                          .D (user_D),
                          .G (user_G));
```

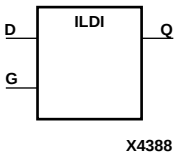
Commonly Used Constraints

INIT

ILDI

Transparent Input Data Latch (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



ILDI is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is High, data on the input (D) appears on the output (Q). Data on the D input during the High-to-Low G transition is stored in the latch.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

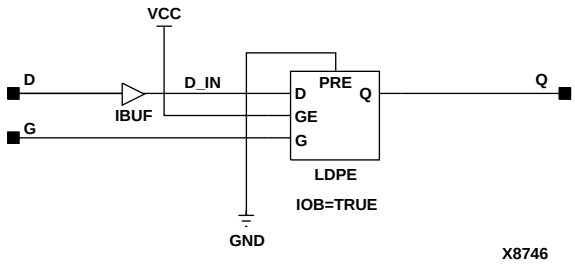
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

ILDIs and IFDIs

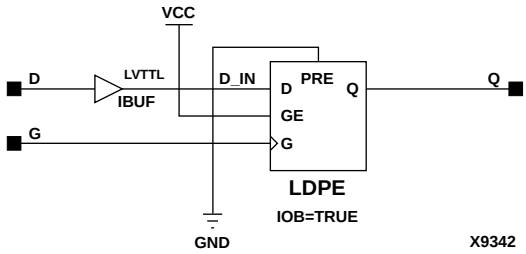
The ILDI is actually the input flip-flop master latch. It is possible to access two different outputs from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILDI) corresponds to a falling edge-triggered flip-flop (IFDI_1). Similarly, a transparent Low latch (ILDI_1) corresponds to a rising edge-triggered flip-flop (IFDI).

Inputs		Outputs
G	D	Q
1	1	1
1	0	0
0	X	d
↓	D	d

d = state of referenced input one setup time prior to High-to-Low gate transition



ILDI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDI Implementation Virtex-II, Virtex-II Pro

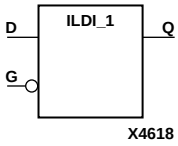
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDI, you would infer an LDP and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

ILDI_1

Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



ILDI_1 is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

The latch is asynchronously preset, output High, when power is applied.

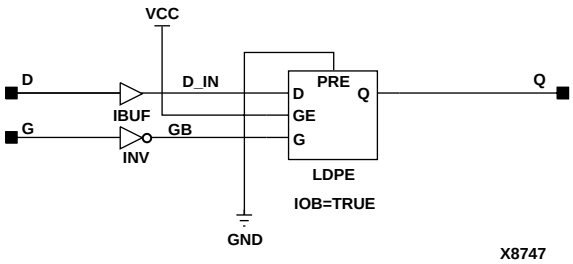
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

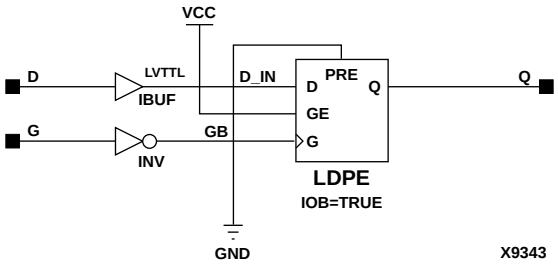
For information on ILDI_1, see the "ILDI" section.

Inputs		Outputs
G	D	Q
0	1	1
0	0	0
1	X	d
↑	D	d

d = state of input one setup time prior to High-to-Low gate transition



ILDI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDI_1 Implementation Virtex-II, Virtex-II Pro

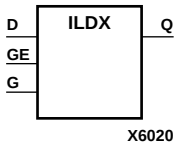
Usage

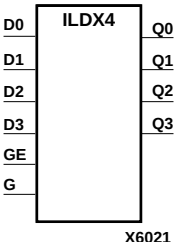
This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDI_1, you would infer an LDP_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

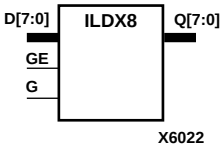
ILDX, 4, 8, 16

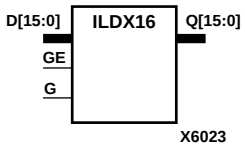
Transparent Input Data Latches

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
ILDX	Macro	Macro	Macro	N/A	N/A	N/A
ILDX4, ILDX8, ILDX16	Macro	Macro	Macro	N/A	N/A	N/A









ILDX, ILDX4, ILDX8, and ILDX16 are single or multiple transparent data latches, which can be used to hold transient data entering a chip. The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF).

The latch is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

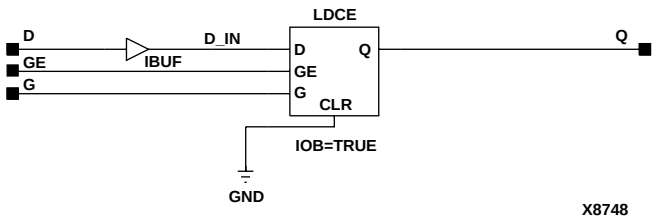
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

ILDXs and IFDXs

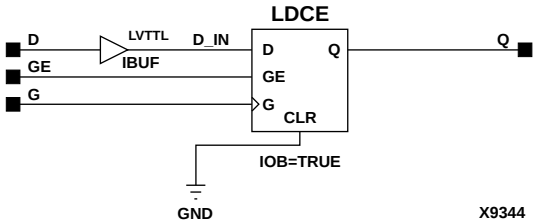
The ILDX is actually the input flip-flop master latch. Two different outputs can be accessed from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILDX) corresponds to a falling edge-triggered flip-flop (IFDX_1). Similarly, a transparent Low latch (ILDX_1) corresponds to a rising edge-triggered flip-flop (IFDX).

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	0	X	No Chg
1	1	1	1
1	1	0	0
1	↓	D	d

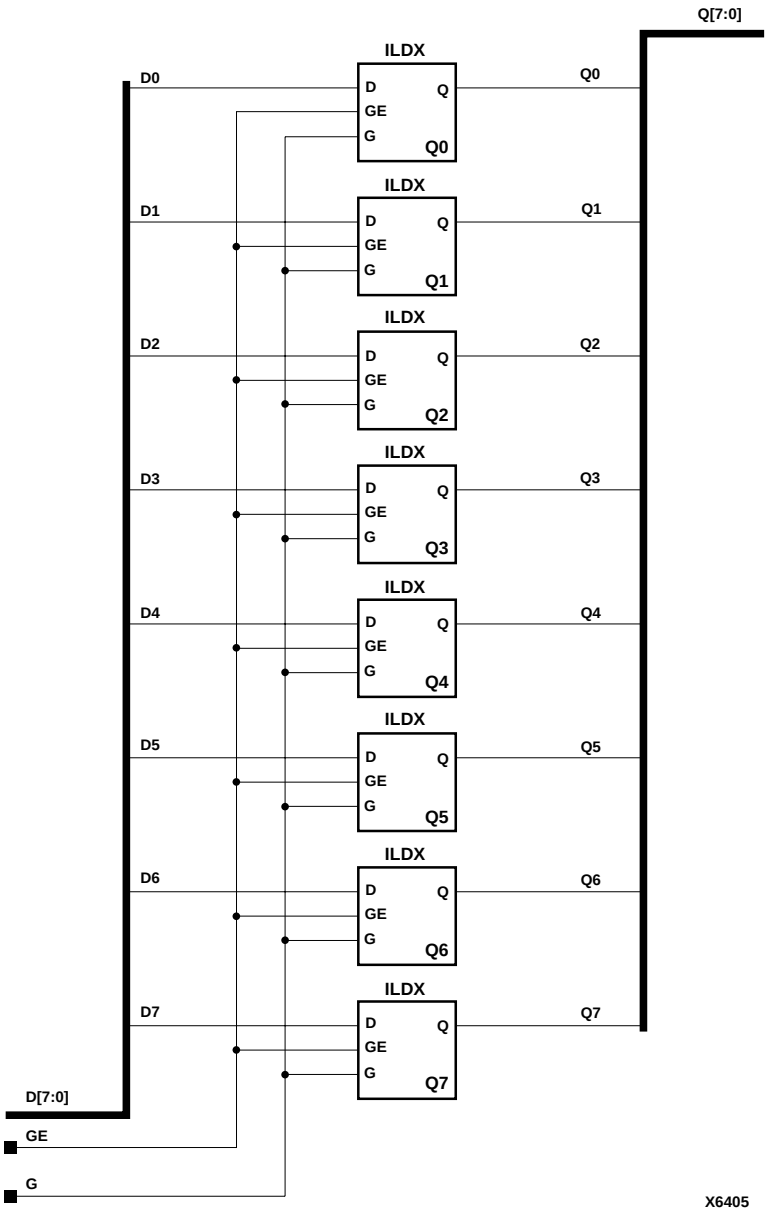
d = state of input one setup time prior to High-to-Low gate transition



ILDX Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDX Implementation Virtex-II, Virtex-II Pro



ILDX8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

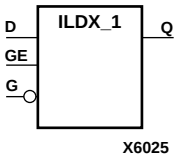
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDX, you would infer an LDCE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

ILDX_1

Transparent Input Data Latch with Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



ILDX_1 is a transparent data latch, which can be used to hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

The latch is asynchronously cleared with Low output, when power is applied.

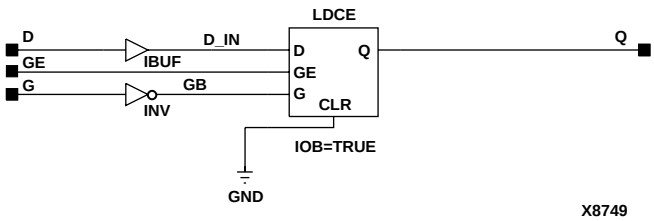
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

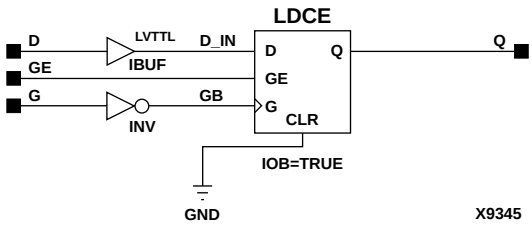
For more information on ILDX_1, see the "ILDX, 4, 8, 16" section.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	X	No Chg
1	0	1	1
1	0	0	0
1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition



ILDX_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDX_1 Implementation Virtex-II, Virtex-II Pro

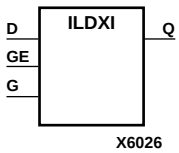
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDX_1, you would infer an LDCE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

ILDXI

Transparent Input Data Latch (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



ILDXI is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is High, data on the input (D) appears on the output (Q). Data on the D input during the High-to-Low G transition is stored in the latch.

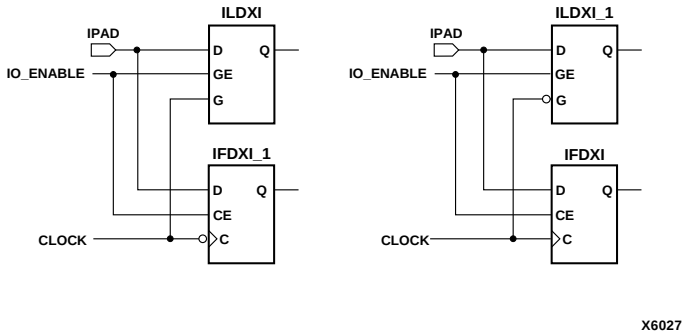
The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

ILDXIs and IFDXIs

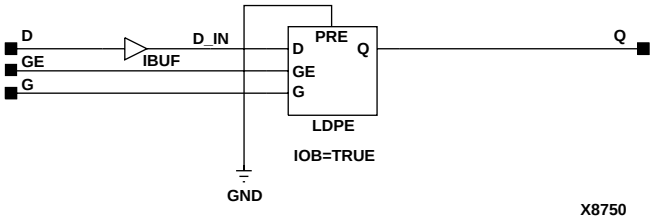
The ILDXI is actually the input flip-flop master latch. Two different outputs can be accessed from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILDXI) corresponds to a falling edge-triggered flip-flop (IFDXI_1). Similarly, a transparent Low latch (ILDXI_1) corresponds to a rising edge-triggered flip-flop (IFDXI). See the following figure for legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations.



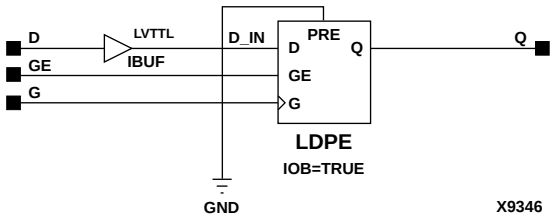
Legal Combinations of IFDXI and ILDXI for a Single IOB

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	0	X	No Chg
1	1	1	1
1	1	0	0
1	↓	D	d

d = state of referenced input one setup time prior to High-to-Low gate transition



ILDXI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDXI Implementation Virtex-II, Virtex-II Pro

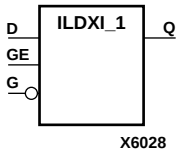
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDXI, you would infer an LDPE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

ILDXI_1

Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



ILDXI_1 is a transparent data latch, which can hold transient data entering a chip. The latch is asynchronously preset, output High, when power is applied.

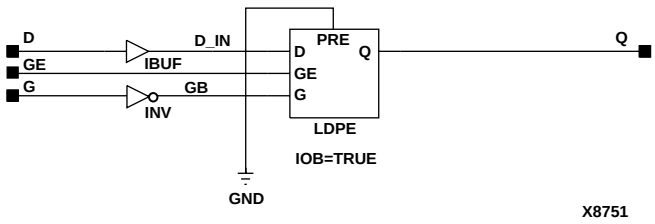
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

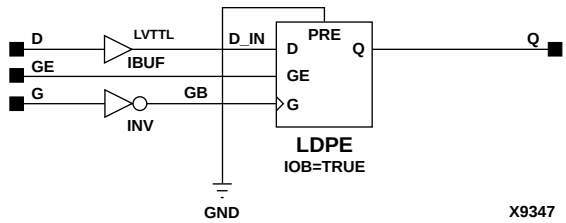
For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, see the "ILDXI" section.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	X	No Chg
1	0	1	1
1	0	0	0
1	↑	D	d

d = state of referenced input one setup time prior to Low-to-High gate transition



ILDXI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDXI_1 Implementation Virtex-II, Virtex-II Pro

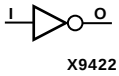
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDXI_1, you would infer an LDPE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr i to pack all input registers into the IOBs.

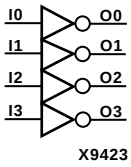
INV, 4, 8, 16

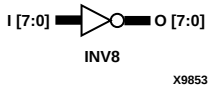
Single and Multiple Inverters

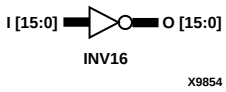
Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
INV	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
INV4, INV8, INV16	Macro	Macro	Macro	Macro	Macro	Macro

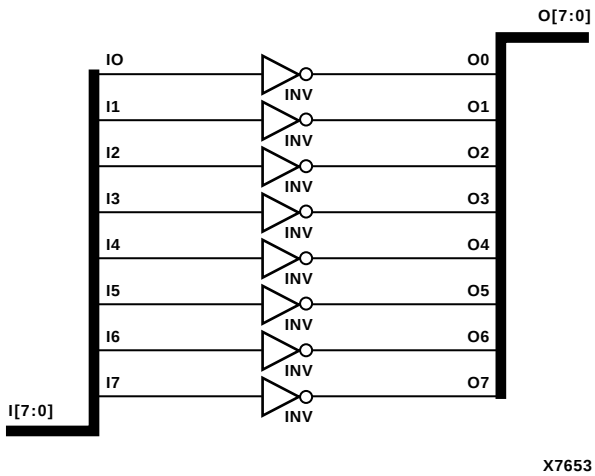


INV, INV4, INV8, and INV16 are single and multiple inverters that identify signal inversions in a schematic.









INV8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Inference Code

```
architecture Behavioral of inv is
begin
process (i)
begin
 o <= not i;
end process;

end Behavioral
```

Verilog Inference Code

```
always @(i)
begin
 o = !i;
end
```

VHDL Instantiation Template

```
-- Component Declaration for INV should be placed
-- after architecture statement but before begin keyword

component INV
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for INV
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for INV should be placed
-- in architecture after the begin keyword

INV_INSTANCE_NAME : INV
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
INV instance_name (.O (user_O),
                   .I (user_I));
```

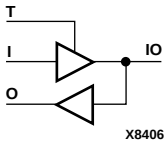
Commonly Used Constraints

None

IOBUF, IOBUF_*selectIO*

Bi-Directional Buffer with Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, IOBUF and its *selectIO* variants (listed in the "Component" column in the table below) are bi-directional buffers whose I/O interface corresponds to a specific I/O standard.The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. The S, F, and 2, 4, 6, 8, 12, 16, 24 extensions specify the slew rate (SLOW or FAST) and the drive power (2, 4, 6, 8, 12, 16, 24 mA) for the LVTTL standard variants. For example, IOBUF_F_2 is a bi-directional buffer that uses the LVTTL I/O-signaling standard with a FAST slew and 2mA of drive power. You can attach an IOSTANDARD attribute to an IOBUF instance instead of using an IOBUF_*selectIO* component. Check marks (√) in the "Spartan-II, Virtex" and "Spartan-IIE, Virtex-E" columns indicate the components and IOSTANDARD attribute values available for each architecture.

IOBUF components that use the LVTTL, LVCMOS15, LVCMOS18, LVCMOS25, LVCMOS33 signaling standards have selectable drive and slew rates using the DRIVE and FAST or SLOW constraints. The defaults are DRIVE=12 mA and SLOW slew.

IOBUFs are composites of IBUF and OBUFT elements. The O output is X (unknown) when IO (input/output) is Z. IOBUFs can be implemented as interconnections of their component elements.

The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffers. See the "SelectI/O Usage Rules" under the IBUF_*selectIO* section for information on using these components and IOSTANDARD attributes.

Inputs		Bidirectional	Outputs
T	I	IO	O
1	X	Z	X
0	1	1	1
0	0	0	0

Spartan-II, Spartan-IIE, Virtex, and Virtex-E IOBUF_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Output VCCO	Input VCCO	VREF
IOBUF	√	√	defaults to LVTTL [b]	3.3	3.3	N / A
IOBUF_S_2	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_S_4	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_S_6	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_S_8	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_S_12	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_S_16	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_S_24	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_F_2	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_F_4	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_F_6	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_F_8	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_F_12	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_F_16	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_F_24	√	√	LVTTL [b]	3.3	3.3	N / A
IOBUF_AGP	√	√	AGP	3.3	N / A	1.32
IOBUF_CTT	√	√	CTT	3.3	N / A	1.50
IOBUF_GTL	√	√	GTL	N / A	N / A	0.80
IOBUF_GTLP	√	√	GTLP	N / A	N / A	0.80
IOBUF_HSTL_I	√	√	HSTL_I	1.5	N / A	0.75
IOBUF_HSTL_III	√	√	HSTL_III	1.5	1.5	0.90
IOBUF_HSTL_IV	√	√	HSTL_IV	1.5	N / A	0.90
IOBUF_LVCMOS2	√	√	LVCMOS2	2.5	2.5	N / A
IOBUF_LVCMOS18		√	LVCMOS18 [b]	1.8	1.8	N / A
IOBUF_LVDS		√	LVDS	2.5	N / A	N / A
IOBUF_LVPECL		√	LVPECL	3.3	N / A	N / A
IOBUF_PCI33_3	√	√	PCI33_3	3.3	3.3	N / A
IOBUF_PCI33_5	√	√	PCI33_5	3.3	N / A	N / A
IOBUF_PCI66_3	√	√	PCI66_3	3.3	3.3	N / A
IOBUF_ PCIX66_3		√	PCIX66_3	3.3	3.3	N / A
IOBUF_SSTL2_I	√	√	SSTL2_I	2.5	N / A	1.25
IOBUF_SSTL2_II	√	√	SSTL2_II	2.5	N / A	1.25
IOBUF_SSTL3_I	√	√	SSTL3_I	3.3	N / A	1.50
IOBUF_SSTL3_II	√	√	SSTL3_II	3.3	N / A	1.50

[b] The LVCMOS18 attribute also requires a slew value (FAST or SLOW) and DRIVE value. See the FAST, SLOW, and DRIVE attribute descriptions in the *Xilinx Constraints Guide* for valid values for each architecture.

The Virtex-II and Virtex-II Pro library includes some IOBUF_*selectIO* components for compatibility with older, existing designs and other architectures. For new Virtex-II and Virtex-II Pro designs, however, the recommended method for using IOBUF selectI/O buffers is to attach an IOSTANDARD attribute to an IOBUF component. For example, attach IOSTANDARD=GTLP to an IOBUF instead of using the IOBUF_GTLP component for new Virtex-II and Virtex-II Pro designs. The IOSTANDARD attributes that can be attached to an IOBUF component are listed in the "IOSTANDARD (Attribute Value)" column in the following table. See the "SelectI/O Usage Rules" section for information on using these IOSTANDARD attributes.

For Virtex-II and Virtex-II Pro, the IOSTANDARD attribute values listed in the following table can be applied to an IOBUF component to provide SelectI/O interface capability for the inputs. The O output uses the LVTTL standard.

Virtex-II, Virtex-II Pro IOBUF_*selectIO* Components and IOSTANDARD Attributes

			Attribute Values						
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Input VCCO**	**VREF**	**Terminate Type**
IOBUF[a]	√		AGP	N / A	N / A	3.3	N / A	1.32	None
IOBUF[a]	√	√	GTL	N / A	N / A	N / A	N / A	0.80	None
IOBUF[a]	√	√	GTL_DCI	N / A	N / A	1.2	1.2	0.80	Single
IOBUF[a]	√	√	GTLP	N / A	N / A	N / A	N / A	1.00	None
IOBUF[a]	√	√	GTLP_DCI	N / A	N / A	1.5	1.5	1.00	Single
IOBUF[a]	√	√	HSTL_I	N / A	N / A	1.5	1.5	0.75	Split
IOBUF[a]	√	√	HSTL_I_18	N / A	N / A	1.8	1.5	0.75	Split
IOBUF[a]	√	√	HSTL_II	N / A	N / A	1.5	N / A	0.75	None
IOBUF[a]	√	√	HSTL_II_18	N / A	N / A	1.8	1.5	0.75	Split
IOBUF[a]	√	√	HSTL_II_DCI	N / A	N / A	1.5	1.5	0.75	Split
IOBUF[a]	√	√	HSTL_II_DCI_18	N / A	N / A	1.8	1.5	0.75	Split
IOBUF[a]	√	√	HSTL_III	N / A	N / A	1.5	N / A	0.75	None
IOBUF[a]	√	√	HSTL_III_18	N / A	N / A	1.8	N / A	0.75	None
IOBUF[a]	√	√	HSTL_IV	N / A	N / A	1.5	N / A	0.90	None
IOBUF[a]	√	√	HSTL_IV_18	N / A	N / A	1.8	N / A	0.90	None
IOBUF[a]	√	√	HSTL_IV_DCI	N / A	N / A	1.5	1.5	0.90	Split
IOBUF[a]	√	√	HSTL_IV_DCI_18	N / A	N / A	1.8	1.5	0.90	Split
IOBUF [a]	√	√	LVCMOS15[b]	N / A	N / A	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS2[b]	N / A	N / A	2.5	2.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	12	F	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	16	F	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	2	F	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	4	F	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	6	F	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	8	F	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	12	S	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	16	S	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	2	S	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	4	S	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	6	S	1.5	1.5	N / A	None
IOBUF [a]	√	√	LVCMOS15[b]	8	S	1.5	1.5	N / A	None
IOBUF[a]	√	√	LVCMOS18[b]	N / A	N / A	1.8	1.8	N / A	None
IOBUF[a]	√	√	LVCMOS18[b]	12	F	1.8	1.8	N / A	None
IOBUF[a]	√	√	LVCMOS18[b]	16	F	1.8	1.8	N / A	None
IOBUF[a]	√	√	LVCMOS18[b]	2	F	1.8	1.8	N / A	None
IOBUF[a]	√	√	LVCMOS18[b]	4	F	1.8	1.8	N / A	None
IOBUF[a]	√	√	LVCMOS18[b]	6	F	1.8	1.8	N / A	None
IOBUF[a]	√	√	LVCMOS18[b]	8	F	1.8	1.8	N / A	None

			Attribute Values						
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Input VCCO**	**VREF**	**Terminate Type**
IOBUF[a]	√	√	LVCMOS18[b]	12	S	1.8	1.8	N/A	None
IOBUF[a]	√	√	LVCMOS18[b]	16	S	1.8	1.8	N/A	None
IOBUF[a]	√	√	LVCMOS18[b]	2	S	1.8	1.8	N/A	None
IOBUF[a]	√	√	LVCMOS18[b]	4	S	1.8	1.8	N/A	None
IOBUF[a]	√	√	LVCMOS18[b]	6	S	1.8	1.8	N/A	None
IOBUF[a]	√	√	LVCMOS18[b]	8	S	1.8	1.8	N/A	None
IOBUF[a]	√	√	LVCMOS2[b]	N/A	N/A	2.0	2.0	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	N/A	N/A	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	12	F	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	16	F	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	2	F	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	24	F	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	4	F	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	6	F	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	8	F	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	12	S	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	16	S	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	2	S	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	24	S	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	4	S	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	6	S	2.5	2.5	N/A	None
IOBUF[a]	√	√	LVCMOS25[b]	8	S	2.5	2.5	N/A	None
IOBUF[a]	√		LVCMOS33[b]	N/A	N/A	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	12	F	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	16	F	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	2	F	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	24	F	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	4	F	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	6	F	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	8	F	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	12	S	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	16	S	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	2	S	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	24	S	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	4	S	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	6	S	3.3	3.3	N/A	None
IOBUF[a]	√		LVCMOS33[b]	8	S	3.3	3.3	N/A	None
IOBUF[a]	√	√	LVDCI_15	N/A	N/A	1.5	N/A	N/A	Driver
IOBUF[a]	√	√	LVDCI_18	N/A	N/A	1.8	N/A	N/A	Driver

			Attribute Values						
Component	Virtex-II	Virtex-II Pro	IOSTANDARD	Drive	Slew	Output VCCO	Input VCCO	VREF	Terminate Type
IOBUF[a]	√	√	LVDCI_25	N / A	N / A	2.5	N / A	N / A	Driver
IOBUF[a]	√	√	LVDCI_33	N / A	N / A	3.3	N / A	N / A	Driver
IOBUF[a]	√	√	LVDCI_DV2_15	N / A	N / A	1.5	N / A	N / A	Driver
IOBUF[a]	√	√	LVDCI_DV2_18	N / A	N / A	1.8	N / A	N / A	Driver
IOBUF[a]	√	√	LVDCI_DV2_25	N / A	N / A	2.5	N / A	N / A	Driver
IOBUF[a]	√		LVDCI_DV2_33	N / A	N / A	3.3	N / A	N / A	Driver
IOBUF[a]	√		LVTTL (default)[b]	N / A	N / A	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	12	F	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	16	F	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	2	F	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	24	F	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	4	F	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	6	F	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	8	F	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	12	S	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	16	S	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	2	S	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	24	S	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	4	S	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	6	S	3.3	3.3	N / A	None
IOBUF[a]	√	√	LVTTL[b]	8	S	3.3	3.3	N / A	None
IOBUF[a]	√	√	PCI33_3	N / A	N / A	3.3	3.3	N / A	None
IOBUF[a]	√	√	PCI66_3	N / A	N / A	3.3	3.3	N / A	None
IOBUF[a]	√	√	PCIX	N / A	N / A	3.3	3.3	N / A	None
IOBUF[a]	√	√	SSTL18_I	N / A	N / A	1.8	1.8	0.9	None
IOBUF[a]	√	√	SSTL18_II	N / A	N / A	1.8	1.8	0.9	None
IOBUF[a]	√	√	SSTL18_II_DCI	N / A	N / A	1.8	1.8	0.9	Split
IOBUF[a]	√	√	SSTL2_II_DCI	N / A	N / A	2.5	2.5	1.25	Split
IOBUF[a]	√	√	SSTL3_II_DCI	N / A	N / A	2.5	3.3	1.25	Split
IOBUF[a]	√	√	SSTL2_I	N / A	N / A	2.5	N / A	1.25	None
IOBUF[a]	√	√	SSTL2_II	N / A	N / A	2.5	N / A	1.25	None
IOBUF[a]	√		SSTL3_I	N / A	N / A	3.3	N / A	1.50	None
IOBUF[a]	√		SSTL3_II	N / A	N / A	3.3	N / A	1.50	None

[a] Attach an IOSTANDARD attribute to an IOBUF and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the input for the I/O standard associated with that value.

[b] The LVTTL, LVCMOS15, LVCMOS18, LVCMOS25, LVCMOS33 attributes also require a slew value (FAST or SLOW) and DRIVE value. See the FAST, SLOW, and DRIVE attribute descriptions in the *Xilinx Constraints Guide* for valid values for Virtex-II.

Usage

For HDL, these design elements are instantiated rather than inferred.

VHDL Instantiation Template for IOBUF

```
-- Component Declaration for IOBUF should be placed
-- after architecture statement but before begin keyword

component IOBUF
   port (O  : out STD_ULOGIC;
         IO : inout STD_ULOGIC;
         I  : in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for IOBUF
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IOBUF should be placed
-- in architecture after the begin keyword

IOBUF_INSTANCE_NAME : IOBUF
      port map  (O => user_O,
                 IO => user_IO,
                 I => user_I,
                 T => user_T);
```

Verilog Instantiation Template for IOBUF

```
IOBUF instance_name (.O (user_O),
                     .IO (user_IO),
                     .I (user_I),
                     .T (user_T));
```

VHDL Instantiation Template for IOBUF_*selectIO*

```
-- Component Declaration for IOBUF_selectIO should be placed
-- after architecture statement but before begin keyword

component IOBUF_selectIO
   port (O  : out STD_ULOGIC;
         IO : inout STD_ULOGIC;
         I  : in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for IOBUF_selectIO
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IOBUF_selectIO should be
-- placed in architecture after the begin keyword

IOBUF_selectIO_INSTANCE_NAME : IOBUF_selectIO
      port map  (O => user_O,
                 IO => user_IO,
                 I => user_I,
                 T => user_T);
```

Verilog Instantiation Template for IOBUF_*selectIO*

```
IOBUF_selectIO instance_name (.O (user_O),
                              .IO (user_IO),
                              .I (user_I),
                              .T (user_T));
```

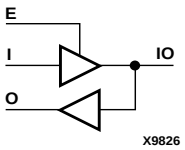
Commonly Used Constraints

IOSTANDARD, DRIVE, SLEW

IOBUFE

Bi-Directional Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	Primitive	Primitive	Primitive



For CPLDs, IOBUFE is a bi-directional buffer that is a composite of the IBUF and OBUFT elements. The O output is X (unknown) when IO (input/output) is Z. IOBUFEs can be implemented as interconnections of their component elements.

Inputs		Bidirectional	Outputs
E	I	IO	O
1	X	Z	X
0	1	1	1
0	0	0	0

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template for IOBUFE

```
-- Component Declaration for IOBUFE should be placed
-- after architecture statement but before begin keyword

component IOBUFE
   port (O  : out STD_ULOGIC;
         IO : inout STD_ULOGIC;
         I  : in STD_ULOGIC;
         E  : in STD_ULOGIC);
end component;

-- Component Attribute specification for IOBUFE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IOBUFE should be placed
-- in architecture after the begin keyword

IOBUFE_INSTANCE_NAME : IOBUFE
      port map  (O => user_O,
                 IO => user_IO,
                 I => user_I,
                 E => user_E);
```

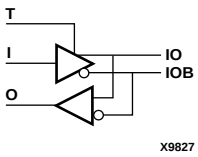
Verilog Instantiation Template for IOBUFE

```
IOBUFE instance_name (.O (user_O),
                      .IO (user_IO),
                      .I (user_I),
                      .E (user_E));
```

IOBUFDS

3-State Differential Signaling I/O Buffer with Active Low Output Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



IOBUFDS is a single 3-state, differential signaling input/output buffer with active Low output enable.

Inputs		Bidirectional		Outputs
I	T	IO	IOB	O
X	1	Z	Z	Z
0	0	0	1	0
1	0	1	0	1

BLVDS_25 is supported for IOSTANDARD.

Component	IOSTANDARD (Attribute Value)	Virtex-II	Virtex-II Pro	Output VCCO	VREF	Terminate Type
IOBUFDS[a]	BLVDS_25	√	√	2.5	N/A	None

[a]Attach an IOSTANDARD attribute to an IOBUFDS and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the outputs for the I/O standard associated with that value.

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for IOBUFDS should be placed
-- after architecture statement but before begin keyword

component IOBUFDS
   port (O  : out STD_ULOGIC;
         IO : inout STD_ULOGIC;
         IOB: inout STD_ULOGIC;
         I  : in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for IOBUFDS
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for IOBUFDS should be
-- placed in architecture after the begin keyword

IOBUFDS_INSTANCE_NAME : IOBUFDS
      port map  (O => user_O,
                 IO => user_IO,
                 IOB => user_IOB,
                 I => user_I,
                 T => user_T);
```

Verilog Instantiation Template

```
IOBUFDS instance_name (.O (user_O),
                       .IO (user_IO),
                       .IOB (user_IOB),
                       .I (user_I),
                       .T (user_T));
```

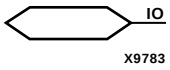
Commonly Used Constraints

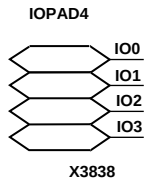
IOSTANDARD, IOBDELAY

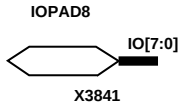
IOPAD, 4, 8, 16

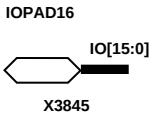
Single- and Multiple-Input/Output Pads

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
IOPAD	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IOPAD4, IOPAD8, IOPAD16	Macro	Macro	Macro	Macro	Macro	Macro



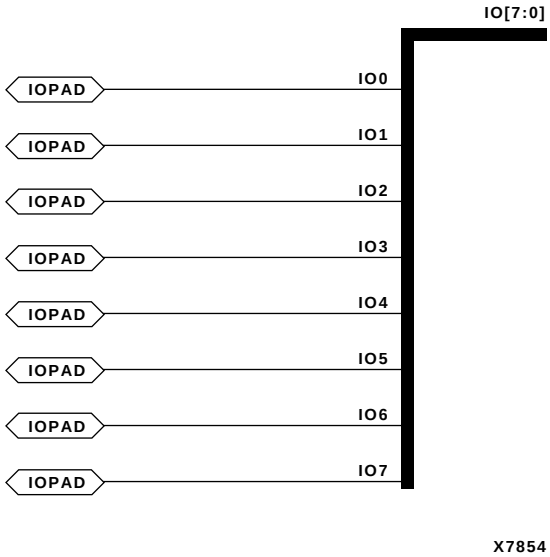






IOPAD, IOPAD4, IOPAD8, and IOPAD16 are single and multiple input/output pads. The IOPAD is a connection point from a device pin, used as a bidirectional signal, to a PLD device. The IOPAD is connected internally to an input/output block (IOB), which is configured by the software as a bidirectional block. Bidirectional blocks can consist of any combination of a 3-state output buffer (such as OBUFT or OFDE) and any available input buffer (such as IBUF or IFD). See the appropriate CAE tool interface user guide for details on assigning pin location and identification.

Note The LOC attribute cannot be used on IOPAD multiples.



IOPAD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, it is not necessary to use these elements in the design. They will be added automatically.

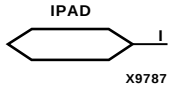
Commonly Used Constraints

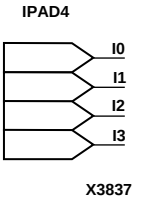
IOBDELAY, PULLDOWN

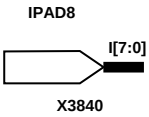
IPAD, 4, 8, 16

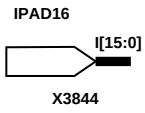
Single- and Multiple-Input Pads

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
IPAD	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IPAD4, IPAD8, IPAD16	Macro	Macro	Macro	Macro	Macro	Macro





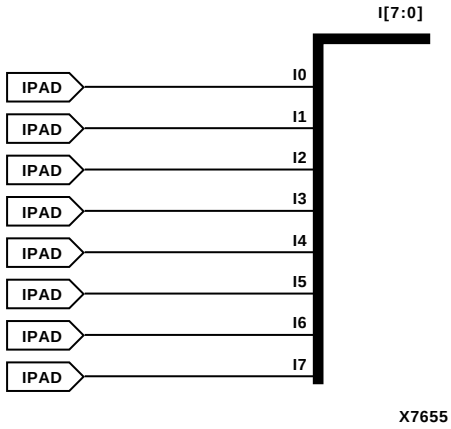




IPAD, IPAD4, IPAD8, and IPAD16 are single and multiple input pads. The IPAD is a connection point from a device pin used for an input signal to the PLD device. It is connected internally to an input/output block (IOB), which is configured by the software as an IBUF, IFD, or ILD. See the appropriate CAE tool interface user guide for details on assigning pin location and identification.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, pads must be used to drive IBUF and IBUFG inputs. An IPAD can be inferred by NGDBUILD if one is missing on an IBUF or IBUFG input.

Note The LOC attribute cannot be used on IPAD multiples.



IPAD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, it is not necessary to use these elements in the design. They will be added automatically.

Commonly Used Constraints

IOBDELAY, PULLDOWN

JTAGPPC

JTAG Primitive for the Power PC

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A

* Not supported for Virtex-II. Only supported for Virtex-II Pro.

The JTAGPPC block allows connection from the JTAG logic in the PPC405 core to the JTAG logic of the Virtex-II Pro device. The connections are made through programmable routing and so the connection only exists after configuration. Following is an example instantiation of the JTAGPPC block in Verilog:

```
JTAGPPC IJTAGPPC(.TDOTSPPC(TDO_TS_PPC),
   .TDOPPC(TDO_PPC),.TMS(TMS_PPC),
   .TDIPPC(TDI_PPC), .TCK(TCK_PPC));

PPC405 IPPC405 (
 ...
.JTGC405TCK     (TCK_PPC),
.JTGC405TDI     (TDI_PPC),
.JTGC405TMS     (TMS_PPC),
.C405JTGTDO     (TDO_PPC),
.C405JTGTDOEN (TDO_TS_PPC),
...
)
```

When the block is instantiated in this fashion, the instruction registers of the PPC405 and the Virtex-II Pro device are linked in series.

The following table lists the input and output pins for JTAGPPC.

Inputs	Outputs
TDOPPC	TCK
TDOTSPPC	TDIPPC
	TMS

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for JTAGPPC should be placed
-- after architecture statement but before begin keyword

component JTAGPPC
   port (TCK      : out STD_ULOGIC;
         TDIPPC   : out STD_ULOGIC;
         TMS      : out STD_ULOGIC;
         TDOPPC   : in STD_ULOGIC;
         TDOTSPPC : in STD_ULOGIC);
end component;

-- Component Attribute specification for JTAGPPC
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for JTAGPPC should be placed
-- in architecture after the begin keyword

JTAGPPC_INSTANCE_NAME : JTAGPPC
      port map  (TCK    => user_TCK,
                 TDIPPC => user_TDIPPC,
                 TMS    => user_TMS,
                 TDOPPC => user_TDOPPC,
                 TDOTSPPC=> user_TDOTSPPC);
```

Verilog Instantiation Template

```
JTAGPPC JTAGPPC_instance_name (.TCK (user_TCK),
                               .TDIPPC (user_TDIPPC),
                               .TMS (user_TMS),
                               .TDOPPC (user_TDOPPC),
                               .TDOTSPPC (user_TDOTSPPC));
```

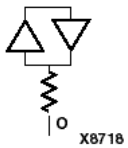
Commonly Used Constraints

None

KEEPER

KEEPER Symbol

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	Primitive



KEEPER is a weak keeper element used to retain the value of the net connected to its bidirectional O pin. For example, if a logic 1 is being driven onto the net, KEEPER drives a weak/resistive 1 onto the net. If the net driver is then 3-stated, KEEPER continues to drive a weak/resistive 1 onto the net.

For additional information on using a KEEPER element with SelectI/O components, see the "SelectI/O Usage Rules" in the "IBUF_*selectIO*" section

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for KEEPER should be placed
-- after architecture statement but before begin keyword

component KEEPER
   port (O : inout STD_ULOGIC);
end component;

-- Component Attribute specification for KEEPER
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for KEEPER should be placed
-- in architecture after the begin keyword

KEEPER_INSTANCE_NAME : KEEPER
      port map  (O => user_O);
```

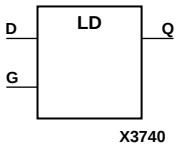
Verilog Instantiation Template

```
KEEPER KEEPER_instance_name (.O (user_O));
```

LD

Transparent Data Latch

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Primitive	Primitive



LD is a transparent data latch. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied.

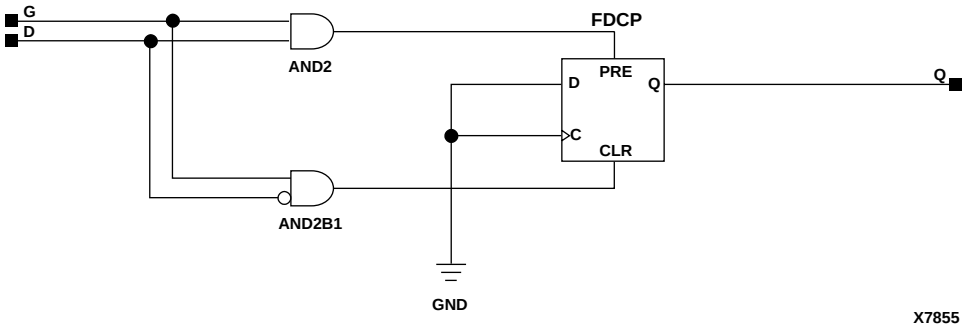
For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-Low but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
G	D	Q
1	0	0
1	1	1
0	X	No Chg
↓	D	d

d = state of input one setup time prior to High-to-Low gate transition



LD Implementation XC9500/XV/XL

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ld is
begin
 process (G,D)
   begin
    if (G = '1') then
      Q <= D;
    end if;
   end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (D or G) begin
 if (G)
   Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LD should be placed
-- after architecture statement but before begin keyword

component LD
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC);
end component;

-- Component Attribute specification for LD
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LD_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LD should be placed
-- in architecture after the begin keyword

LD_INSTANCE_NAME : LD
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G);
```

Verilog Instantiation Template

```
LD LD_instance_name (.Q (user_Q),
                     .D (user_D),
                    .G (user_G));

defparam LD_instance_name.INIT = bit_value;
```

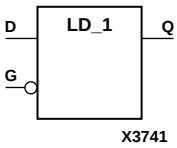
Commonly Used Constraints

INIT

LD_1

Transparent Data Latch with Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LD_1 is a transparent data latch with an inverted gate. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
G	D	Q
0	0	0
0	1	1
1	X	No Chg
↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ld_1 is
begin
process (G, D)
   begin
      if (G='0') then
        Q <= D;
      end if;
end process;

end Behavioral
```

Verilog Inference Code

```
always @(D or G) begin
      if (!G)
        Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LD_1 should be placed
-- after architecture statement but before begin keyword

component LD_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q  : out STD_ULOGIC;
         D  : in STD_ULOGIC;
         G  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LD_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LD_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LD_1 should be placed
-- in architecture after the begin keyword

LD_1_INSTANCE_NAME : LD_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G);
```

Verilog Instantiation Template

```
LD_1 LD_1_instance_name (.Q (user_Q),
                         .D (user_D),
                         .G (user_G));

defparam LD_1_instance_name.INIT = bit_value;
```

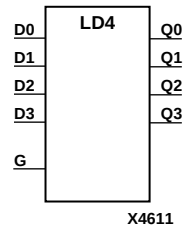
Commonly Used Constraints

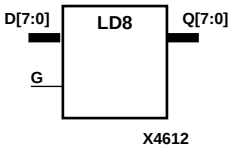
INIT

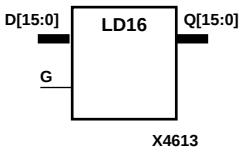
LD4, 8, 16

Multiple Transparent Data Latches

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







LD4, LD8, and LD16 have, respectively, 4, 8, and 16 transparent data latches with a common gate enable (G). The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

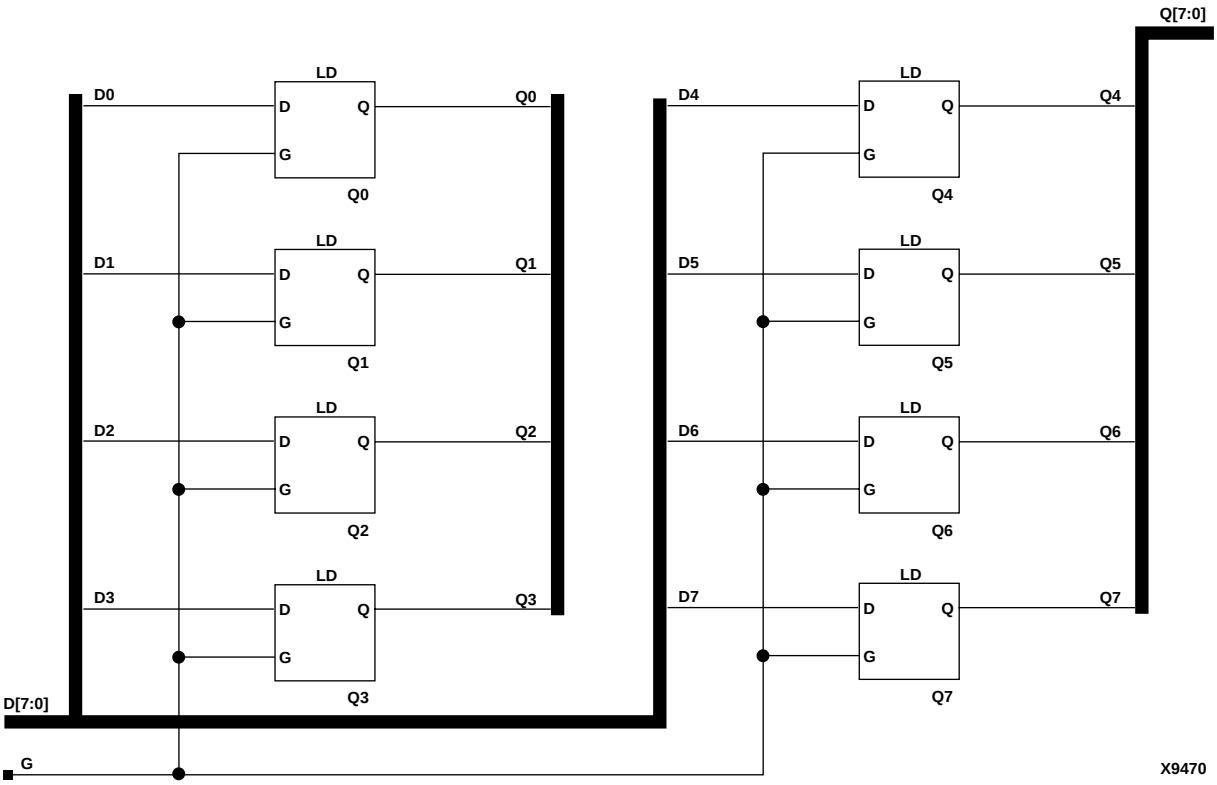
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

See the "LD" section for information on single transparent data latches.

Inputs		Outputs
G	D	Q
1	0	0
1	1	1
0	X	No Chg
↓	D	d

d = state of input one setup time prior to High-to-Low gate transition



LD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ld4 is
begin
 process (G,D)
  begin
  if (G = '1') then
  Q <= D;
   end if;
 end process;
end Behavioral;
```

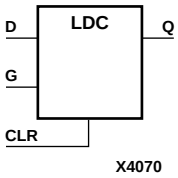
Verilog Inference Code

```
always @ (D or G) begin
 if (G)
   Q <= D;
end
```

LDC

Transparent Data Latch with Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Primitive	Primitive



LDC is a transparent data latch with asynchronous clear. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate enable (G) input is High and CLR is Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains low.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	G	D	Q
1	X	X	0
0	1	0	0
0	1	1	1
0	0	X	No Chg
0	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldc is
begin
   process (CLR, D, G)
      begin
        if (CLR='1') then
          Q <= '0';
        elsif (G='1') then
          Q <= D;
        end if;
   end process;
```

```
end Behavioral;
```

Verilog Inference Code

```
always @ (G or D or CLR) begin
   if (CLR)
      Q <= 0;
   else if (G)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDC should be placed
-- after architecture statement but before begin keyword

component LDC
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDC
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDC_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDC should be placed
-- in architecture after the begin keyword

LDC_INSTANCE_NAME : LDC
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 CLR => user_CLR,
                 D => user_D,
                 G => user_G);
```

Verilog Instantiation Template

```
LDC LDC_instance_name (.Q (user_Q),
                       .CLR (user_CLR),
                       .D (user_D),
                       .G (user_G));
```

```
defparam LDC_instance_name.INIT = bit_value;
```

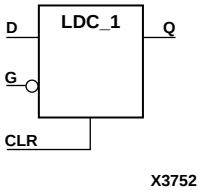
Commonly Used Constraints

INIT

LDC_1

Transparent Data Latch with Asynchronous Clear and Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDC_1 is a transparent data latch with asynchronous clear and inverted gate. When the asynchronous clear input (CLR) is High, it overrides the other inputs (D and G) and resets the data (Q) output Low. Q reflects the data (D) input while the gate enable (G) input and CLR are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	G	D	Q
1	X	X	0
0	0	0	0
0	0	1	1
0	1	X	No Chg
0	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldc_1 is
begin
   process (CLR, D, G)
      begin
        if (CLR='1') then
          Q <= '0';
        elsif (G='0') then
          Q <= D;
        end if;
   end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (G or D or CLR) begin
   if (CLR)
      Q <= 0;
   else if (!G)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDC_1 should be placed
-- after architecture statement but before begin keyword

component LDC_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDC_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDC_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDC_1 should be placed
-- in architecture after the begin keyword

LDC_1_INSTANCE_NAME : LDC_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
```

```
   -- synthesis translate_on
      port map  (Q => user_Q,
                 CLR => user_CLR,
                 D => user_D,
                 G => user_G);
```

Verilog Instantiation Template

```
LDC_1 LDC_1_instance_name (.Q (user_Q),
                           .CLR (user_CLR),
                           .D (user_D),
                           .G (user_G));

defparam LDC_1_instance_name.INIT = bit_value;
```

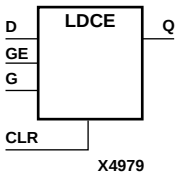
Commonly Used Constraints

INIT

LDCE

Transparent Data Latch with Asynchronous Clear and Gate Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDCE is a transparent data latch with asynchronous clear and gate enable. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High and CLR is Low. If GE is Low, data on D cannot be latched. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains low.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	GE	G	D	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	0	0
0	1	1	1	1
0	1	0	X	No Chg
0	1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldce is

begin

process (CLR, D, G, GE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (GE = '1') then
                if (G='1') then
                        Q <= D;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @(CLR or D or G or GE) begin
   if (CLR)
      Q <= 0;
   else if (G && GE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDCE should be placed
-- after architecture statement but before begin keyword

component LDCE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         GE   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDCE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDCE should be placed
```

```
-- in architecture after the begin keyword

LDCE_INSTANCE_NAME : LDCE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 CLR => user_CLR,
                 D => user_D,
                 G => user_G,
                 GE => user_GE);
```

Verilog Instantiation Template

```
LDCE LDCE_instance_name (.Q (user_Q),
                         .CLR (user_CLR),
                         .D (user_D),
                         .G (user_G),
                         .GE (user_GE));

defparam LDCE_instance_name.INIT = bit_value;
```

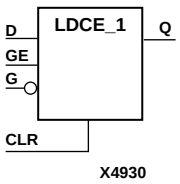
Commonly Used Constraints

INIT

LDCE_1

Transparent Data Latch with Asynchronous Clear, Gate Enable, and Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDCE_1 is a transparent data latch with asynchronous clear, gate enable, and inverted gate. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate (G) input and CLR are Low and gate enable (GE) is High. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High or GE remains Low.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	GE	G	D	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	0	0	0
0	1	0	1	1
0	1	1	X	No Chg
0	1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldce_1 is

begin

process (CLR, D, G, GE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (GE = '1') then
                if (G='0') then
                        Q <= D;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @(CLR or D or G or GE) begin
   if (CLR)
      Q <= 0;
   else if (!G && GE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDCE_1 should be placed
-- after architecture statement but before begin keyword

component LDCE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         GE   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDCE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDCE_1 should be placed
```

```
-- in architecture after the begin keyword

LDCE_1_INSTANCE_NAME : LDCE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 CLR => user_CLR,
                 D => user_D,
                 G => user_G,
                 GE => user_GE);
```

Verilog Instantiation Template

```
LDCE_1 LDCE_1_instance_name (.Q (user_Q),
                             .CLR (user_CLR),
                             .D (user_D),
                             .G (user_G),
                             .GE (user_GE));

defparam LDCE_1_instance_name.INIT = bit_value;
```

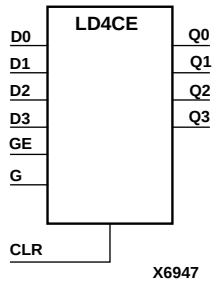
Commonly Used Constraints

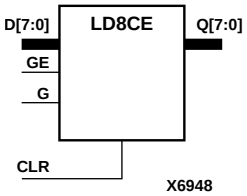
INIT

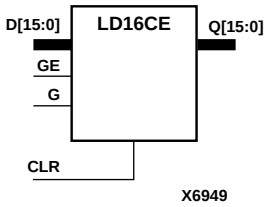
LD4CE, LD8CE, LD16CE

Transparent Data Latches with Asynchronous Clear and Gate Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A







LD4CE, LD8CE, and LD16CE have, respectively, 4, 8, and 16 transparent data latches with asynchronous clear and gate enable. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) outputs Low. Q reflects the data (D) inputs while the gate (G) and gate enable (GE) are High, and CLR is Low. If GE is Low, data on D cannot be latched. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

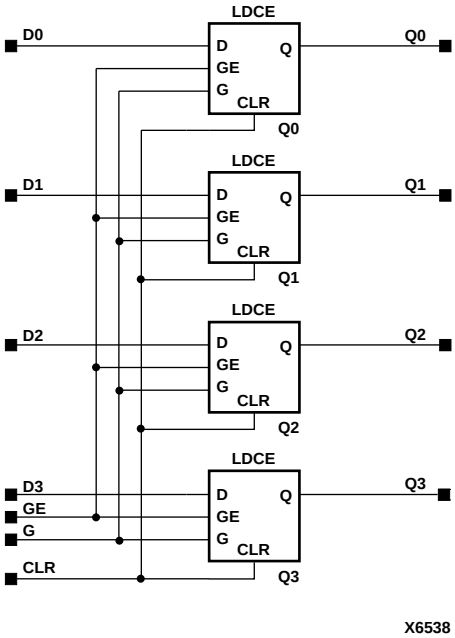
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	GE	G	Dn	Qn
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	1	1
0	1	1	0	0
0	1	0	X	No Chg
0	1	↓	Dn	dn

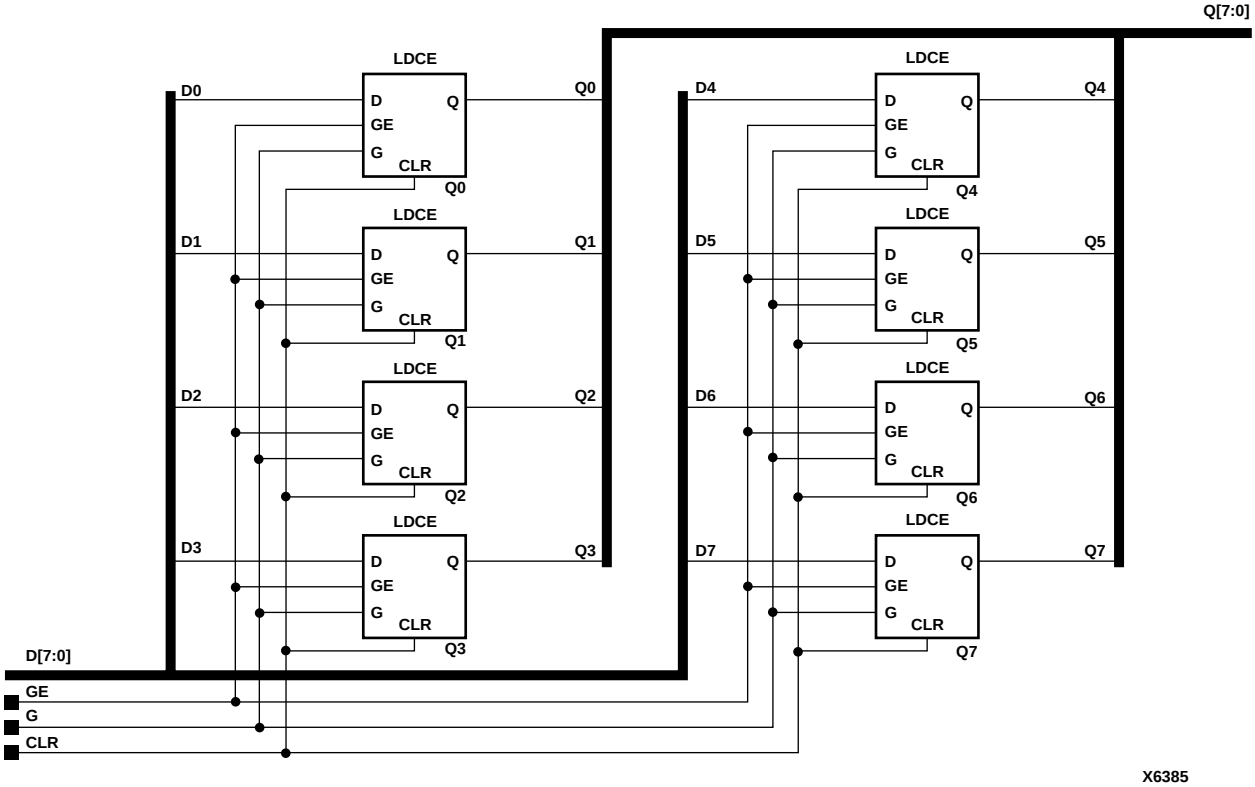
Dn = referenced input, for example, D0, D1, D2

Qn = referenced output, for example, Q0, Q1, Q2

dn = referenced input state, one setup time prior to High-to-Low gate transition



LD4CE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



LD8CE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are supported for inference only.

VHDL Inference Code

```
architecture Behavioral of ld4ce is

begin

process (CLR, D, G, GE)
begin
        if (CLR='1') then
                Q <= "0000";
        elsif (GE = '1') then
                if (G='1') then
                        Q <= D;
                end if;
        end if;
end process;

end Behavioral;
```

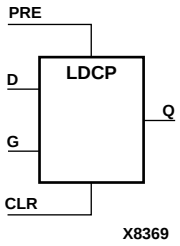
Verilog Inference Code

```
always @(CLR or D or G or GE) begin
   if (CLR)
      Q <= 0;
   else if (G && GE)
      Q <= D;
end
```

LDCP

Transparent Data Latch with Asynchronous Clear and Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Primitive	Primitive



LDCP is a transparent data latch with data (D), asynchronous clear (CLR) and preset (PRE) inputs. When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is low, it presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input is High and CLR and PRE are Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	PRE	G	D	Q
1	X	X	X	0
0	1	X	X	1
0	0	1	1	1
0	0	1	0	0
0	0	0	X	No Chg
0	0	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldcp is

begin

process (CLR, D, G, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (G='1') then
                Q <= D;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (CLR or PRE or D or G) begin
   if (CLR)
      Q <= 0;
   else if (PRE)
      Q <= 1;
   else if (G)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDCP should be placed
-- after architecture statement but before begin keyword

component LDCP
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDCP_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for LDCP should be placed
-- in architecture after the begin keyword

LDCP_INSTANCE_NAME : LDCP
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 CLR => user_CLR,
                 D => user_D,
                 G => user_G,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
LDCP LDCP_instance_name (.Q (user_Q),
                         .CLR (user_CLR),
                         .D (user_D),
                         .G (user_G),
                         .PRE (user_PRE));

defparam LDCP_instance_name.INIT = bit_value;
```

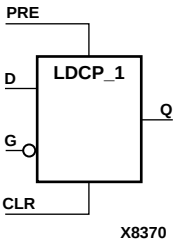
Commonly Used Constraints

INIT

LDCP_1

Transparent Data Latch with Asynchronous Clear and Preset and Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDCP_1 is a transparent data latch with data (D), asynchronous clear (CLR), preset (PRE) inputs, and inverted gate (G). When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while gate (G) input, CLR, and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	PRE	G	D	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	1	1
0	0	0	0	0
0	0	1	X	No Chg
0	0	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldcp_1 is

begin

process (CLR, D, G, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (G='0') then
                Q <= D;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (CLR or PRE or D or G) begin
   if (CLR)
      Q <= 0;
   else if (PRE)
      Q <= 1;
   else if (!G)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDCP_1 should be placed
-- after architecture statement but before begin keyword

component LDCP_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCP_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDCP_1_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for LDCP_1 should be placed
-- in architecture after the begin keyword

LDCP_1_INSTANCE_NAME : LDCP_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 CLR => user_CLR,
                 D => user_D,
                 G => user_G,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
LDCP_1 LDCP_1_instance_name (.Q (user_Q),
                             .CLR (user_CLR),
                             .D (user_D),
                             .G (user_G),
                             .PRE (user_PRE));

defparam LDCP_1_instance_name.INIT = bit_value;
```

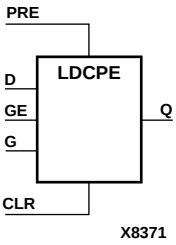
Commonly Used Constraints

INIT

LDCPE

Transparent Data Latch with Asynchronous Clear and Preset and Gate Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDCPE is a transparent data latch with data (D), asynchronous clear (CLR), asynchronous preset (PRE), and gate enable (GE). When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High and CLR and PRE are Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	PRE	GE	G	D	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	1	0	0
0	0	1	1	1	1
0	0	1	0	X	No Chg
0	0	1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldcpe is

begin

process (CLR, D, G, GE, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (GE = '1') then
                if (G='1') then
                        Q <= D;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (CLR or PRE or D or G or GE) begin
   if (CLR)
      Q <= 0;
   else if (PRE)
      Q <= 1;
   else if (G && GE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDCPE should be placed
-- after architecture statement but before begin keyword

component LDCPE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         GE   : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCPE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
```

```
attribute INIT of LDCPE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDCPE should be placed
-- in architecture after the begin keyword

LDCPE_INSTANCE_NAME : LDCPE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 CLR => user_CLR,
                 D => user_D,
                 G => user_G,
                 GE => user_GE,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
LDCPE LDCPE_instance_name (.Q (user_Q),
                           .CLR (user_CLR),
                           .D (user_D),
                           .G (user_G),
                           .GE (user_D),
                           .PRE (user_PRE));

defparam LDCPE_instance_name.INIT = bit_value;
```

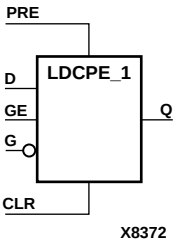
Commonly Used Constraints

INIT

LDCPE_1

Transparent Data Latch with Asynchronous Clear and Preset, Gate Enable, and Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDCPE_1 is a transparent data latch with data (D), asynchronous clear (CLR), asynchronous preset (PRE), gate enable (GE), and inverted gate (G). When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while gate enable (GE) is High and gate (G), CLR, and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G is High or GE is Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	PRE	GE	G	D	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	X	No Chg
0	0	1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldcpe_1 is

begin

process (CLR, D, G, GE, PRE)
begin
        if (CLR='1') then
                Q <= '0';
        elsif (PRE='1') then
                Q <= '1';
        elsif (GE = '1') then
                if (G='0') then
                        Q <= D;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (CLR or PRE or D or G or GE) begin
   if (CLR)
      Q <= 0;
   else if (PRE)
      Q <= 1;
   else if (!G && GE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDCPE_1 should be placed
-- after architecture statement but before begin keyword

component LDCPE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         CLR  : in STD_ULOGIC;
         D    : in STD_ULOGIC;
         G  : in STD_ULOGIC;
         GE   : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCPE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
```

```
attribute INIT of LDCPE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDCPE_1 should be placed
-- in architecture after the begin keyword

LDCPE_1_INSTANCE_NAME : LDCPE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 CLR => user_CLR,
                 D => user_D,
                 G => user_G,
                 GE => user_GE,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
LDCPE_1 LDCPE_1_instance_name (.Q (user_Q),
                               .CLR (user_CLR),
                               .D (user_D),
                               .G (user_G),
                               .GE (user_D),
                               .PRE (user_PRE));

defparam LDCPE_1_instance_name.INIT = bit_value;
```

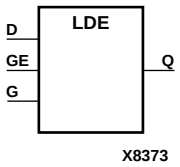
Commonly Used Constraints

INIT

LDE

Transparent Data Latch with Gate Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDE is a transparent data latch with data (D) and gate enable (GE) inputs. Output Q reflects the data (D) while the gate (G) input and gate enable (GE) are High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	0	0
1	1	1	1
1	0	X	No Chg
1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of lde is

begin

process (D, G, GE)
begin
        if (GE = '1' and G = '1') then
                Q <= D;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @(D or G or GE) begin
   if (G && GE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDE should be placed
-- after architecture statement but before begin keyword

component LDE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         GE   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDE should be placed
-- in architecture after the begin keyword

LDE_INSTANCE_NAME : LDE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
```

```
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G,
                 GE => user_GE);
```

Verilog Instantiation Template

```
LDE LDE_instance_name (.Q (user_Q),
                       .D (user_D),
                       .G (user_G),
                       .GE (user_GE));

defparam LDE_instance_name.INIT = bit_value;
```

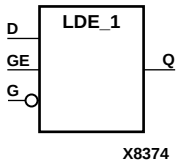
Commonly Used Constraints

INIT

LDE_1

Transparent Data Latch with Gate Enable and Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDE_1 is a transparent data latch with data (D), gate enable (GE), and inverted gate (G). Output Q reflects the data (D) while the gate (G) input is Low and gate enable (GE) is High. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G is High or GE is Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	0	0	0
1	0	1	1
1	1	X	No Chg
1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of lde_1 is

begin

process (D, G, GE)
begin
        if (GE = ’1’ and G = ’0’) then
                Q <= D;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (D or G or GE) begin
   if (!G && GE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDE_1 should be placed
-- after architecture statement but before begin keyword

component LDE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         GE   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDE_1 should be placed
-- in architecture after the begin keyword

LDE_1_INSTANCE_NAME : LDE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G,
                 GE => user_GE);
```

Verilog Instantiation Template

```
LDE_1 LDE_1_instance_name (.Q (user_Q),
                           .D (user_D),
                           .G (user_G),
                           .GE (user_GE));

defparam LDE_1_instance_name.INIT = bit_value;
```

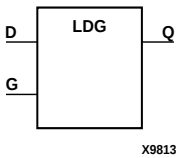
Commonly Used Constraints

INIT

LDG

Transparent Datagate Latch

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	N/A	N/A	N/A	Primitive



LDG is a transparent DataGate latch used for gating input signals to decrease power dissipation. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The D input(s) of the LDG must be connected to a device input pad(s) and must have no other fan-outs (must not branch). The CPLD fitter maps the G input to the device's DataGate Enable control pin (DGE). There must be no more than one DataGate Enable signal in the design. The DataGate Enable signal may be driven either by a device input pin or any on-chip logic source. The DataGate Enable signal may be reused by other ordinary logic in the design.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. Refer to the LDG4, LDG8, LDG16 section for information on multiple transparent datagate latches for the CoolRunner-II series.

Inputs		Outputs
G	D	Q
0	0	0
0	1	1
1	X	No Chg
↑	D	d

d = state of input one setup time prior to High-to-Low gate transition

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Inference Code

```
architecture Behavioral of ldg is
begin
 process (G,D)
  begin
  if (G = '0') then
  Q <= D;
   end if;
 end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (D or G) begin
 if (!G)
  Q <= D;
end
```

VHDL Instantiation Template

```
component LDG
       port (Q : out STD_ULOGIC;
             D : in STD_ULOGIC;
             G : in STD_ULOGIC);
     end component;
     ...
     begin
     ...
     INSTANCE_NAME : LDG
       port map (Q => user_Q,
                 D => user_D,
                 G => user_G);
```

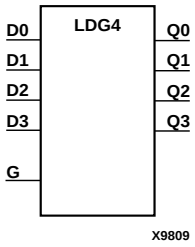
Verilog Instantiation Template

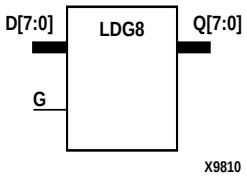
```
     LDG instance_name (.Q (user_Q),
                        .D (user_D),
                        .G (user_G));
```

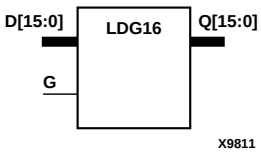
LDG4, 8, 16

Multiple Transparent Datagate Latches

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







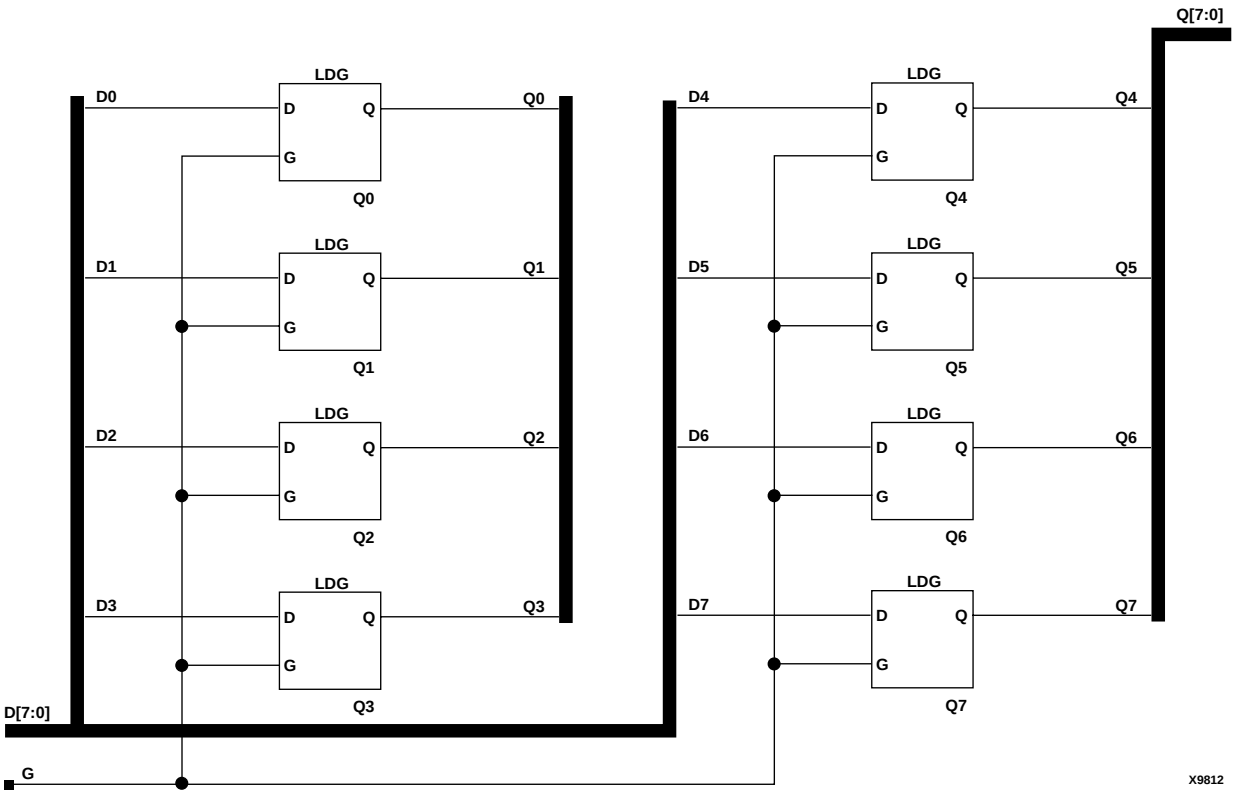
LDG4, LDG8, and LDG16 have, respectively, 4, 8, and 16 transparent DataGate latches with a common gate enable (G). These latches are used to gate input signals in order to decrease power dissipation during periods when activity on the input pins is not of interest to the CPLD. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The D input(s) of the LDG must be connected to a device input pad(s) and must have no other fan-outs (must not branch). The CPLD fitter maps the G input to the device's DataGate Enable control pin (DGE). There must be no more than one DataGate Enable signal in the design. The DataGate Enable signal may be driven either by a device input pin or any on-chip logic source. The DataGate Enable signal may be reused by other ordinary logic in the design.

The latch is asynchronously cleared, output Low, when power is applied. Refer to the LDG section for information on single transparent data latches.

Inputs		Outputs
G	D	Q
0	0	0
0	1	1
1	X	No Chg
↑	D	d

d = state of input one setup time prior to High-to-Low gate transition



LDG8 Implementation

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of ldg4 is
begin
 process (G,D)
  begin
  if (G = '0') then
    Q <= D;
   end if;
 end process;
end Behavioral;
```

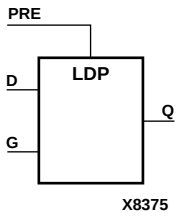
Verilog Inference Code

```
always @ (D or G) begin
  if (!G)
    Q <= D;
end
```

LDP

Transparent Data Latch with Asynchronous Preset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Macro	Primitive	Primitive



LDP is a transparent data latch with asynchronous preset (PRE). When the PRE input is High, it overrides the other inputs and resets the data (Q) output High. Q reflects the data (D) input while gate (G) input is High and PRE is Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-Low but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	G	D	Q
1	X	X	1
0	1	0	0
0	1	1	1
0	0	X	No Chg
0	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldp is
begin
   process (PRE, G)
      begin
        if (PRE='1') then
          Q <= "1111";
        elsif (G='1') then
          Q <= D;
        end if;
   end process;
end Behavioral;
```

Verilog Inference Code

```
always @ (PRE or D or G) begin
   if (PRE)
     Q <= 1;
   else if (G)
     Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDP should be placed
-- after architecture statement but before begin keyword

component LDP
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDP_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDP should be placed
-- in architecture after the begin keyword

LDP_INSTANCE_NAME : LDP
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
LDP LDP_instance_name (.Q (user_Q),
                       .D (user_D),
                       .G (user_G),
                       .PRE (user_PRE));

defparam LDP_instance_name.INIT = bit_value;
```

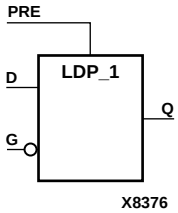
Commonly Used Constraints

INIT

LDP_1

Transparent Data Latch with Asynchronous Preset and Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDP_1 is a transparent data latch with asynchronous preset (PRE) and inverted gate (G). When the PRE input is High, it overrides the other inputs and presets the data (Q) output High. Q reflects the data (D) input while gate (G) input and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	G	D	Q
1	X	X	1
0	0	0	0
0	0	1	1
0	1	X	No Chg
0	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldp_1is
begin
   process (PRE, G)
      begin
        if (PRE='1') then
          Q <= "1111";
        elsif (G='0') then
          Q <= D;
        end if;
   end process;
end Behavioral;
```

Verilog Inference Code

```
   always @ (PRE or D or G) begin
      if (PRE)
        Q <= 1;
      else if (!G)
        Q <= D;
   end
```

VHDL Instantiation Template

```
-- Component Declaration for LDP_1 should be placed
-- after architecture statement but before begin keyword

component LDP_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDP_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDP_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDP_1 should be placed
-- in architecture after the begin keyword

LDP_1_INSTANCE_NAME : LDP_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
```

```
   -- synthesis translate_on
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
LDP_1 LDP_1_instance_name (.Q (user_Q),
                           .D (user_D),
                           .G (user_G),
                           .PRE (user_PRE));

defparam LDP_1_instance_name.INIT = bit_value;
```

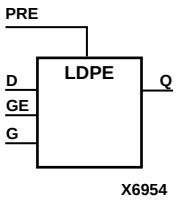
Commonly Used Constraints

INIT

LDPE

Transparent Data Latch with Asynchronous Preset and Gate Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDPE is a transparent data latch with asynchronous preset and gate enable. When the asynchronous preset (PRE) is High, it overrides the other input and presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
PRE	GE	G	D	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	1	0	0
0	1	1	1	1
0	1	0	X	No Chg
0	1	↓	D	d

d = state of input one setup time prior to High-to-Low gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldpe is

begin

process (D, G, GE, PRE)
begin
        if (PRE='1') then
                Q <= '1';
        elsif (GE = '1') then
                if (G='1') then
                        Q <= D;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (PRE or D or G or GE) begin
   if (PRE)
      Q <= 1;
   else if (G && GE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDPE should be placed
-- after architecture statement but before begin keyword

component LDPE
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         GE   : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDPE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDPE_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDPE should be placed
```

```
-- in architecture after the begin keyword

LDPE_INSTANCE_NAME : LDPE
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G,
                 GE => user_GE,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
LDPE LDPE_instance_name (.Q (user_Q),
                         .D (user_D),
                         .G (user_G),
                         .GE (user_GE),
                         .PRE (user_PRE));

defparam LDPE_instance_name.INIT = bit_value;
```

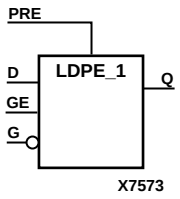
Commonly Used Constraints

INIT

LDPE_1

Transparent Data Latch with Asynchronous Preset, Gate Enable, and Inverted Gate

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



LDPE_1 is a transparent data latch with asynchronous preset, gate enable, and inverted gates. When the asynchronous preset (PRE) is High, it overrides the other input and presets the data (Q) output High. Q reflects the data (D) input while the gate (G) and PRE are Low and gate enable (GE) is High.

The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High or GE remains Low.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
PRE	GE	G	D	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	0	0
0	1	0	1	1
0	1	1	X	No Chg
0	1	↑	D	d

d = state of input one setup time prior to Low-to-High gate transition

Usage

This design element typically should be inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes need to be applied.

VHDL Inference Code

```
architecture Behavioral of ldpe_1 is

begin

process (D, G, GE, PRE)
begin
        if (PRE='1') then
                Q <= '1';
        elsif (GE = '1') then
                if (G='0') then
                        Q <= D;
                end if;
        end if;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (PRE or D or G or GE) begin
   if (PRE)
      Q <= 1;
   else if (!G && GE)
      Q <= D;
end
```

VHDL Instantiation Template

```
-- Component Declaration for LDPE_1 should be placed
-- after architecture statement but before begin keyword

component LDPE_1
   -- synthesis translate_off
   generic (
        INIT  : bit := '1');
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         D    : in STD_ULOGIC;
         G    : in STD_ULOGIC;
         GE   : in STD_ULOGIC;
         PRE  : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDPE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDPE_1_instance_name : label is "0";
-- values can be (0 or 1)


-- Component Instantiation for LDPE_1 should be placed
```

```
-- in architecture after the begin keyword

LDPE_1_INSTANCE_NAME : LDPE_1
   -- synthesis translate_off
   generic map(
        INIT => bit_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 D => user_D,
                 G => user_G,
                 GE => user_GE,
                 PRE => user_PRE);
```

Verilog Instantiation Template

```
LDPE_1 LDPE_1_instance_name (.Q (user_Q),
                             .D (user_D),
                             .G (user_G),
                             .GE (user_GE),
                             .PRE (user_PRE));

defparam LDPE_1_instance_name.INIT = bit_value;
```

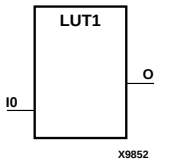
Commonly Used Constraints

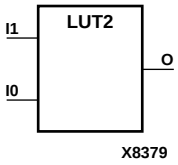
INIT

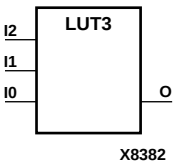
LUT1, 2, 3, 4

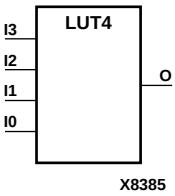
1-, 2-, 3-, 4-Bit Look-Up-Table with General Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N / A	N / A	N / A









LUT1, LUT2, LUT3, and LUT4 are, respectively, 1-, 2-, 3-, and 4-bit look-up-tables (LUTs) with general output (O).

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1 provides a look-up-table version of a buffer or inverter.

LUTs are the basic Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, and Spartan-IIE building blocks. Two LUTs are available in each CLB slice; four LUTs are available in each CLB. The variants, "LUT1_D, LUT2_D, LUT3_D, LUT4_D" and "LUT1_L, LUT2_L, LUT3_L, LUT4_L" provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

LUT3 Function Table

Inputs			Outputs
I2	I1	I0	O
0	0	0	INIT[0]
0	0	1	INIT[1]
0	1	0	INIT[2]
0	1	1	INIT[3]
1	0	0	INIT[4]
1	0	1	INIT[5]
1	1	0	INIT[6]
1	1	1	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

Usage

LUTs are generally inferred with the logic portions of the HDL code. Xilinx suggests that you instantiate LUTs only if you have a need to implicitly specify the logic mapping, or if you need to manually place or relationally place the logic.

VHDL Instantiation Template for LUT1

```
-- Component Declaration for LUT1 should be placed
-- after architecture statement but before begin keyword

component LUT1
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"2");
   -- synthesis translate_on
   port (O    : out STD_ULOGIC;
         IO   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT1_instance_name : label is "2";
-- values can be 0, 1, 2,or 3


-- Component Instantiation for LUT1 should be placed
-- in architecture after the begin keyword

LUT1_INSTANCE_NAME : LUT1
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (O => user_O,
                 IO => user_IO);
```

Verilog Instantiation Template For LUT1

```
LUT1 LUT1_instance_name (.O (user_O),
                         .IO (user_IO));

defparam LUT1_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT2

```
-- Component Declaration for LUT2 should be placed
-- after architecture statement but before begin keyword

component LUT2
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"4");
   -- synthesis translate_on
   port (O    : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for LUT2
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT2_instance_name : label is "4";
-- values can be 0, 1, 2,3, or 4


-- Component Instantiation for LUT2 should be placed
-- in architecture after the begin keyword

LUT2_INSTANCE_NAME : LUT2
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (O => user_O,
                 I0 => user_I0,
                 I1 => user_I1);
```

Verilog Instantiation Template For LUT2

```
LUT2 LUT2_instance_name (.O (user_O),
                         .I0 (user_I0),
                         .I1 (user_I1));

defparam LUT2_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT3

```
-- Component Declaration for LUT3 should be placed
-- after architecture statement but before begin keyword

component LUT3
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"8");
   -- synthesis translate_on
   port (O    : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT3
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT3_instance_name : label is "8";
-- values can be 0, 1, 2, 3, 4, 5, 6, 7, 8
```

```
-- Component Instantiation for LUT3 should be placed
-- in architecture after the begin keyword --

LUT3_INSTANCE_NAME : LUT3
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (O => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2);
```

Verilog Instantiation Template For LUT3

```
LUT3 LUT3_instance_name (.O (user_O),
                         .I0 (user_I0),
                         .I1 (user_I1),
                         .I2 (user_I2));

defparam LUT4_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT4

```
-- Component Declaration for LUT4 should be placed
-- after architecture statement but before begin keyword

component LUT4
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"16");
   -- synthesis translate_on
   port (O    : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC;
         I3   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT4
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT4_instance_name : label is "16";
-- values can be 0 through 16


-- Component Instantiation for LUT4 should be placed
-- in architecture after the begin keyword

LUT4_INSTANCE_NAME : LUT4
   -- synthesis translate_off
```

```
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (O => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3);
```

Verilog Instantiation Template For LUT4

```
LUT4 LUT4_instance_name (.O (user_O),
                         .I0 (user_I0),
                         .I1 (user_I1),
                         .I2 (user_I2),
                         .I3 (user_I3));

defparam LUT4_instance_name.INIT = hex_value;
```

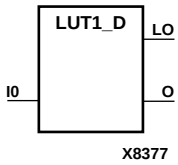
Commonly Used Constraints

BEL, INIT, LOC, U_SET

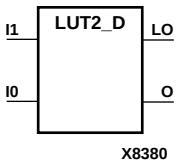
LUT1_D, LUT2_D, LUT3_D, LUT4_D

1-, 2-, 3-, 4-Bit Look-Up-Table with Dual Output

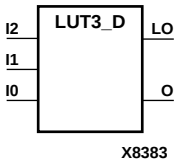
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N / A	N / A	N / A



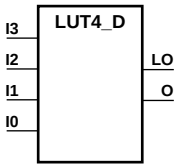
X8377



X8380



X8383



X8386

LUT1_D, LUT2_D, LUT3_D, and LUT4_D are, respectively, 1-, 2-, 3-, and 4-bit look-up-tables (LUTs) with two functionally identical outputs, O and LO. The O output is a general interconnect. The LO output is used to connect to another output within the same CLB slice and to the fast connect buffer.

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1_D provides a look-up-table version of a buffer or inverter.

See also "LUT1, 2, 3, 4" and "LUT1_L, LUT2_L, LUT3_L, LUT4_L."

LUT3_D Function Table

Inputs			Outputs	
I2	I1	I0	O	LO
0	0	0	INIT[0]	INIT[0]
0	0	1	INIT[1]	INIT[1]
0	1	0	INIT[2]	INIT[2]
0	1	1	INIT[3]	INIT[3]
1	0	0	INIT[4]	INIT[4]
1	0	1	INIT[5]	INIT[5]
1	1	0	INIT[6]	INIT[6]
1	1	1	INIT[7]	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

Usage

LUTs are generally inferred with the logic portions of the HDL code. Xilinx suggests that you instantiate LUTs only if you have a need to implicitly specify the logic mapping, or if you need to manually place or relationally place the logic.

VHDL Instantiation Template for LUT1_D

```
-- Component Declaration for LUT1_D should be placed
-- after architecture statement but before begin keyword

component LUT1_D
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"2");
   -- synthesis translate_on
   port (LO   : out STD_ULOGIC;
         O    : out STD_ULOGIC;
         I0   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT1_D
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT1_D_instance_name : label is "2";
-- values can be 0, 1, 2,or 3


-- Component Instantiation for LUT1_D should be placed
-- in architecture after the begin keyword

LUT1_D_INSTANCE_NAME : LUT1_D
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (LO => user_LO,
                 O => user_O,
                 IO => user_IO);
```

Verilog Instantiation Template For LUT1_D

```
LUT1_D LUT1_D_instance_name (.LO (user_LO),
                             .O (user_O),
                             .IO (user_IO));

defparam LUT1_D_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT2_D

```
-- Component Declaration for LUT2_D should be placed
-- after architecture statement but before begin keyword

component LUT2_D
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"4");
   -- synthesis translate_on
   port (LO   : out STD_ULOGIC;
         O    : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT2_D
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT2_D_instance_name : label is "4";



-- Component Instantiation for LUT2_D should be placed
-- in architecture after the begin keyword

LUT2_D_INSTANCE_NAME : LUT2_D
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (LO => user_LO,
                 O => user_O,
                 I0 => user_I0,
                 I1 => user_I1);
```

Verilog Instantiation Template For LUT2_D

```
LUT2_D LUT2_D_instance_name (.LO (user_O),
                             .O (user_O),
                             .I0 (user_I0),
                             .I1 (user_I1));

defparam LUT2_D_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT3_D

```
-- Component Declaration for LUT3_D should be placed
-- after architecture statement but before begin keyword

component LUT3_D
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"8");
   -- synthesis translate_on
   port (LO   : out STD_ULOGIC;
         O    : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT3_D
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT3_D_instance_name : label is "8";



-- Component Instantiation for LUT3_D should be placed
-- in architecture after the begin keyword

LUT3_D_INSTANCE_NAME : LUT3_D
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (LO => user_LO,
                 O => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2);
```

Verilog Instantiation Template For LUT3_D

```
LUT3_D LUT3_D_instance_name (.LO (user_LO),
                             .O (user_O),
                             .I0 (user_I0),
                             .I1 (user_I1),
                             .I2 (user_I2));

defparam LUT3_D_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT4_D

```
-- Component Declaration for LUT4_D should be placed
-- after architecture statement but before begin keyword

component LUT4_D
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"16");
   -- synthesis translate_on
   port (LO   : out STD_ULOGIC;
         O    : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC;
         I3   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT4_D
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT4_D_instance_name : label is "16";
-- values can be 0 through 16


-- Component Instantiation for LUT4_D should be placed
-- in architecture after the begin keyword

LUT4_D_INSTANCE_NAME : LUT4_D
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (LO => user_LO,
                 O => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3);
```

Verilog Instantiation Template For LUT4_D

```
LUT4_D LUT4_D_instance_name (.LO (user_LO),
                             .O (user_O),
                             .I0 (user_I0),
                             .I1 (user_I1),
                             .I2 (user_I2),
                             .I3 (user_I3));

defparam LUT4_D_instance_name.INIT = hex_value;
```

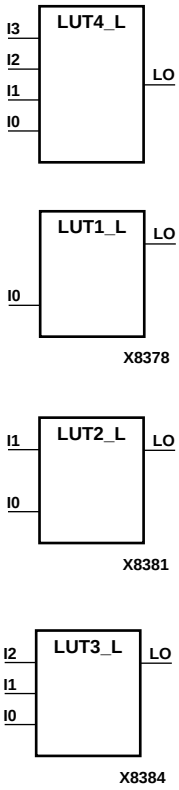
Commonly Used Constraints

INIT, LOC, RLOC, BEL, U_SET

LUT1_L, LUT2_L, LUT3_L, LUT4_L

1-, 2-, 3-, 4-Bit Look-Up-Table with Local Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N / A	N / A	N / A



LUT1_L, LUT2_L, LUT3_L, and LUT4_L are, respectively, 1-, 2-, 3-, and 4- bit look-up-tables (LUTs) with a local output (LO) that is used to connect to another output within the same CLB slice and to the fast connect buffer.

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1_L provides a look-up-table version of a buffer or inverter.

See also "LUT1, 2, 3, 4" and "LUT1_D, LUT2_D, LUT3_D, LUT4_D."

LUT3_L Function Table

Inputs			Outputs
I2	I1	I0	LO
0	0	0	INIT[0]
0	0	1	INIT[1]
0	1	0	INIT[2]
0	1	1	INIT[3]
1	0	0	INIT[4]
1	0	1	INIT[5]
1	1	0	INIT[6]
1	1	1	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

Usage

LUTs are generally inferred with the logic portions of the HDL code. Xilinx suggests that you instantiate LUTs only if you have a need to implicitly specify the logic mapping, or if you need to manually place or relationally place the logic.

VHDL Instantiation Template for LUT1_L

```
-- Component Declaration for LUT1_L should be placed
-- after architecture statement but before begin keyword

component LUT1_L
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"2");
   -- synthesis translate_on
```

```
   port (LO   : out STD_ULOGIC;
         IO   : in  STD_ULOGIC);
end component;

-- Component Attribute specification for LUT1_L
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT1_L_instance_name : label is "2";
-- values can be 0, 1, 2,or 3


-- Component Instantiation for LUT1_L should be placed
-- in architecture after the begin keyword

LUT1_L_INSTANCE_NAME : LUT1_L
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (LO => user_LO,
                 IO => user_IO);
```

Verilog Instantiation Template For LUT1_L

```
LUT1_L LUT1_L_instance_name (.LO (user_LO),
                             .IO (user_IO));

defparam LUT1_L_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT2_L

```
-- Component Declaration for LUT2_L should be placed
-- after architecture statement but before begin keyword

component LUT2_L
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"4");
   -- synthesis translate_on
   port (LO   : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT2_L
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT2_L_instance_name : label is "4";
```

```
-- Component Instantiation for LUT2_L should be placed
-- in architecture after the begin keyword

LUT2_L_INSTANCE_NAME : LUT2_L
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (LO => user_LO,
                 I0 => user_I0,
                 I1 => user_I1);
```

Verilog Instantiation Template For LUT2_L

```
LUT2_L LUT2_L_instance_name (.LO (user_O),
                             .I0 (user_I0),
                             .I1 (user_I1));

defparam LUT2_L_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT3_L

```
-- Component Declaration for LUT3_L should be placed
-- after architecture statement but before begin keyword

component LUT3_L
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"8");
   -- synthesis translate_on
   port (LO   : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT3_L
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT3_L_instance_name : label is "8";



-- Component Instantiation for LUT3_L should be placed
-- in architecture after the begin keyword --

LUT3_L_INSTANCE_NAME : LUT3_L
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
```

```
      port map  (LO => user_LO,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2);
```

Verilog Instantiation Template For LUT3_L

```
LUT3_L LUT3_L_instance_name (.LO (user_LO),
                             .I0 (user_I0),
                             .I1 (user_I1),
                             .I2 (user_I2));

defparam LUT3_L_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT4_L

```
-- Component Declaration for LUT4_L should be placed
-- after architecture statement but before begin keyword --

component LUT4_L
   -- synthesis translate_off
   generic (
        INIT  : bit_vector := X"16");
   -- synthesis translate_on
   port (LO   : out STD_ULOGIC;
         IO   : in STD_ULOGIC;
         I1   : in STD_ULOGIC;
         I2   : in STD_ULOGIC;
         I3   : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT4_L
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT4_L_instance_name : label is "16";
-- values can be 0 through 16


-- Component Instantiation for LUT4_L should be placed
-- in architecture after the begin keyword --

LUT4_L_INSTANCE_NAME : LUT4_L
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (LO => user_LO,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3);
```

Verilog Instantiation Template For LUT4_L

```
LUT4_L LUT4_L_instance_name (.LO (user_LO),
                             .I0 (user_I0),
                             .I1 (user_I1),
                             .I2 (user_I2),
                             .I3 (user_I3));

defparam LUT4_L_instance_name.INIT = hex_value;
```

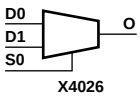
Commonly Used Constraints

INIT, LOC, RLOC, BEL, U_SET

M2_1

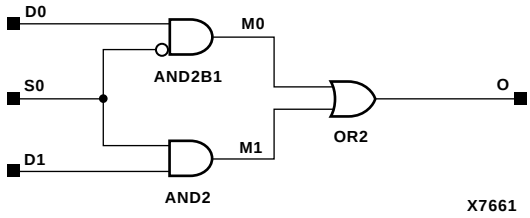
2-to-1 Multiplexer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



The M2_1 multiplexer chooses one data bit from two sources (D1 or D0) under the control of the select input (S0). The output (O) reflects the state of the selected data input. When Low, S0 selects D0 and when High, S0 selects D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	1
1	0	X	0
0	X	1	1
0	X	0	0



M2_1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of m2_1 is

begin

process (D0,D1,S0)
begin
        case S0 is
                when '0' => O <= D0;
                when '1' => O <= D1;
                when others => NULL;
        end case;
end process;

end Behavioral;
```

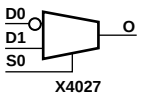
Verilog Inference Code

```
always @(D0 or D1 or S0)
begin
        case (S0)
                0 : O <= D0;
                1 : O <= D1;
        endcase
end
```

M2_1B1

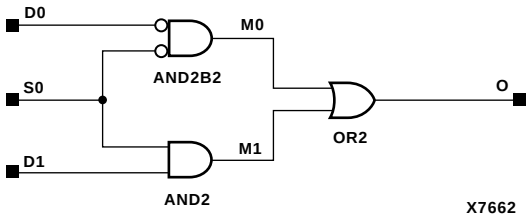
2-to-1 Multiplexer with D0 Inverted

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



The M2_1B1 multiplexer chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When S0 is Low, the output (O) reflects the inverted value of D0. When S0 is High, O reflects the state of D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	1
1	0	X	0
0	X	1	0
0	X	0	1



M2_1B1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of m2_1b1 is

begin

process (D0,D1,S0)
begin
        case S0 is
                when '0' => O <= not D0;
                when '1' => O <= D1;
                when others => NULL;
        end case;
end process;

end Behavioral;
```

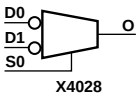
Verilog Inference Code

```
always @(D0 or D1 or S0)

begin

    case (S0)

        0 : O <= !D0;

        1 : O <= D1;

    endcase

end
```

M2_1B2

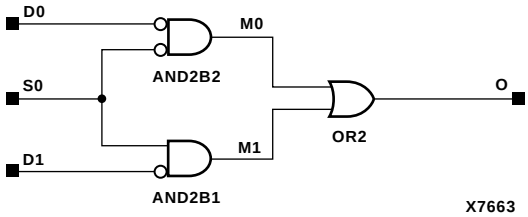
2-to-1 Multiplexer with D0 and D1 Inverted

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



The M2_1B2 multiplexer chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When S0 is Low, the output (O) reflects the inverted value of D0. When S0 is High, O reflects the inverted value of D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	0
1	0	X	1
0	X	1	0
0	X	0	1



M2_1B2 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of m2_1b2 is

begin

process (D0,D1,S0)
begin
        case S0 is
                when '0' => O <= not D0;
                when '1' => O <= not D1;
                when others => NULL;
        end case;
end process;

end Behavioral;
```

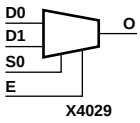
Verilog Inference Code

```
always @(D0 or D1 or S0)
begin
        case (S0)
                0 : O <= !D0;
                1 : O <= !D1;
        endcase
end
```

M2_1E

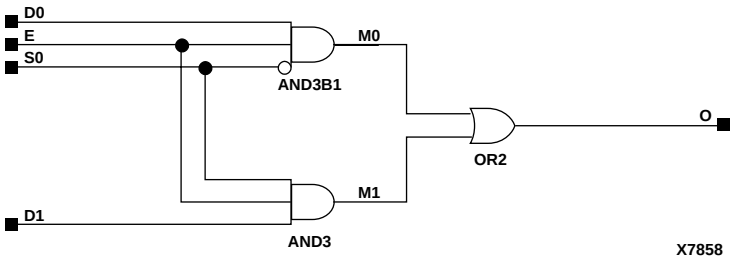
2-to-1 Multiplexer with Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



M2_1E is a 2-to-1 multiplexer with enable. When the enable input (E) is High, the M2_1E chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When Low, S0 selects D0 and when High, S0 selects D1. When E is Low, the output is Low.

Inputs				Outputs
E	S0	D1	D0	O
0	X	X	X	0
1	0	X	1	1
1	0	X	0	0
1	1	1	X	1
1	1	0	X	0



M2_1E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of m2_1e is

begin

process (D0,D1,E,S0)
begin
        if (E='0') then
                O <= '0';
        else
                case S0 is
                        when '0' => O <= D0;
                        when '1' => O <= D1;
                        when others => NULL;
                end case;
        end if;
end process;

end Behavioral;
```

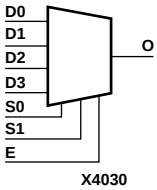
Verilog Inference Code

```
begin
        if (!E)
                O <= 0;
        else
        begin
                case (S0)
                        0 : O <= D0;
                        1 : O <= D1;
                endcase
        end
end
```

M4_1E

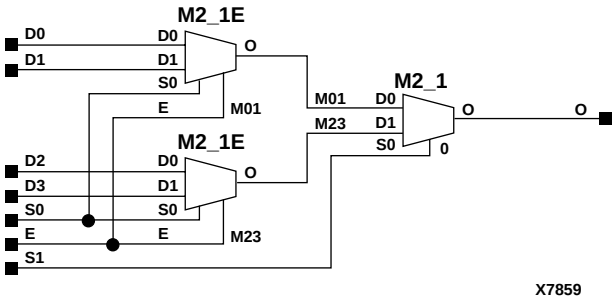
4-to-1 Multiplexer with Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro

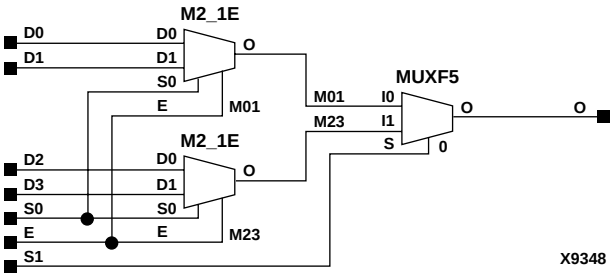


M4_1E is an 4-to-1 multiplexer with enable. When the enable input (E) is High, the M4_1E multiplexer chooses one data bit from four sources (D3, D2, D1, or D0) under the control of the select inputs (S1 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

Inputs							Outputs
E	S1	S0	D0	D1	D2	D3	O
0	X	X	X	X	X	X	0
1	0	0	D0	X	X	X	D0
1	0	1	X	D1	X	X	D1
1	1	0	X	X	D2	X	D2
1	1	1	X	X	X	D3	D3



M4_1E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



M4_1E Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of m4_1e is

begin

process (D,E,S)
begin
        if (E='0') then
                O <= '0';
        else
                case S is
                        when "00" => O <= D(0);
                        when "01" => O <= D(1);
                        when "10" => O <= D(2);
                        when "11" => O <= D(3);
                        when others => NULL;
                end case;
        end if;
end process;


end Behavioral;
```

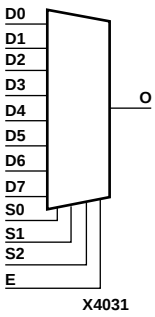
Verilog Inference Code

```
always @(D or E or S)
begin
        if (!E)
                O <= 0;
        else
        begin
                case (S)
                        0 : O <= D[0];
                        1 : O <= D[1];
                        2 : O <= D[2];
                        3 : O <= D[3];
                endcase
        end
end
```

M8_1E

8-to-1 Multiplexer with Enable

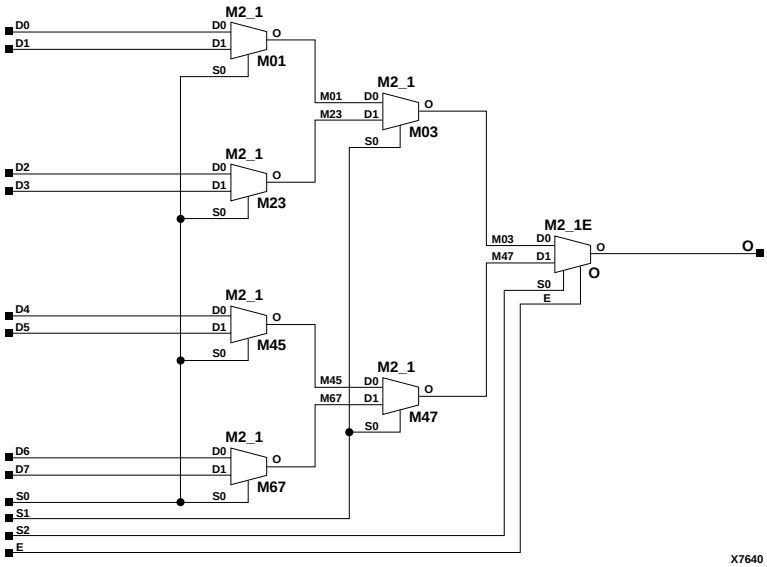
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



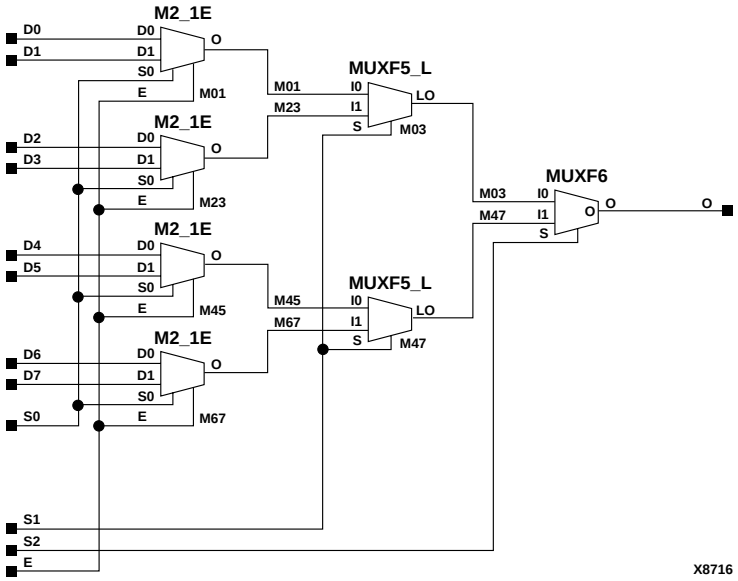
M8_1E is an 8-to-1 multiplexer with enable. When the enable input (E) is High, the M8_1E multiplexer chooses one data bit from eight sources (D7 – D0) under the control of the select inputs (S2 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

Inputs					Outputs
E	S2	S1	S0	D7 – D0	O
0	X	X	X	X	0
1	0	0	0	D0	D0
1	0	0	1	D1	D1
1	0	1	0	D2	D2
1	0	1	1	D3	D3
1	1	0	0	D4	D4
1	1	0	1	D5	D5
1	1	1	0	D6	D6
1	1	1	1	D7	D7

Dn represents signal on the Dn input; all other data inputs are don't-cares (X).



M8_1E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



M8_1E Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of m8_1e is

begin

process (D,E,S)
begin
        if (E='0') then
                O <= '0';
        else
                case S is
                        when "000" => O <= D(0);
                        when "001" => O <= D(1);
                        when "010" => O <= D(2);
                        when "011" => O <= D(3);
                        when "100" => O <= D(4);
                        when "101" => O <= D(5);
                        when "110" => O <= D(6);
                        when "111" => O <= D(7);
                        when others => NULL;
                end case;
        end if;
end process;

end Behavioral;
```

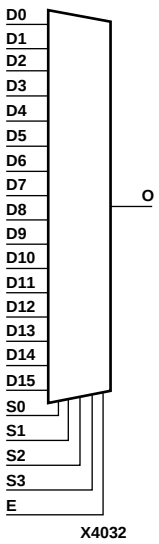
Verilog Inference Code

```
always @(D or E or S)
begin
        if (!E)
                O <= 0;
        else
        begin
                case (S)
                        0 : O <= D[0];
                        1 : O <= D[1];
                        2 : O <= D[2];
                        3 : O <= D[3];
                        4 : O <= D[4];
                        5 : O <= D[5];
                        6 : O <= D[6];
                        7 : O <= D[7];
                endcase
        end
end
```

M16_1E

16-to-1 Multiplexer with Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



M16_1E is a 16-to-1 multiplexer with enable. When the enable input (E) is High, the M16_1E multiplexer chooses one data bit from 16 sources (D15 – D0) under the control of the select inputs (S3 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

Inputs						Outputs
E	S3	S2	S1	S0	D15 – D0	O
0	X	X	X	X	X	0
1	0	0	0	0	D0	D0
1	0	0	0	1	D1	D1
1	0	0	1	0	D2	D2
1	0	0	1	1	D3	D3
. . .	. . .	. . .	. . .	. . .	. . .	. . .
1	1	1	0	0	D12	D12
1	1	1	0	1	D13	D13
1	1	1	1	0	D14	D14
1	1	1	1	1	D15	D15

Dn represents signal on the Dn input; all other data inputs are don't-cares (X).

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of m16_1e is

begin

process (D,E,S)
begin
        if (E='0') then
                O <= '0';
        else
                case S is
                        when "0000" => O <= D(0);
                        when "0001" => O <= D(1);
                        when "0010" => O <= D(2);
                        when "0011" => O <= D(3);
                        when "0100" => O <= D(4);
                        when "0101" => O <= D(5);
                        when "0110" => O <= D(6);
                        when "0111" => O <= D(7);
                        when "1000" => O <= D(8);
                        when "1001" => O <= D(9);
                        when "1010" => O <= D(10);
                        when "1011" => O <= D(11);
                        when "1100" => O <= D(12);
                        when "1101" => O <= D(13);
                        when "1110" => O <= D(14);
                        when "1111" => O <= D(15);
                        when others => NULL;
                end case;
        end if;
end process;

end Behavioral;
```

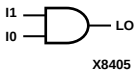
Verilog Inference Code

```
always @(D or E or S)
begin
        if (!E)
                O <= 0;
        else
        begin
                case (S)
                        0 : O <= D[0];
                        1 : O <= D[1];
                        2 : O <= D[2];
                        3 : O <= D[3];
                        4 : O <= D[4];
                        5 : O <= D[5];
                        6 : O <= D[6];
                        7 : O <= D[7];
                        8 : O <= D[8];
                        9 : O <= D[9];
                        10 : O <= D[10];
                        11 : O <= D[11];
                        12 : O <= D[12];
                        13 : O <= D[13];
                        14 : O <= D[14];
                        15 : O <= D[15];
                endcase
        end
end
```

MULT_AND

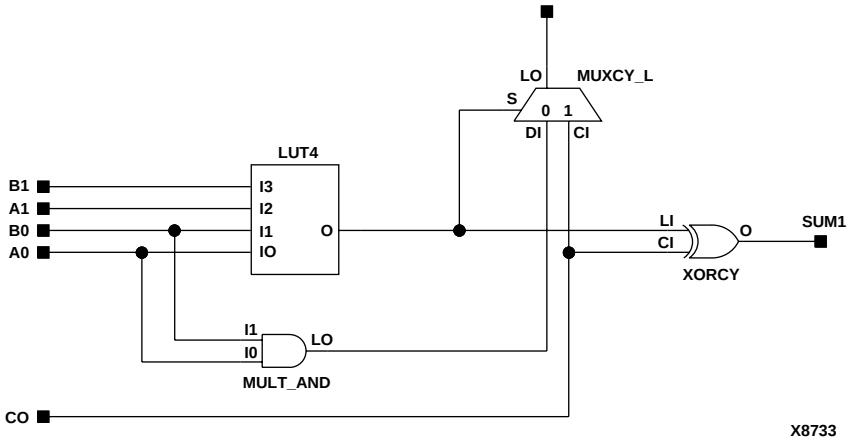
Fast Multiplier AND

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N / A	N / A	N / A



MULT_AND is an AND component used exclusively for building fast and smaller multipliers. The I1 and I0 inputs *must* be connected to the I1 and I0 inputs of the associated LUT. The LO output *must* be connected to the DI input of the associated MUXCY, MUXCY_D, or MUXCY_L.

Inputs		Output
I1	I0	LO
0	0	0
0	1	0
1	0	0
1	1	1



Example Multiplier Using MULT_AND

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for MULT_AND should be placed
-- after architecture statement but before begin keyword

component MULT_AND
   port (LO   : out STD_ULOGIC;
         I0   : in STD_ULOGIC;
         I1 : in STD_ULOGIC);
end component;

-- Component Attribute specification for MULT_AND
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MULT_AND should be placed
-- in architecture after the begin keyword

MULT_AND_INSTANCE_NAME : MULT_AND
      port map  (LO => user_LO,
                 I0 => user_I0,
                 I1 => user_I1);
```

Verilog Instantiation Template

```
MULT_AND MULT_AND_instance_name (.LO (user_LO),
                                 .I0 (user_I0),
                                 .I1 (user_I1));
```

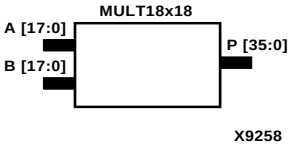
Commonly Used Constraints

U_SET, MULT_STYLE

MULT18X18

18 x 18 Signed Multiplier

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



MULT18X18 is a combinational signed 18-bit by 18-bit multiplier. The value represented in the 18-bit input A is multiplied by the value represented in the 18-bit input B. Output P is the 36-bit product of A and B.

A, B, and P are two's complement.

Inputs		Output
A	B	P
A	B	A * B

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MULT18X18 should be placed
-- after architecture statement but before begin keyword

component MULT18X18
   port (P  : out STD_LOGIC_VECTOR (35 downto 0);
         A  : in STD_LOGIC_VECTOR (17 downto 0);
         B  : in STD_LOGIC_VECTOR (17 downto 0));
end component;

-- Component Attribute specification for MULT18X18
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MULT18X18 should be placed
-- in architecture after the begin keyword

MULT18X18_INSTANCE_NAME : MULT18X18
      port map  (P => user_P,
                 A => user_A,
                 B => user_B);
```

Verilog Instantiation Template

```
MULT18X18 MULT18X18_instance_name (.P (user_P),
                                   .A (user_A),
                                   .B (user_B));
```

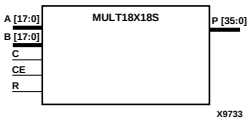
Commonly Used Constraints

MULT_STYLE

MULT18X18S

18 x 18 Signed Multiplier -- Registered Version

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



MULT18X18S is the registered version of the 18 x 18 signed multiplier with output P and inputs A, B, C, CE, and R. The registers are initialized to 0 after the GSR pulse.

The value represented in the 18-bit input A is multiplied by the value represented in the 18-bit input B. Output P is the 36-bit product of A and B.

A, B, and P are two's complement.

Inputs					Output
C	CE	Am	Bn	R	P
↑	X	X	X	1	0
↑	1	Am	Bn	0	A * B
X	0	X	X	0	No Chg

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for MULT18X18S should be placed
-- after architecture statement but before begin keyword

component MULT18X18S
   port (P  : out STD_LOGIC_VECTOR (35 downto 0);
         A  : in STD_LOGIC_VECTOR (17 downto 0);
         B  : in STD_LOGIC_VECTOR (17 downto 0);
         C  : in STD_ULOGIC;
         CE : in STD_ULOGIC;
         R  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MULT18X18S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MULT18X18S should be placed
-- in architecture after the begin keyword

MULT18X18S_INSTANCE_NAME : MULT18X18S
      port map  (P  => user_P,
```

```
A  => user_A,
B  => user_B,
CE => user_CE,
C  => user_C,
R  => user_R);
```

Verilog Instantiation Template

```
MULT18X18S MULT18X18S_instance_name (.P (user_P),
                                     .A (user_A),
                                     .B (user_B),
                                     .C (user_C),
                                     .CE (user_CE),
                                     .R (user_R));
```

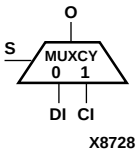
Commonly Used Constraints

HU_SET, MULT_STYLE

MUXCY

2-to-1 Multiplexer for Carry Logic with General Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXCY is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of:

- 2-bits per CLB for Virtex, Virtex-E, Spartan-II, and Spartan-IIE
- 4-bits per configurable logic block (CLB) for Virtex-II, and Virtex-II Pro

The direct input (DI) of an LC is connected to the DI input of the MUXCY. The carry in (CI) input of an LC is connected to the CI input of the MUXCY. The select input (S) of the MUXCY is driven by the output of the lookup table (LUT) and configured as a MUX function. The carry out (O) of the MUXCY reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

The variants, "MUXCY_D" and "MUXCY_L" provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	DI	CI	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXCY should be placed
-- after architecture statement but before begin keyword

component MUXCY
   port (O  : out STD_ULOGIC;
         CI : in STD_ULOGIC;
         DI : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXCY
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXCY should be placed
-- in architecture after the begin keyword

MUXCY_INSTANCE_NAME : MUXCY

      port map  (O  => user_O,
                 CI => user_CI,
                 DI => user_DI,
                 S  => user_S);
```

Verilog Instantiation Template

```
MUXCY MUXCY_instance_name (.O (user_O),
                           .CI (user_CI),
                           .DI (user_DI),
                           .S (user_S));
```

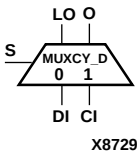
Commonly Used Constraints

U_SET

MUXCY_D

2-to-1 Multiplexer for Carry Logic with Dual Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXCY_D is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of 4-bits per configurable logic block (CLB). The direct input (DI) of an LC is connected to the DI input of the MUXCY_D. The carry in (CI) input of an LC is connected to the CI input of the MUXCY_D. The select input (S) of the MUX is driven by the output of the lookup table (LUT) and configured as an XOR function. The carry out (O and LO) of the MUXCY_D reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

Outputs O and LO are functionally identical. The O output is a general interconnect.

See also "MUXCY" and "MUXCY_L"

Inputs			Outputs	
S	DI	CI	O	LO
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXCY_D should be placed
-- after architecture statement but before begin keyword

component MUXCY_D
   port (LO : out STD_ULOGIC;
         O  : out STD_ULOGIC;
         CI : in STD_ULOGIC;
         DI : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXCY_D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXCY_D should be placed
-- in architecture after the begin keyword

MUXCY_D_INSTANCE_NAME : MUXCY_D
      port map  (LO => user_O,
                 O  => user_O,
                 CI => user_CI,
                 DI => user_DI,
                 S  => user_S);
```

Verilog Instantiation Template

```
MUXCY_D MUXCY_D_instance_name (.LO (user_LO),
                               .O (user_O),
                               .CI (user_CI),
                               .DI (user_DI),
                               .S (user_S));
```

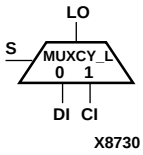
Commonly Used Constraints

U_SET

MUXCY_L

2-to-1 Multiplexer for Carry Logic with Local Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXCY_L is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of 4-bits per configurable logic block (CLB). The direct input (DI) of an LC is connected to the DI input of the MUXCY_L. The carry in (CI) input of an LC is connected to the CI input of the MUXCY_L. The select input (S) of the MUXCY_L is driven by the output of the lookup table (LUT) and configured as an XOR function. The carry out (LO) of the MUXCY_L reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

See also "MUXCY" and "MUXCY_D"

Inputs			Outputs
S	DI	CI	LO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXCY_L should be placed
-- after architecture statement but before begin keyword

component MUXCY_L
   port (LO : out STD_ULOGIC;
         CI : in STD_ULOGIC;
         DI : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXCY_L
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXCY_L should be placed
-- in architecture after the begin keyword

MUXCY_L_INSTANCE_NAME : MUXCY_L
      port map  (LO => user_O,
                 CI => user_CI,
                 DI => user_DI,
                 S  => user_S);
```

Verilog Instantiation Template

```
MUXCY_L MUXCY_L_instance_name (.LO (user_LO),
                               .CI (user_CI),
                               .DI (user_DI),
                               .S (user_S));
```

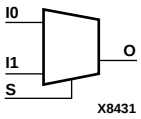
Commonly Used Constraints

U_SET

MUXF5

2-to-1 Lookup Table Multiplexer with General Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXF5 provides a multiplexer function in a CLB slice for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The variants, "MUXF5_D" and "MUXF5_L", provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	I0	I1	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF5 should be placed
-- after architecture statement but before begin keyword

component MUXF5
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF5
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF5 should be placed
-- in architecture after the begin keyword
```

```
MUXF5_INSTANCE_NAME : MUXF5
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

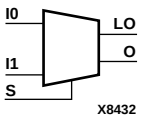
Verilog Instantiation Template

```
MUXF5 MUXF5_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .S (user_S));
```

MUXF5_D

2-to-1 Lookup Table Multiplexer with Dual Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXF5_D provides a multiplexer function in a CLB slice for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output is used to connect to other inputs within the same CLB slice.

See also "MUXF5" and "MUXF5_L"

Inputs			Outputs	
S	I0	I1	O	LO
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF5_D should be placed
-- after architecture statement but before begin keyword

component MUXF5_D
   port (LO : out STD_ULOGIC;
         O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF5_D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here
```

```
-- Component Instantiation for MUXF5_D should be placed
-- in architecture after the begin keyword

MUXF5_D_INSTANCE_NAME : MUXF5_D
      port map  (LO => user_LO,
                 O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

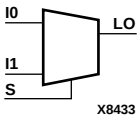
Verilog Instantiation Template

```
MUXF5_D MUXF5_D_instance_name (.LO (user_LO),
                               .O (user_O),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

MUXF5_L

2-to-1 Lookup Table Multiplexer with Local Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXF5_L provides a multiplexer function in a CLB slice for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output is used to connect to other inputs within the same CLB slice.

See also "MUXF5" and "MUXF5_D".

Inputs			Output
S	I0	I1	LO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF5_L should be placed
-- after architecture statement but before begin keyword

component MUXF5_L
   port (LO : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF5_L
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF5_L should be placed
-- in architecture after the begin keyword
```

```
MUXF5_L_INSTANCE_NAME : MUXF5_L
      port map  (LO => user_LO,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

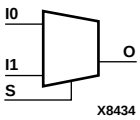
Verilog Instantiation Template

```
MUXF5_L MUXF5_L_instance_name (.LO (user_LO),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

MUXF6

2-to-1 Lookup Table Multiplexer with General Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXF6 provides a multiplexer function in a full Virtex, Virtex-E, Spartan-II, or Spartan-IIE CLB, or one half of a Virtex-II or Virtex-II Pro CLB (two slices) for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF6. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The variants, "MUXF6_D" and "MUXF6_L", provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	I0	I1	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF6 should be placed
-- after architecture statement but before begin keyword

component MUXF6
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF6
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF6 should be placed
```

```
-- in architecture after the begin keyword

MUXF6_INSTANCE_NAME : MUXF6
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

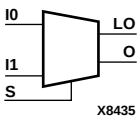
Verilog Instantiation Template

```
MUXF6 MUXF6_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .S (user_S));
```

MUXF6_D

2-to-1 Lookup Table Multiplexer with Dual Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXF6_D provides a multiplexer function in a full Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, or Spartan-IIE CLB, or one half of a Virtex-II or Virtex-II Pro CLB (two slices) for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF6. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output is used to connect to other inputs within the same CLB slice.

See also "MUXF6" and "MUXF6_L"

Inputs			Outputs	
S	I0	I1	O	LO
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF6_D should be placed
-- after architecture statement but before begin keyword

component MUXF6_D
   port (LO : out STD_ULOGIC;
         O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF6_D
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Attributes should be placed here

-- Component Instantiation for MUXF6_D should be placed
-- in architecture after the begin keyword

MUXF6_D_INSTANCE_NAME : MUXF6_D
      port map  (LO => user_LO,
                 O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

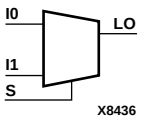
Verilog Instantiation Template

```
MUXF6_D MUXF6_D_instance_name (.LO (user_LO),
                               .O (user_O),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

MUXF6_L

2-to-1 Lookup Table Multiplexer with Local Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



MUXF6_L provides a multiplexer function in a full Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, or Spartan-IIE CLB, or one half of a Virtex-II or Virtex-II Pro CLB (two slices) for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF6. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output is used to connect to other inputs within the same CLB slice.

See also "MUXF6" and "MUXF6_D".

Inputs			Output
S	I0	I1	LO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF6_L should be placed
-- after architecture statement but before begin keyword

component MUXF6_L
   port (LO : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF6_L
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here
```

```
-- Component Instantiation for MUXF6_L should be placed
-- in architecture after the begin keyword

MUXF6_L_INSTANCE_NAME : MUXF6_L
      port map  (LO => user_LO,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

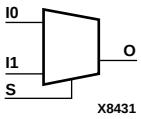
Verilog Instantiation Template

```
MUXF6_L MUXF6_L_instance_name (.LO (user_LO),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

MUXF7

2-to-1 Lookup Table Multiplexer with General Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



MUXF7 provides a multiplexer function in a full Virtex-II and Virtex-II Pro CLB for creating a function-of-7 lookup table or a 16-to-1 multiplexer in combination with the associated lookup tables. Local outputs (LO) of MUXF6 are connected to the I0 and I1 inputs of the MUXF7. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The variants, "MUXF7_D" and "MUXF7_L", provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	I0	I1	O
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF7 should be placed
-- after architecture statement but before begin keyword

component MUXF7
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF7
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF7 should be placed
-- in architecture after the begin keyword
```

```
MUXF7_INSTANCE_NAME : MUXF7
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

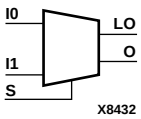
Verilog Instantiation Template

```
MUXF7 MUXF7_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .S (user_S));
```

MUXF7_D

2-to-1 Lookup Table Multiplexer with Dual Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



MUXF7_D provides a multiplexer function in one full Virtex-II and Virtex-II Pro CLB for creating a function-of-7 lookup table or a 16-to-1 multiplexer in combination with the associated lookup tables. Local outputs (LO) of MUXF6 are connected to the I0 and I1 inputs of the MUXF7. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output is used to connect to other inputs within the same CLB slice.

See also "MUXF7" and "MUXF7_L".

Inputs			Outputs	
S	I0	I1	O	LO
0	I0	X	I0	I0
1	X	I1	I1	I1
X	0	0	0	0
X	1	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF7_D should be placed
-- after architecture statement but before begin keyword

component MUXF7_D
   port (LO : out STD_ULOGIC;
         O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF7_D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here
```

```
-- Component Instantiation for MUXF7_D should be placed
-- in architecture after the begin keyword

MUXF7_D_INSTANCE_NAME : MUXF7_D
      port map  (LO => user_LO,
                 O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

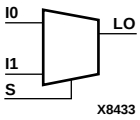
Verilog Instantiation Template

```
MUXF7_D MUXF7_D_instance_name (.LO (user_LO),
                               .O (user_O),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

MUXF7_L

2-to-1 Lookup Table Multiplexer with Local Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



MUXF7_L provides a multiplexer function in a full Virtex-II and Virtex-II Pro CLB for creating a function-of-7 lookup table or a 16-to-1 multiplexer in combination with the associated lookup tables. Local outputs (LO) of MUXF6 are connected to the I0 and I1 inputs of the MUXF7. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output is used to connect to other inputs within the same CLB slice.

See also "MUXF7" and "MUXF7_D".

Inputs			Output
S	I0	I1	LO
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF7_L should be placed
-- after architecture statement but before begin keyword

component MUXF7_L
   port (LO : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF7_L
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF7_L should be placed
-- in architecture after the begin keyword
```

```
MUXF7_L_INSTANCE_NAME : MUXF7_L
      port map  (LO => user_LO,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

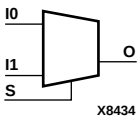
Verilog Instantiation Template

```
MUXF7_L MUXF7_L_instance_name (.LO (user_LO),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

MUXF8

2-to-1 Lookup Table Multiplexer with General Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



MUXF8 provides a multiplexer function in two full Virtex-II and Virtex-II Pro CLBs for creating a function-of-8 lookup table or a 32-to-1 multiplexer in combination with the associated lookup tables, MUXF5s, MUXF6s, and MUXF7s. Local outputs (LO) of MUXF7 are connected to the I0 and I1 inputs of the MUXF8. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The variants, "MUXF8_D" and "MUXF8_L", provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	I0	I1	O
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF8 should be placed
-- after architecture statement but before begin keyword

component MUXF8
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF8
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF8 should be placed
-- in architecture after the begin keyword
```

```
MUXF8_INSTANCE_NAME : MUXF8
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

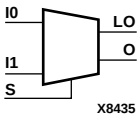
Verilog Instantiation Template

```
MUXF8 MUXF8_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .S (user_S));
```

MUXF8_D

2-to-1 Lookup Table Multiplexer with Dual Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



MUXF8_D provides a multiplexer function in two full Virtex-II and Virtex-II Pro CLBs for creating a function-of-8 lookup table or a 32-to-1 multiplexer in combination with the associated four lookup tables and two MUXF8s. Local outputs (LO) of MUXF7 are connected to the I0 and I1 inputs of the MUXF8. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output is used to connect to other inputs within the same CLB slice.

See also "MUXF8" and "MUXF8_L".

Inputs			Outputs	
S	I0	I1	O	LO
0	I0	X	I0	I0
1	X	I1	I1	I1
X	0	0	0	0
X	1	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF8_D should be placed
-- after architecture statement but before begin keyword

component MUXF8_D
   port (LO : out STD_ULOGIC;
         O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF8_D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here
```

```
-- Component Instantiation for MUXF8_D should be placed
-- in architecture after the begin keyword

MUXF8_D_INSTANCE_NAME : MUXF8_D
      port map  (LO => user_LO,
                 O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

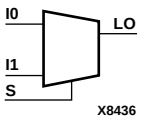
Verilog Instantiation Template

```
MUXF8_D MUXF8_D_instance_name (.LO (user_LO),
                               .O (user_O),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

MUXF8_L

2-to-1 Lookup Table Multiplexer with Local Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



MUXF8_L provides a multiplexer function in two full Virtex-II and Virtex-II Pro CLBs for creating a function-of-8 lookup table or a 32-to-1 multiplexer in combination with the associated four lookup tables and two MUXF8s. Local outputs (LO) of MUXF7 are connected to the I0 and I1 inputs of the MUXF8. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output is used to connect to other inputs within the same CLB slice.

See also "MUXF8" and "MUXF8_D".

Inputs			Output
S	I0	I1	LO
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF8_L should be placed
-- after architecture statement but before begin keyword

component MUXF8_L
   port (LO : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXF8_L
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF8_L should be placed
-- in architecture after the begin keyword
```

```
MUXF8_L_INSTANCE_NAME : MUXF8_L

      port map  (LO => user_LO,
                 I0 => user_I0,
                 I1 => user_I1,
                 S  => user_S);
```

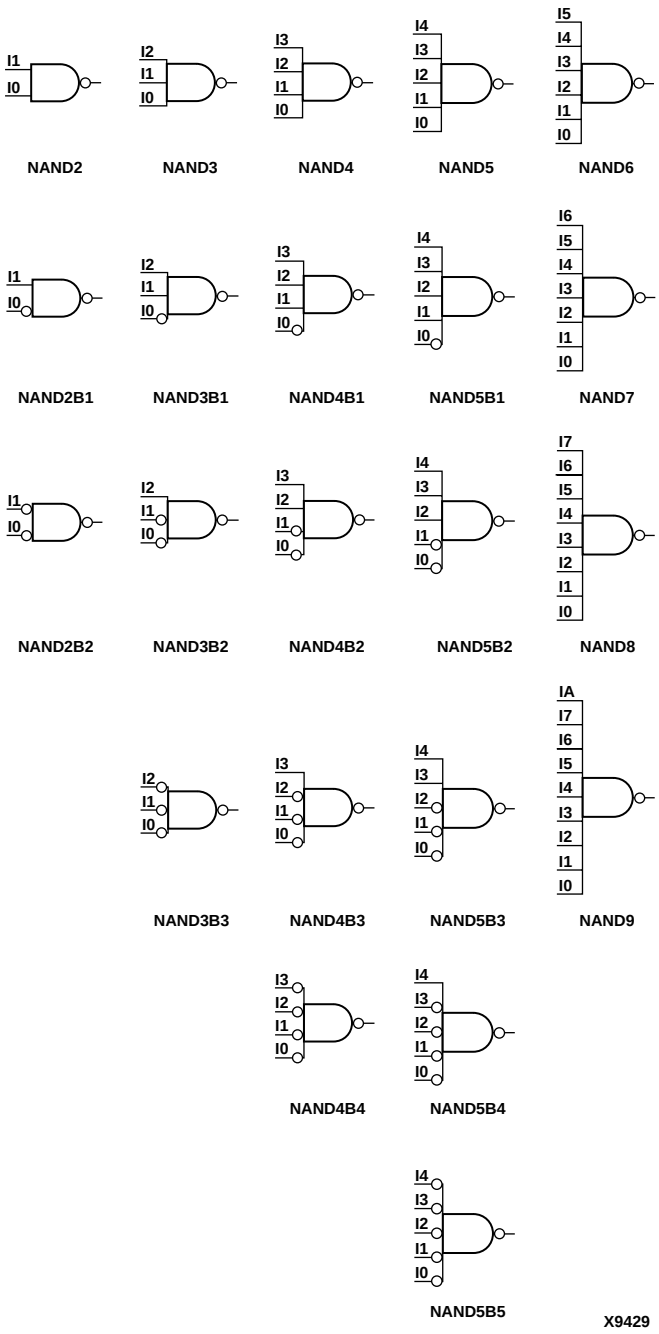
Verilog Instantiation Template

```
MUXF8_L MUXF8_L_instance_name (.LO (user_LO),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

NAND2-9

2- to 9-Input NAND Gates with Inverted and Non-Inverted Inputs

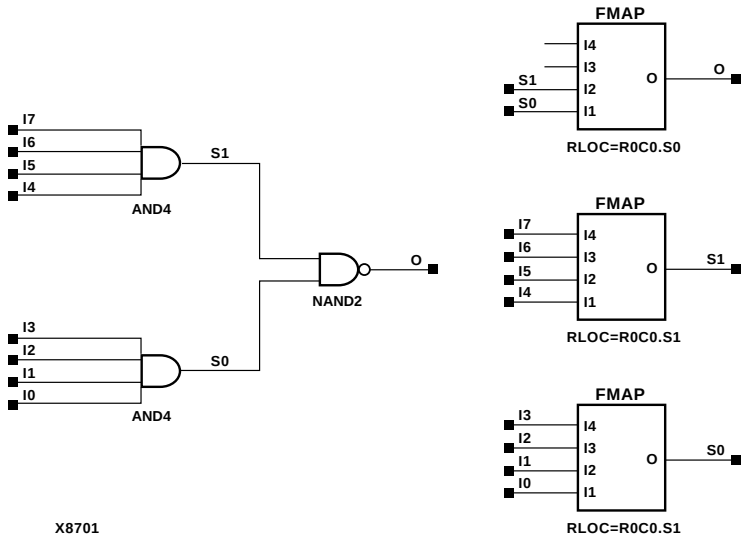
Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
NAND2, NAND2B1, NAND2B2, NAND3, NAND3B1, NAND3B2, NAND3B3, NAND4, NAND4B1, NAND4B2, NAND4B3, NAND4B4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND5, NAND5B1, NAND5B2, NAND5B3, NAND5B4, NAND5B5	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND6, NAND7, NAND8, NAND9	Macro	Macro	Macro	Primitive	Primitive	Primitive



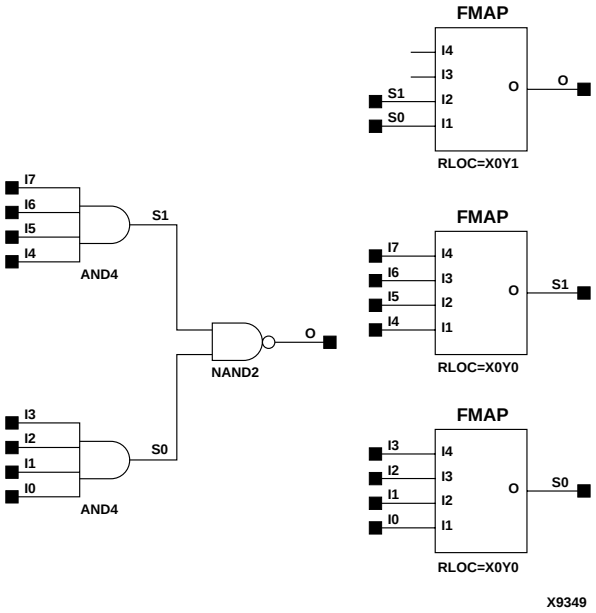
NAND Gate Representations

NAND gates of up to five inputs are available in any combination of inverting and non-inverting inputs. NAND gates of six to nine inputs are available with only non-inverting inputs. To invert inputs, use external inverters. Since each input uses a CLB resource, replace gates with unused inputs with gates having the necessary number of inputs.

See the "NAND12, 16" section for information on additional NAND functions.



NAND8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NAND8 Implementation Virtex-II, Virtex-II Pro

Usage

NAND2 through NAND5 are primitives that can be inferred or instantiated. NAND6 through NAND9 are macros which can be inferred.

VHDL Inference Code

```
architecture Behavioral of nand6 is

begin

process (I0, I1, I2, I3, I4, I5)
begin
       O <= not (I0 and I1 and I2 and I3 and I4 and I5);
end process;

end Behavioral;
```

Verilog Inference Code

Following is the a Verilog example of how to instantiate a NAND6 element:

```
always @(I0 or I1 or I2 or I3 or I4 or I5)
begin
       O <= !(I0 && I1 && I2 && I3 && I4 && I5);
end
```

VHDL Instantiation Template for NAND5

Following is the VHDL code for NAND5. To instantiate NAND2, remove I2, I3, and I4. To instantiate NAND3, remove I3 and I4, For NAND4, remove I4. NAND2B1, and NAND2B2 have the same code as NAND2. NAND3B1, 3B2, and 3B3 have the same code as NAND3 and so forth.

```
-- Component Declaration for NAND5 should be placed
-- after architecture statement but before begin keyword

component NAND5
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         I2 : in STD_ULOGIC;
         I3 : in STD_ULOGIC;
         I4 : in STD_ULOGIC);
end component;

-- Component Attribute specification for NAND5
-- should be placed after architecture declaration but
-- before the begin keyword
-- Attributes should be placed here

-- Component Instantiation for NAND5 should be placed
-- in architecture after the begin keyword

NAND5_INSTANCE_NAME : NAND5
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3,
                 I4 => user_I4);
```

Verilog Instantiation Template for NAND5

```
NAND5 NAND5_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .I2 (user_I2),
                           .I3 (user_I3),
                           .I4 (user_I4));
```

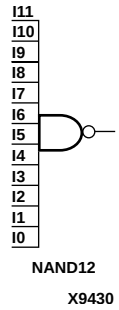
Commonly Used Constraints

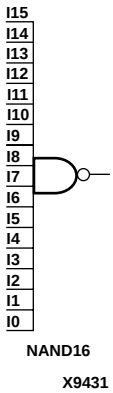
None

NAND12, 16

12- and 16-Input NAND Gates with Non-Inverted Inputs

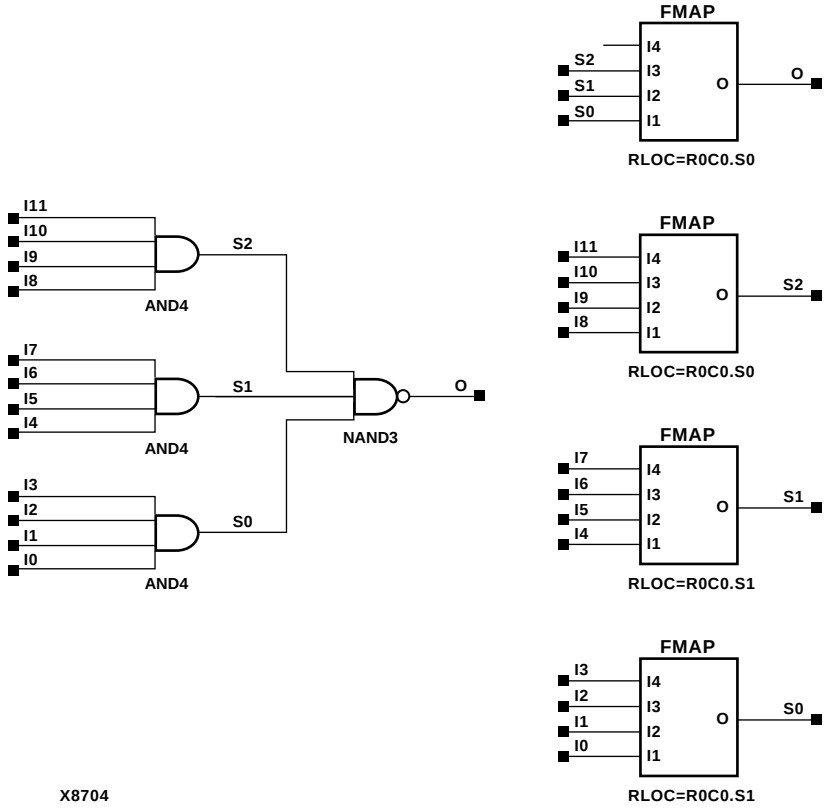
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



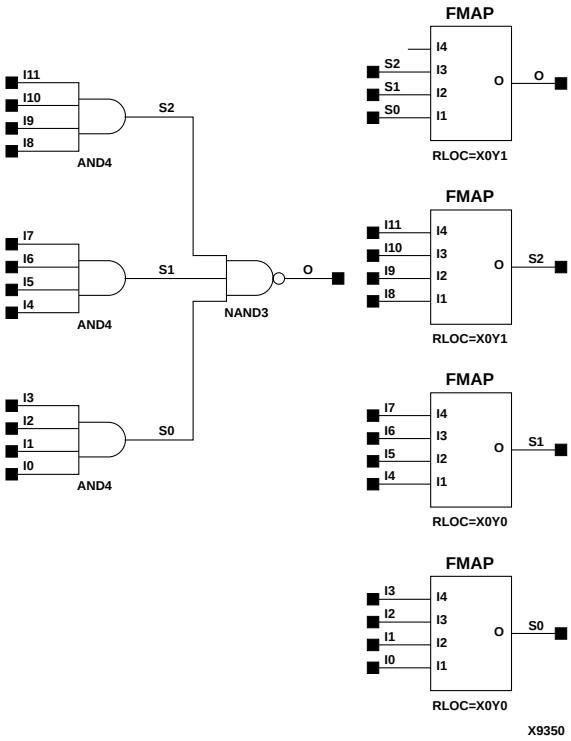


The NAND function is performed in the Configurable Logic Block (CLB) function generators for Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro. The 12- and 16-input NAND functions are available only with non-inverting inputs. To invert some or all inputs, use external inverters.

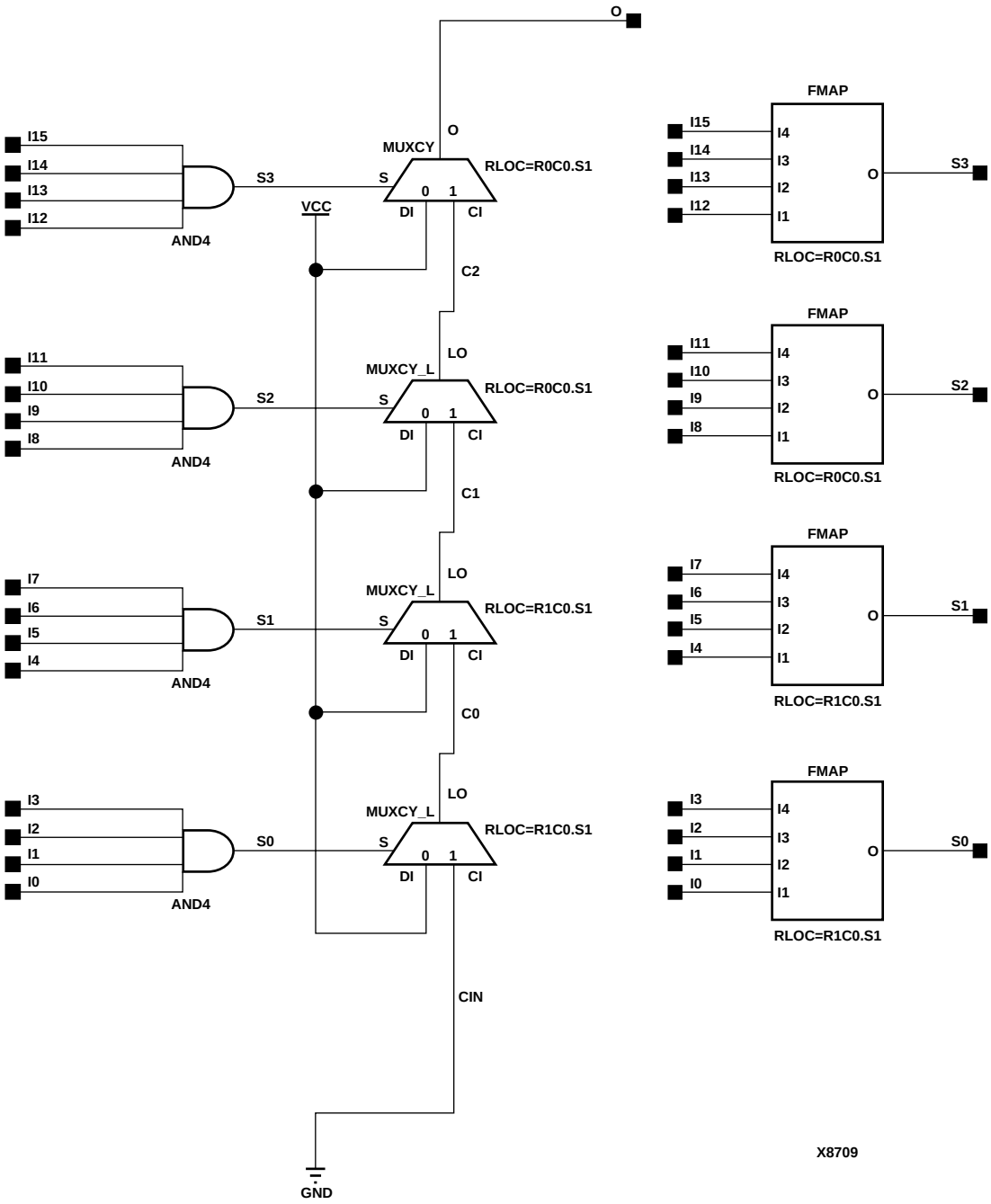
See NAND2-9 for more information on NAND functions.



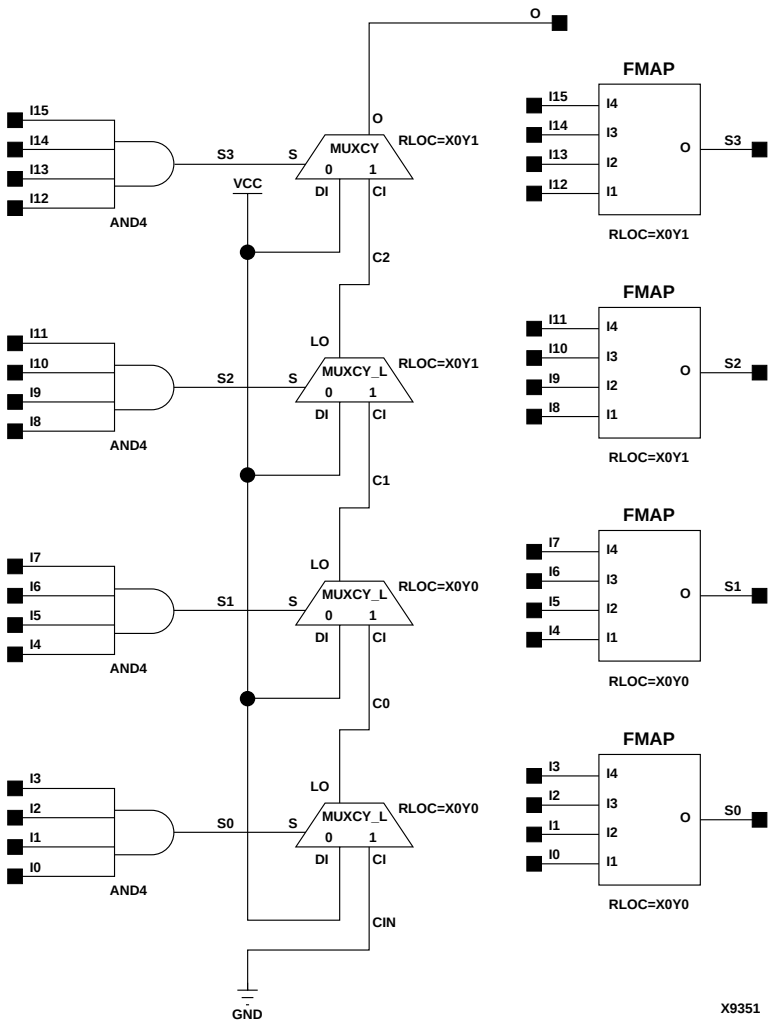
NAND12 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NAND12 Implementation Virtex-II, Virtex-II Pro



NAND16 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NAND16 Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, NAND12 and NAND16 are macros that are inferred. See NAND2-9 for more information about inferring NAND gates.

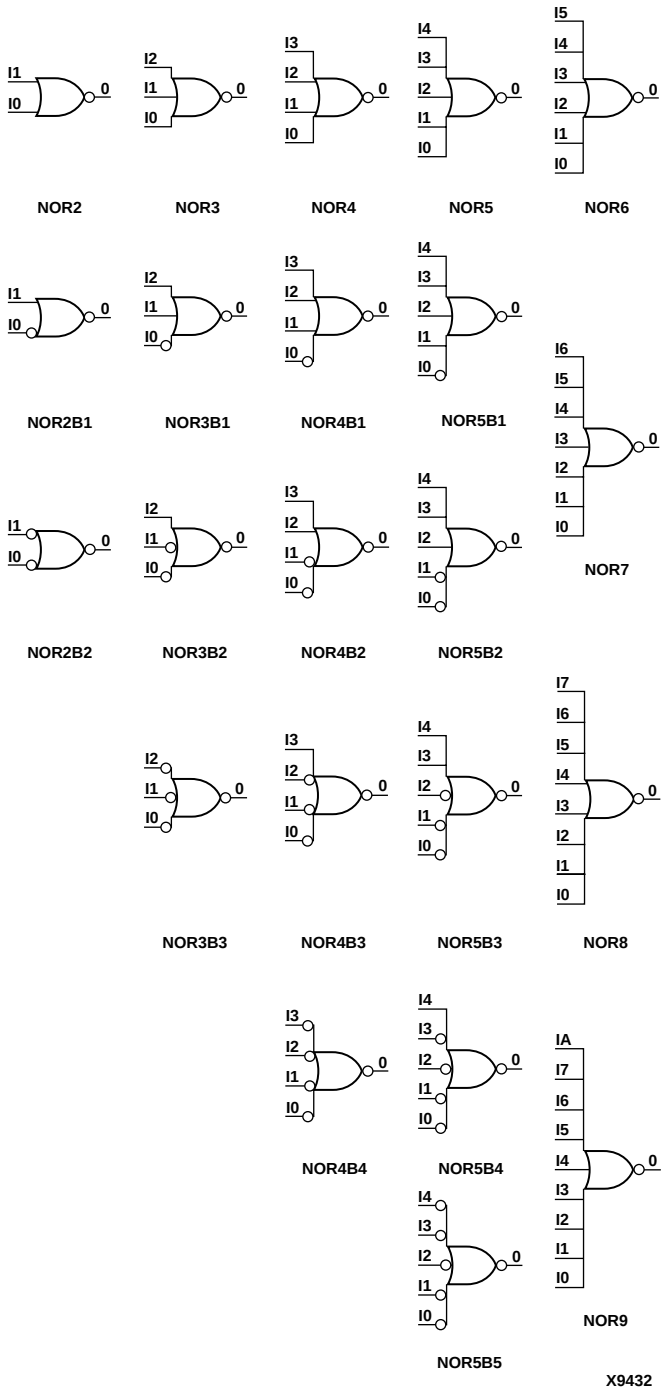
Commonly Used Constraints

None

NOR2-9

2- to 9-Input NOR Gates with Inverted and Non-Inverted Inputs

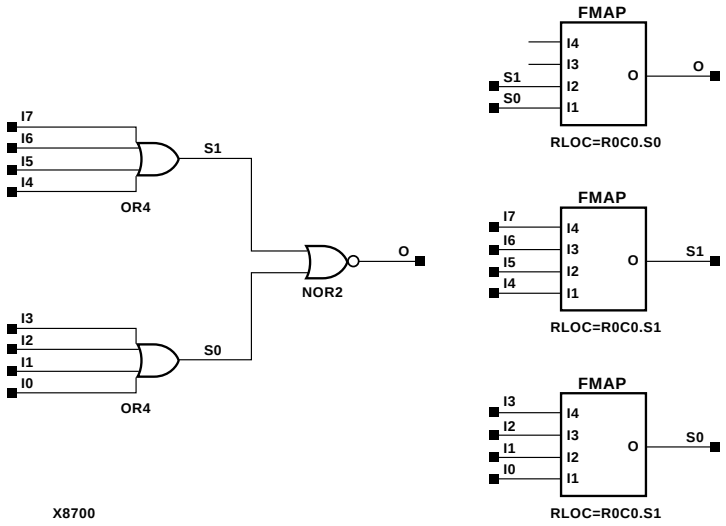
Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
NOR2, NOR2B1, NOR2B2, NOR3, NOR3B1, NOR3B2, NOR3B3, NOR4, NOR4B1, NOR4B2, NOR4B3, NOR4B4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR5, NOR5B1, NOR5B2, NOR5B3, NOR5B4, NOR5B5	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR6, NOR7, NOR8, NOR9	Macro	Macro	Macro	Primitive	Primitive	Primitive



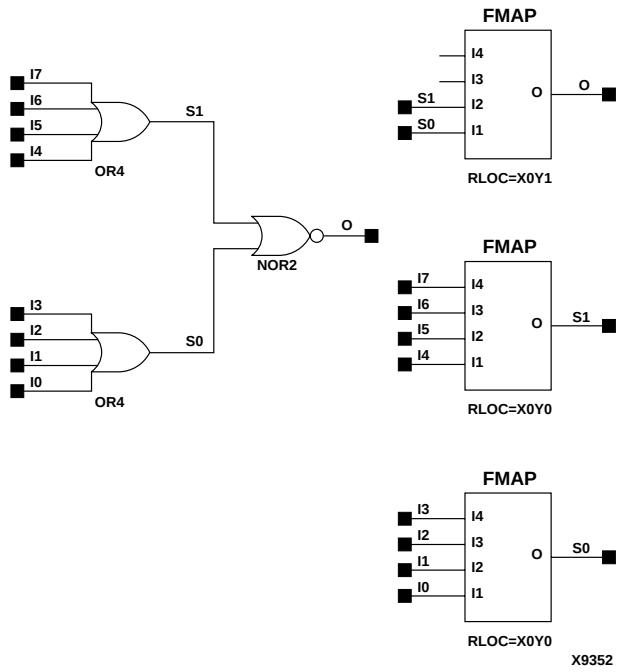
NOR Gate Representations

NOR gates of up to five inputs are available in any combination of inverting and non-inverting inputs. NOR gates of six to nine inputs are available with only non-inverting inputs. To invert some or all inputs, use external inverters. Since each input uses a CLB resource, replace gates with unused inputs with gates having the necessary number of inputs.

See the "NOR12, 16" section for information on additional NOR functions.



NOR8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NOR8 Implementation Virtex-II, Virtex-II Pro

Usage

NOR2 through NOR5 are primitives that can be inferred or instantiated. NOR6 through NOR9 are macros which can be inferred.

VHDL Inference Code

```
architecture Behavioral of nor6 is

begin

process (I0, I1, I2, I3, I4, I5)
begin
       O <= not (I0 or I1 or I2 or I3 or I4 or I5);
end process;

end Behavioral;
```

Verilog Inference Code (NOR6)

```
always @(I0 or I1 or I2 or I3 or I4 or I5)
begin
       O <= !(I0 || I1 || I2 || I3 || I4 || I5);
end
```

VHDL Instantiation Template for NOR5

Following is the VHDL code for NOR5. To instantiate NOR2, remove I2, I3, and I4. To instantiate NOR3, remove I3 and I4, For NOR4, remove I4. NOR2B1, and NOR2B2 have the same code as NOR2. NOR3B1, 3B2, and 3B3 have the same code as NOR3 and so forth.

```
-- Component Declaration for NOR5 should be placed
-- after architecture statement but before begin keyword

component NOR5
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         I2 : in STD_ULOGIC;
         I3 : in STD_ULOGIC;
         I4 : in STD_ULOGIC);
end component;

-- Component Attribute specification for NOR5
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for NOR5 should be placed
-- in architecture after the begin keyword

NOR5_INSTANCE_NAME : NOR5
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3,
                 I4 => user_I4);
```

Verilog Instantiation Template for NOR5

```
NOR5 NOR5_instance_name (.O (user_O),
                         .I0 (user_I0),
                         .I1 (user_I1),
                         .I2 (user_I2),
                         .I3 (user_I3),
                         .I4 (user_I4));
```

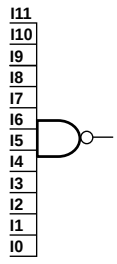
Commonly Used Constraints

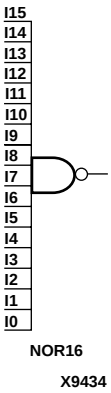
None

NOR12, 16

12- and 16-Input NOR Gates with Non-Inverted Inputs

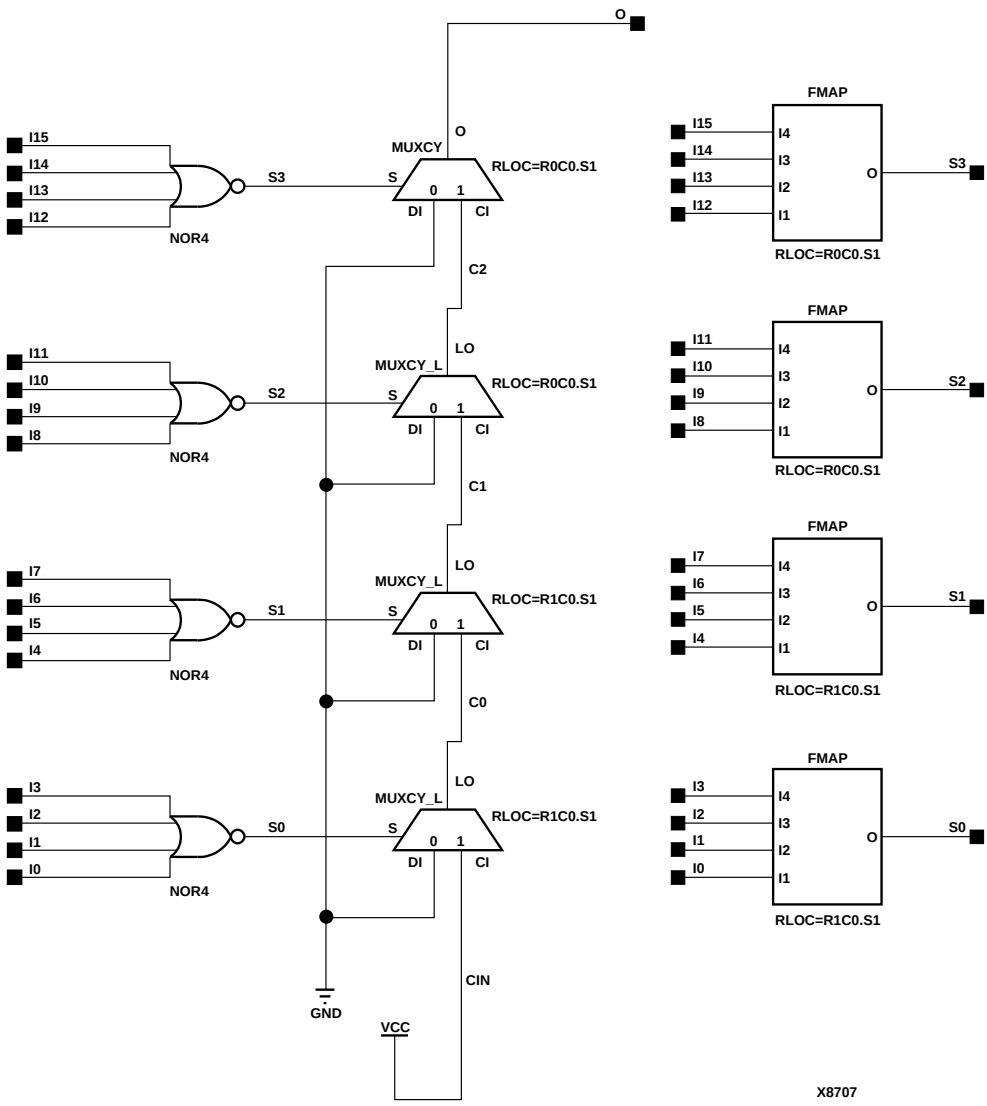
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



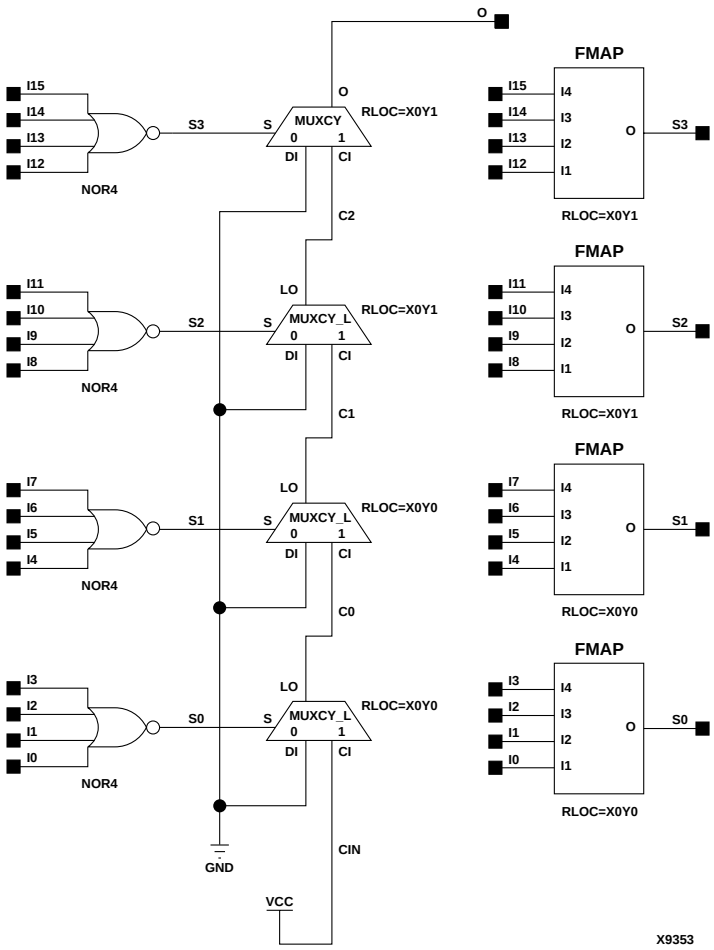


The 12- and 16-input NOR functions are available only with non-inverting inputs. To invert some or all inputs, use external inverters.

See the "NOR2-9" section for more information on NOR functions.



NOR16 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NOR16 Implementation Virtex-II, Virtex-II Pro

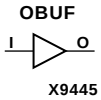
Usage

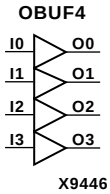
For HDL, NOR12 and NOR16 are macros that can be inferred. See the "NOR2-9" section for more information about inferring NOR gates.

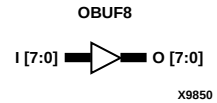
OBUF, 4, 8, 16

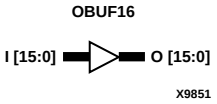
Single- and Multiple-Output Buffers

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OBUF	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OBUF4, OBUF8, OBUF16	Macro	Macro	Macro	Macro	Macro	Macro





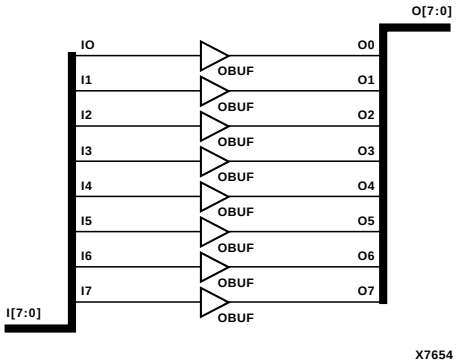




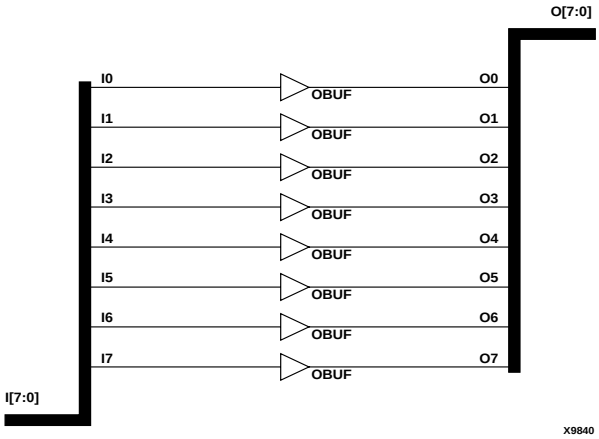
OBUF, OBUF4, OBUF8, and OBUF16 are single and multiple output buffers. An OBUF isolates the internal circuit and provides drive current for signals leaving a chip. OBUFs exist in input/output blocks (IOB). The output (O) of an OBUF is connected to an OPAD or an IOPAD.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, if a high impedance (Z) signal from an on-chip 3-state buffer (like BUFE) is applied to the input of an OBUF, it is propagated to the CPLD device output pin.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, see the OBUF_*selectIO* section for information on OBUF variants with selectable I/O interfaces. The I/O interface standard used by OBUF, 4, 8, and 16 is LVTTL. Also, Virtex, Virtex-E, Spartan-II, and Spartan-IIE OBUF, 4, 8, and 16 have selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.



OBUF8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E



OBUF8 Implementation Virtex-II, Virtex-II Pro

Usage

OBUFs are typically inferred for all top level input ports, but they can also be instantiated if necessary.

VHDL Instantiation Template

```
-- Component Declaration for OBUF should be placed
-- after architecture statement but before begin keyword

component OBUF
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OBUF
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OBUF should be placed
-- in architecture after the begin keyword

OBUF_INSTANCE_NAME : OBUF
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
OBUF_instance_name (.O (user_O),
                    .I (user_I));
```

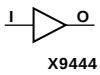
Commonly Used Constraints

DRIVE, IOSTANDARD, IOBDELAY, SLEW

OBUF_*selectIO*

Single Output Buffer with Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



X9444

For Virtex, Virtex-E, Virtex-II, and Virtex-II Pro, Spartan-II, and Spartan-IIE, OBUF and its selectIO variants (listed in the "Components" column in the table below) are single output buffers whose I/O interface corresponds to a specific I/O standard. The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. The S, F, and 2, 4, 6, 8, 12, 16, 24 extensions specify the slew rate (SLOW or FAST) and the drive power (2, 4, 6, 8, 12, 16, 24 mA) for the LVTTL standard variants. For example, OBUF_F_12 is a single output buffer that uses the LVTTL I/O-signaling standard with a FAST slew and 12mA of drive power. You can attach an IOSTANDARD attribute to an OBUF instance instead of using an OBUF_*selectIO* component. Check marks (√) in the "Spartan-II, Virtex" and "Spartan-IIE, Virtex-E" columns indicate the components and IOSTANDARD attribute values available for those architectures.

For Virtex-II and Virtex-II Pro, an OBUF that uses the LVTTL, LVCMOS15, LVCMOS18, LVCMOS25, or LVCMOS33 signaling standards has selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.

For Virtex, Virtex-E, Spartan-II, and Spartan-IIE, an OBUF that uses the LVTTL signaling standard has selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.

An OBUF isolates the internal circuit and provides drive current for signals leaving a chip. OBUFs exist in input/output blocks (IOB). The output (O) of an OBUF is connected to an OPAD or an IOPAD.

The hardware implementation of the I/O standard requires that you follow a set of usage rules for the SelectI/O buffer components. See the "SelectI/O Usage Rules" under the IBUF_*selectIO* section for information on using these components.

Spartan-II, Spartan-IIE, Virtex, and Virtex-E OBUF_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Output VCCO
OBUF	√	√	(default is LVTTL) [b]	3.3
OBUF_S_2	√	√	LVTTL [b]	3.3
OBUF_S_4	√	√	LVTTL [b]	3.3
OBUF_S_6	√	√	LVTTL [b]	3.3
OBUF_S_8	√	√	LVTTL [b]	3.3
OBUF_S_12	√	√	LVTTL [b]	3.3
OBUF_S_16	√	√	LVTTL [b]	3.3
OBUF_S_24	√	√	LVTTL [b]	3.3

Spartan-II, Spartan-IIE, Virtex, and Virtex-E OBUF_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Output VCCO
OBUF_F_2	√	√	LVTTL [b]	3.3
OBUF_F_4	√	√	LVTTL [b]	3.3
OBUF_F_6	√	√	LVTTL [b]	3.3
OBUF_F_8	√	√	LVTTL [b]	3.3
OBUF_F_12	√	√	LVTTL [b]	3.3
OBUF_F_16	√	√	LVTTL [b]	3.3
OBUF_F_24	√	√	LVTTL [b]	3.3
OBUF_AGP	√	√	AGP	3.3
OBUF_CTT	√	√	CTT	3.3
OBUF_GTL	√	√	GTL	N / A
OBUF_GTLP	√	√	GTLP	N / A
OBUF_HSTL_I	√	√	HSTL_I	1.5
OBUF_HSTL_III	√	√	HSTL_III	1.5
OBUF_HSTL_IV	√	√	HSTL_IV	1.5
OBUF_LVCMOS2	√	√	LVCMOS2	2.5
OBUF_LVCMOS18		√	LVCMOS18	1.8
OBUF_LVDS		√	LVDS	2.5
OBUF_LVPECL		√	LVPECL	3.3
OBUF_PCI33_3	√	√	PCI33_3	3.3
OBUF_PCI33_5	√	√	PCI33_5	3.3
OBUF_PCI66_3	√	√	PCI66_3	3.3
OBUF_PCIx66_3		√	PCIx66_3	3.3
OBUF_SSTL2_I	√	√	SSTL2_I	2.5
OBUF_SSTL2_II	√	√	SSTL2_II	2.5
OBUF_SSTL3_I	√	√	SSTL3_I	3.3
OBUF_SSTL3_II	√	√	SSTL3_II	3.3

[b] The LVTTL attribute also requires a slew value (FAST or SLOW) and DRIVE value. See the FAST, SLOW, and DRIVE attribute descriptions in the *Xilinx Constraints Guide* for valid values for each architecture.

The Virtex-II and Virtex-II Pro library includes some OBUF_*selectIO* components for compatibility with older, existing designs and other architectures. For new Virtex-II and Virtex-II Pro designs, however, the recommended method for using OBUF selectI/O buffers is to attach an IOSTANDARD attribute to an OBUF component. For example, attach IOSTANDARD=GTLP to an OBUF instead of using the OBUF_GTLP component for new Virtex-II and Virtex-II Pro designs. The IOSTANDARD attributes that can be attached to an OBUF component are listed in the "IOSTANDARD (Attribute Value)" column.

Virtex-II, Virtex-II Pro OBUF_*selectIO* IOSTANDARD Attributes

			Attribute Values				
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Terminate Type**
OBUF[a]	√		AGP	N / A	N / A	3.3	None
OBUF[a]	√	√	GTL	N / A	N / A	N / A	None
OBUF[a]	√	√	GTL_DCI	N / A	N / A	1.2	Single
OBUF[a]	√	√	GTLP	N / A	N / A	N / A	None
OBUF[a]	√	√	GTLP_DCI	N / A	N / A	1.5	Single
OBUF[a]	√	√	HSTL_I	N / A	N / A	1.5	None
OBUF[a]	√	√	HSTL_I_18	N / A	N / A	1.8	None
OBUF[a]	√	√	HSTL_I_DCI	N / A	N / A	1.5	Split
OBUF[a]	√	√	HSTL_I_DCI_18	N / A	N / A	1.5	Split
OBUF[a]	√	√	HSTL_II	N / A	N / A	1.5	None
OBUF[a]	√	√	HSTL_II_18	N / A	N / A	1.5	None
OBUF[a]	√	√	HSTL_II_DCI	N / A	N / A	1.5	Split
OBUF[a]	√	√	HSTL_II_DCI_18	N / A	N / A	1.8	Split
OBUF[a]	√	√	HSTL_III	N / A	N / A	1.5	None
OBUF[a]	√	√	HSTL_III_18	N / A	N / A	1.8	None
OBUF[a]	√	√	HSTL_III_DCI	N / A	N / A	1.5	Split
OBUF[a]	√	√	HSTL_III_DCI_18	N / A	N / A	1.8	Split
OBUF[a]	√	√	HSTL_IV	N / A	N / A	1.5	None
OBUF[a]	√	√	HSTL_IV_18	N / A	N / A	1.8	None
OBUF[a]	√	√	HSTL_IV_DCI	N / A	N / A	1.5	Split
OBUF[a]	√	√	HSTL_IV_DCI_18	N / A	N / A	1.8	Split
OBUF[a]	√	√	LVCMOS15[b]	N / A	N / A	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	12	F	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	16	F	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	2	F	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	4	F	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	6	F	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	8	F	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	12	S	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	16	S	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	2	S	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	4	S	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	6	S	1.5	None
OBUF[a]	√	√	LVCMOS15[b]	8	S	1.5	None
OBUF[a]	√	√	LVCMOS18[b]	N / A	N / A	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	12	F	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	16	F	1.8	None

Virtex-II, Virtex-II Pro OBUF_*selectIO* IOSTANDARD Attributes

			Attribute Values				
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Terminate Type**
OBUF[a]	√	√	LVCMOS18[b]	2	F	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	4	F	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	6	F	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	8	F	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	12	S	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	16	S	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	2	S	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	4	S	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	6	S	1.8	None
OBUF[a]	√	√	LVCMOS18[b]	8	S	1.8	None
OBUF[a]	√	√	LVCMOS2[b]	N / A	N / A	1.8	None
OBUF[a]	√	√	LVCMOS25[b]	N / A	N / A	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	12	F	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	16	F	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	2	F	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	24	F	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	4	F	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	6	F	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	8	F	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	12	S	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	16	S	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	2	S	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	24	S	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	4	S	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	6	S	2.5	None
OBUF[a]	√	√	LVCMOS25[b]	8	S	2.5	None
OBUF[a]	√		LVCMOS33[b]	N / A	N / A	3.3	None
OBUF[a]	√		LVCMOS33[b]	12	F	3.3	None
OBUF[a]	√		LVCMOS33[b]	16	F	3.3	None
OBUF[a]	√		LVCMOS33[b]	2	F	3.3	None
OBUF[a]	√		LVCMOS33[b]	24	F	3.3	None
OBUF[a]	√		LVCMOS33[b]	4	F	3.3	None
OBUF[a]	√		LVCMOS33[b]	6	F	3.3	None
OBUF[a]	√		LVCMOS33[b]	8	F	3.3	None
OBUF[a]	√		LVCMOS33[b]	12	S	3.3	None
OBUF[a]	√		LVCMOS33[b]	16	S	3.3	None
OBUF[a]	√		LVCMOS33[b]	2	S	3.3	None
OBUF[a]	√		LVCMOS33[b]	24	S	3.3	None

Virtex-II, Virtex-II Pro OBUF_*selectIO* IOSTANDARD Attributes

			Attribute Values				
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Terminate Type**
OBUF[a]	√		LVCMOS33[b]	4	S	3.3	None
OBUF[a]	√		LVCMOS33[b]	6	S	3.3	None
OBUF[a]	√		LVCMOS33[b]	8	S	3.3	None
OBUF[a]	√	√	LVDCI_15	N / A	N / A	1.5	Driver
OBUF[a]	√	√	LVDCI_18	N / A	N / A	1.8	Driver
OBUF[a]	√	√	LVDCI_25	N / A	N / A	2.5	Driver
OBUF[a]	√	√	LVDCI_33	N / A	N / A	3.3	Driver
OBUF[a]	√	√	LVDCI_DV2_15	N / A	N / A	1.5	Driver
OBUF[a]	√	√	LVDCI_DV2_18	N / A	N / A	1.8	Driver
OBUF[a]	√	√	LVDCI_DV2_25	N / A	N / A	2.5	Driver
OBUF[a]	√		LVDCI_DV2_33	N / A	N / A	3.3	Driver
OBUF[a]	√	√	LVTTL[b]	N / A	N / A	3.3	None
OBUF[a]	√	√	LVTTL[b]	12	F	3.3	None
OBUF[a]	√	√	LVTTL[b]	16	F	3.3	None
OBUF[a]	√	√	LVTTL[b]	2	F	3.3	None
OBUF[a]	√	√	LVTTL[b]	24	F	3.3	None
OBUF[a]	√	√	LVTTL[b]	4	F	3.3	None
OBUF[a]	√	√	LVTTL[b]	6	F	3.3	None
OBUF[a]	√	√	LVTTL[b]	8	F	3.3	None
OBUF[a]	√	√	LVTTL[b]	12	S	3.3	None
OBUF[a]	√	√	LVTTL[b]	16	S	3.3	None
OBUF[a]	√	√	LVTTL[b]	2	S	3.3	None
OBUF[a]	√	√	LVTTL[b]	24	S	3.3	None
OBUF[a]	√	√	LVTTL[b]	4	S	3.3	None
OBUF[a]	√	√	LVTTL[b]	6	S	3.3	None
OBUF[a]	√	√	LVTTL[b]	8	S	3.3	None
OBUF[a]	√	√	PCI33_3	N / A	N / A	3.3	None
OBUF[a]	√	√	PCI66_3	N / A	N / A	3.3	None
OBUF[a]	√		PCIX	N / A	N / A	3.3	None
OBUF[a]	√	√	SSTL18_I	N / A	N / A	1.8	None
OBUF[a]	√	√	SSTL18_I_DCI	N / A	N / A	1.8	Split
OBUF[a]	√	√	SSTL18_II	N / A	N / A	1.8	None
OBUF[a]	√	√	SSTL18_II_DCI	N / A	N / A	1.8	Split
OBUF[a]	√	√	SSTL2_I	N / A	N / A	2.5	None
OBUF[a]	√	√	SSTL2_I_DCI	N / A	N / A	2.5	Split
OBUF[a]	√	√	SSTL2_II	N / A	N / A	2.5	None
OBUF[a]	√	√	SSTL2_II_DCI	N / A	N / A	2.5	Split
OBUF[a]	√		SSTL3_I	N / A	N / A	3.3	None

Virtex-II, Virtex-II Pro OBUF_*selectIO* IOSTANDARD Attributes

			Attribute Values				
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Terminate Type**
OBUF[a]	√		SSTL3_I_DCI	N/A	N/A	3.3	Split
OBUF[a]	√		SSTL3_II	N/A	N/A	3.3	None
OBUF[a]	√		SSTL3_II_DCI	N/A	N/A	3.3	Split

[a]Attach an IOSTANDARD attribute to an OBUF and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the output for the I/O standard associated with that value.

[b] The LVTTL, LVCMOS15, LVCMOS18, LVCMOS25, LVCMOS33 attributes also require a slew value (FAST or SLOW) and DRIVE value. See the FAST, SLOW, and DRIVE attribute descriptions in the *Xilinx Constraints Guide*.

Usage

The recommended usage for OBUF_selectIO is to allow the OBUFs be inferred and apply the IOSTANDARD constraint to the input in either the UCF or in the HDL code. OBUF_selectIO can also be instantiated if necessary..

VHDL Instantiation Template

```
-- Component Declaration for OBUF_selectIO should be placed
-- after architecture statement but before begin keyword

component OBUF_selectIO
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OBUF_selectIO
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OBUF_selectIO should be placed
-- in architecture after the begin keyword

OBUF_selectIO_INSTANCE_NAME : OBUF_selectIO
      port map  (O => user_O,
                 I => user_I);
```

Verilog Instantiation Template

```
OBUF_selectIO instance_name (.O (user_O),
                             .I (user_I));
```

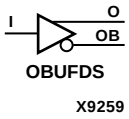
Commonly Used Constraints

DRIVE, IOBDELAY, SLEW, IOSTANDARD

OBUFDS

Differential Signaling Output Buffer with Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



OBUFDS is a single output buffer that supports low-voltage, differential signaling (1.8v CMOS). OBUFDS isolates the internal circuit and provides drive current for signals leaving the chip. Its output is represented as two distinct ports (O and OB), one deemed the "master" and the other the "slave." The master and the slave are opposite phases of the same logical signal (for example, MYNET and MYNETB).

Inputs	Outputs	
I	O	OB
0	0	1
1	1	0

The IOSTANDARD attribute values listed in the following table can be applied to an OBUFDS component to provide selectI/O interface capability. See the *Xilinx Constraints Guide* for information using these attributes.

Component	IOSTANDARD (Attribute Value)	Virtex-II	Virtex-II Pro	Output VCCO	VREF	Terminate Type
OBUFDS[a]	BLVDS_25	√	√	2.5	N/A	TBD
OBUFDS[a]	LDT_25	√	√	2.5	N/A	TBD
OBUFDS[a]	LVDS_25 (default)	√	√	2.5	N/A	None
OBUFDS[a]	LVDS_33 *	√		3.3	N/A	None
OBUFDS[a]	LVDSEXT_25	√	√	2.5	N/A	None
OBUFDS[a]	LVDSEXT_33*	√		3.3	N/A	None
OBUFDS[a]	LVPECL_25*		√	2.5	N/A	None
OBUFDS[a]	LVPECL_33*	√		3.3	N/A	None
OBUFDS[a]	ULVDS_25	√	√	TBD	TBD	TBD

[a]Attach an IOSTANDARD attribute to an OBUFDS and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the output for the I/O standard associated with that value.

Usage

For HDL, this design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OBUFDS should be placed
-- after architecture statement but before begin keyword

component OBUFDS
   port (O  : out STD_ULOGIC;
         OB : out STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OBUFDS
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OBUFDS should be placed
-- in architecture after the begin keyword

OBUFDS_INSTANCE_NAME : OBUFDS
      port map  (O => user_O,
                 OB => user_OB,
                 I => user_I);
```

Verilog Instantiation Template

```
OBUFDS instance_name (.O (user_O),
                      .OB (user_OB),
                      .I (user_I));
```

Commonly Used Constraints

IOSTANDARD, IOBDELAY

OBUFE, 4, 8, 16

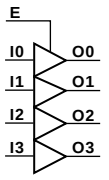
3-State Output Buffers with Active-High Output Enable

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OBUFE	Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUFE4, OBUFE8, OBUFE16	Macro	Macro	Macro	Macro	Macro	Macro

OBUFE

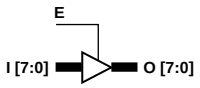
X9447

OBUFE4



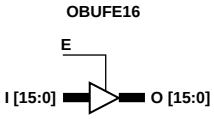
X9448

OBUFE8



X9848

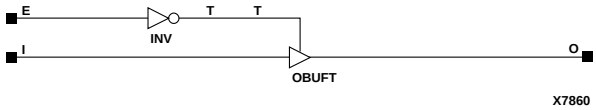
OBUFE16



X9849

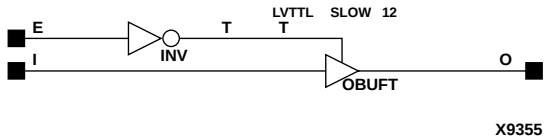
OBUFE, OBUFE4, OBUFE8, and OBUFE16 are 3-state buffers with inputs I, I3 – I0, I7 – I0, and I15-I0, respectively; outputs O, O3 – O0, O7 – O0, and O15-O0, respectively; and active-High output enable (E). When E is High, data on the inputs of the buffers is transferred to the corresponding outputs. When E is Low, the output is High impedance (off or Z state). An OBUFE isolates the internal circuit and provides drive current for signals leaving a chip. An OBUFE output is connected to an OPAD or an IOPAD. An OBUFE input is connected to the internal circuit.

Inputs		Outputs
E	I	O
0	X	Z
1	1	1
1	0	0



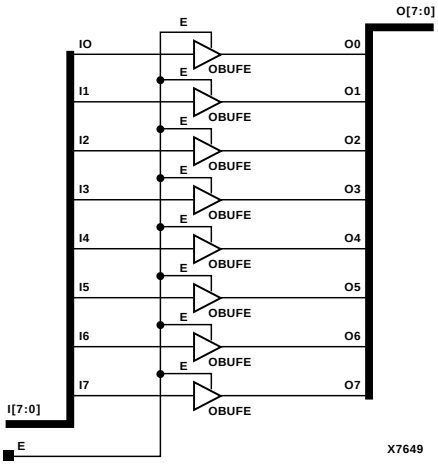
X7860

OBUFE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



X9355

OBUFE Implementation Virtex-II, Virtex-II Pro



OBUFE8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OBUFE should be placed
-- after architecture statement but before begin keyword

component OBUFE
   port (O  : out STD_ULOGIC;
         E  : in STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OBUFE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OBUFE should be placed
-- in architecture after the begin keyword

OBUFE_INSTANCE_NAME : OBUFE
      port map  (O  => user_O,
                 E  => user_E,
                 I  => user_I);
```

Verilog Instantiation Template

```
OBUFE_instance_name (.O (user_O),
                     .E (user_E),
                     .I (user_I));
```

Commonly Used Constraints

IOBDELAY

OBUFT, 4, 8, 16

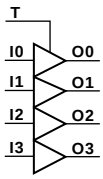
Single and Multiple 3-State Output Buffers with Active-Low Output Enable

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OBUFT	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OBUFT4, OBUFT8, OBUFT16	Macro	Macro	Macro	Macro	Macro	Macro

OBUFT

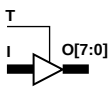
X9449

OBUFT4



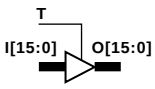
X9450

OBUFT8



X3805

OBUFT16

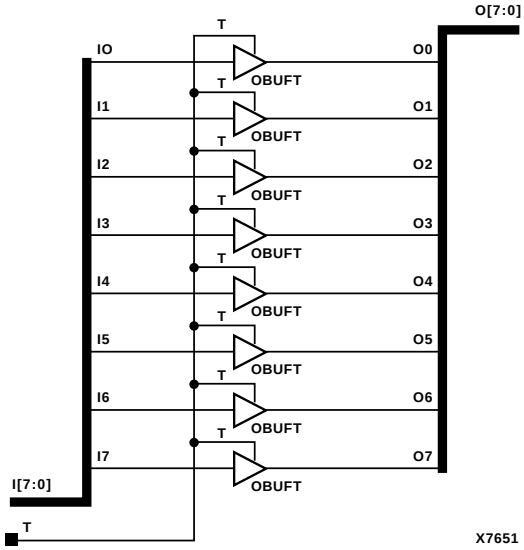


X3817

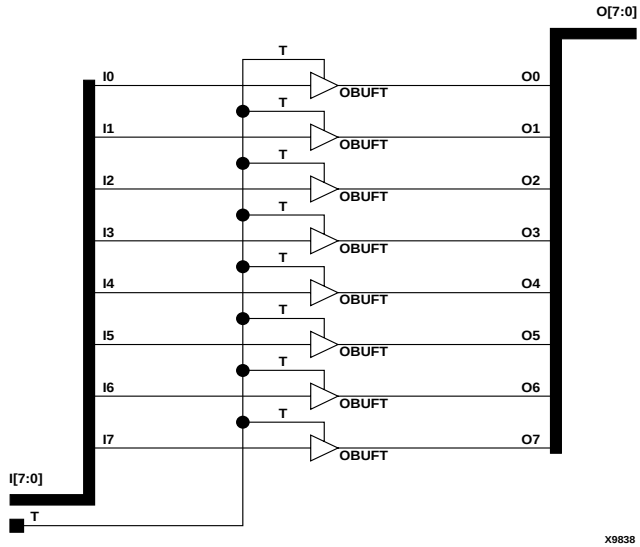
OBUFT, OBUFT4, OBUFT8, and OBUFT16 are single and multiple 3-state output buffers with inputs I, I3 – I0, I7 – I0, I15 – I0, outputs O, O3 – O0, O7 – O0, O15 – O0, and active-Low output enables (T). When T is Low, data on the inputs of the buffers is transferred to the corresponding outputs. When T is High, the output is high impedance (off or Z state). OBUFTs isolate the internal circuit and provide extra drive current for signals leaving a chip. An OBUFT output is connected to an OPAD or an IOPAD.

For Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, and Spartan-IIE, see the "OBUFT_selectIO" section for information on OBUFT variants with selectable I/O interfaces. OBUFT, 4, 8, and 16 use the LVTTL standard. Also, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, and Spartan-IIE, OBUFT, 4, 8, and 16 have selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.

Inputs		Outputs
T	I	O
1	X	Z
0	1	1
0	0	0



OBUFT8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E



OBUFT8 Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, OBUFT is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OBUFT should be placed
-- after architecture statement but before begin keyword

component OBUFT
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OBUFT
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OBUFT should be placed
-- in architecture after the begin keyword

OBUFT_INSTANCE_NAME : OBUFT
      port map  (O => user_O,
                 I => user_I,
                 T => user_T);
```

Verilog Instantiation Template

```
OBUFT instance_name (.O (user_O),
                     .I (user_I),
                     .T (user_T));
```

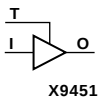
Commonly Used Constraints

DRIVE, IOSTANDARD, IOBDELAY, SLEW

OBUFT_*selectIO*

Single 3-State Output Buffer with Active-Low Output Enable and Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



For Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, and Spartan-IIE, OBUFT and its selectIO variants (listed in the "Component" column in the table below) are single 3-state output buffers with active-Low output Enable whose I/O interface corresponds to a specific I/O standard. The name extensions (LVCMOS2, PCI33_3, PCI33_5, etc.) specify the standard. The S, F, and 2, 4, 6, 8, 12, 16, 24 extensions specify the slew rate (SLOW or FAST) and the drive power (2, 4, 6, 8, 12, 16, 24 mA) for the LVTTL standard. For example, OBUFT_S_4 is a 3-state output buffer with active-Low output enable that uses the LVTTL I/O signaling standard with a SLOW slew and 4mA of drive power. You can attach an IOSTANDARD attribute to an OBUFT instance instead of using an OBUFT_*selectIO* component. Check marks (√) in the "Spartan-II, Virtex," and "Spartan-IIE, Virtex-E" columns indicate the components and IOSTANDARD attribute values available for each architecture.

The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffers. See "SelectI/O Usage Rules" under the IBUF_*selectIO* section for information on using these components and IOSTANDARD attributes.

Spartan-II, Spartan-IIE, Virtex, and Virtex-E OBUFT_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Output VCCO
OBUFT	√	√	defaults to LVTTL	3.3
OBUFT_S_2	√	√	LVTTL [b]	3.3
OBUFT_S_4	√	√	LVTTL [b]	3.3
OBUFT_S_6	√	√	LVTTL [b]	3.3
OBUFT_S_8	√	√	LVTTL [b]	3.3
OBUFT_S_12	√	√	LVTTL [b]	3.3
OBUFT_S_16	√	√	LVTTL [b]	3.3
OBUFT_S_24	√	√	LVTTL [b]	3.3
OBUFT_F_2	√	√	LVTTL [b]	3.3
OBUFT_F_4	√	√	LVTTL [b]	3.3
OBUFT_F_6	√	√	LVTTL [b]	3.3
OBUFT_F_8	√	√	LVTTL [b]	3.3
OBUFT_F_12	√	√	LVTTL [b]	3.3
OBUFT_F_16	√	√	LVTTL [b]	3.3

Spartan-II, Spartan-IIE, Virtex, and Virtex-E OBUFT_*selectIO* Components and IOSTANDARD Attributes

Component	Spartan-II, Virtex	Spartan-IIE, Virtex-E	IOSTANDARD (Attribute Value)	Output VCCO
OBUFT_F_24	√	√	LVTTL [b]	3.3
OBUFT_AGP	√	√	AGP	3.3
OBUFT_CTT	√	√	CTT	3.3
OBUFT_GTL	√	√	GTL	N / A
OBUFT_GTLP	√	√	GTLP	N / A
OBUFT_HSTL_I	√	√	HSTL_I	1.5
OBUFT_HSTL_III	√	√	HSTL_III	1.5
OBUFT_HSTL_IV	√		HSTL_IV	1.5
OBUFT_LVCMOS2	√	√	LVCMOS2	2.5
OBUFT_LVCMOS18		√	LVCMOS18 [b]	1.8
OBUFT_LVDS		√	LVDS	2.5
OBUFT_LVPECL		√	LVPECL	3.3
OBUFT_PCI33_3	√	√	PCI33_3	3.3
OBUFT_PCI33_5	√	√	PCI33_5	3.3
OBUFT_PCI66_3	√	√	PCI66_3	3.3
OBUFT_PCIX66_3		√	PCIX66_3	3.3
OBUFT_SSTL2_I	√	√	SSTL2_I	2.5
OBUFT_SSTL2_II	√	√	SSTL2_II	2.5
OBUFT_SSTL3_I	√	√	SSTL3_I	3.3
OBUFT_SSTL3_II	√	√	SSTL3_II	3.3

[b] The LVTTL, LVCMOS15, LVCMOS18, LVCMOS25, LVCMOS33 attributes also require a slew value (FAST or SLOW) and DRIVE value. See the FAST, SLOW, and DRIVE attribute descriptions in the *Xilinx Constraints Guide* for valid values for each architecture.

The Virtex-II and Virtex-II Pro library includes some OBUFT_*selectIO* components for compatibility with older, existing designs and other architectures. For new Virtex-II and Virtex-II Pro designs, however, the recommended method for using OBUFT selectI/O buffers is to attach an IOSTANDARD attribute to an OBUFT component. For example, attach IOSTANDARD=GTLP to an OBUFT instead of using the OBUFT_GTLP component for new Virtex-II and Virtex-II Pro designs. The IOSTANDARD attributes that can be attached to an OBUFT component are listed in the "IOSTANDARD (Attribute Value)" column in the following table. See "SelectI/O Usage Rules" for information on using these IOSTANDARD attributes.

Virtex-II, Virtex-II Pro OBUFT_*selectIO* IOSTANDARD Attributes

			Attribute Values				
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Terminate Type**
OBUFT[a]	√		AGP	N/A	N/A	3.3	None
OBUFT[a]	√	√	GTL	N/A	N/A	N/A	None
OBUFT[a]	√	√	GTL_DCI	N/A	N/A	1.2	Single
OBUFT[a]	√	√	GTLP	N/A	N/A	N/A	None
OBUFT[a]	√	√	GTLP_DCI	N/A	N/A	1.5	Single
OBUFT[a]	√	√	HSTL_I	N/A	N/A	1.5	None
OBUFT[a]	√	√	HSTL_I_18	N/A	N/A	1.8	None
OBUFT[a]	√	√	HSTL_I_DCI	N/A	N/A	1.5	Split
OBUFT[a]	√	√	HSTL_I_DCI_18	N/A	N/A	1.8	Split
OBUFT[a]	√	√	HSTL_II	N/A	N/A	1.5	None
OBUFT[a]	√	√	HSTL_II_18	N/A	N/A	1.8	None
OBUFT[a]	√	√	HSTL_II_DCI	N/A	N/A	1.5	Split
OBUFT[a]	√	√	HSTL_II_DCI_18	N/A	N/A	1.8	Split
OBUFT[a]	√	√	HSTL_III	N/A	N/A	1.5	None
OBUFT[a]	√	√	HSTL_III_18	N/A	N/A	1.8	None
OBUFT[a]	√	√	HSTL_III_DCI	N/A	N/A	1.5	Split
OBUFT[a]	√	√	HSTL_III_DCI_18	N/A	N/A	1.8	Split
OBUFT[a]	√	√	HSTL_IV	N/A	N/A	1.5	None
OBUFT[a]	√	√	HSTL_IV_18	N/A	N/A	1.8	None
OBUFT[a]	√	√	HSTL_IV_DCI	N/A	N/A	1.5	Split
OBUFT[a]	√	√	HSTL_IV_DCI_18	N/A	N/A	1.8	Split
OBUFT[a]	√	√	LVCMOS15 [b]	N/A	N/A	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	12	F	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	16	F	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	2	F	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	4	F	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	6	F	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	8	F	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	12	S	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	16	S	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	2	S	1.5	None

Virtex-II, Virtex-II Pro OBUFT_*selectIO* IOSTANDARD Attributes

			Attribute Values				
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Terminate Type**
OBUFT[a]	√	√	LVCMOS15[b]	4	S	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	6	S	1.5	None
OBUFT[a]	√	√	LVCMOS15[b]	8	S	1.5	None
OBUFT[a]	√	√	LVCMOS18 [b]	N / A	N / A	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	12	F	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	16	F	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	2	F	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	4	F	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	6	F	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	8	F	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	12	S	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	16	S	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	2	S	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	4	S	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	6	S	1.8	None
OBUFT[a]	√	√	LVCMOS18[b]	8	S	1.8	None
OBUFT[a]	√	√	LVCMOS2 [b]	N / A	N / A	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	N / A	N / A	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	12	F	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	16	F	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	2	F	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	24	F	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	4	F	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	6	F	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	8	F	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	12	S	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	16	S	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	2	S	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	24	S	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	4	S	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	6	S	2.5	None
OBUFT[a]	√	√	LVCMOS25[b]	8	S	2.5	None
OBUFT[a]	√		LVCMOS33 [b]	N / A	N / A	3.3	None
OBUFT[a]	√		LVCMOS33[b]	12	F	3.3	None
OBUFT[a]	√		LVCMOS33[b]	16	F	3.3	None
OBUFT[a]	√		LVCMOS33[b]	2	F	3.3	None
OBUFT[a]	√		LVCMOS33[b]	24	F	3.3	None
OBUFT[a]	√		LVCMOS33[b]	4	F	3.3	None

Virtex-II, Virtex-II Pro OBUFT_*selectIO* IOSTANDARD Attributes

			Attribute Values				
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Terminate Type**
OBUFT[a]	√		LVCMOS33[b]	6	F	3.3	None
OBUFT[a]	√		LVCMOS33[b]	8	F	3.3	None
OBUFT[a]	√		LVCMOS33[b]	12	S	3.3	None
OBUFT[a]	√		LVCMOS33[b]	16	S	3.3	None
OBUFT[a]	√		LVCMOS33[b]	2	S	3.3	None
OBUFT[a]	√		LVCMOS33[b]	24	S	3.3	None
OBUFT[a]	√		LVCMOS33[b]	4	S	3.3	None
OBUFT[a]	√		LVCMOS33[b]	6	S	3.3	None
OBUFT[a]	√		LVCMOS33[b]	8	S	3.3	None
OBUFT[a]	√	√	LVDCI_15	N / A	N / A	1.5	Driver
OBUFT[a]	√	√	LVDCI_18	N / A	N / A	1.8	Driver
OBUFT[a]	√	√	LVDCI_25	N / A	N / A	2.5	Driver
OBUFT[a]	√	√	LVDCI_33	N / A	N / A	3.3	Driver
OBUFT[a]	√	√	LVDCI_DV2_15	N / A	N / A	1.5	Driver
OBUFT[a]	√	√	LVDCI_DV2_18	N / A	N / A	1.8	Driver
OBUFT[a]	√	√	LVDCI_DV2_25	N / A	N / A	2.5	Driver
OBUFT[a]	√		LVDCI_DV2_33	N / A	N / A	3.3	Driver
OBUFT[a]	√	√	LVTTL[b]	N / A	N / A	3.3	None
OBUFT[a]	√	√	LVTTL[b]	12	F	3.3	None
OBUFT[a]	√	√	LVTTL[b]	16	F	3.3	None
OBUFT[a]	√	√	LVTTL[b]	2	F	3.3	None
OBUFT[a]	√	√	LVTTL[b]	24	F	3.3	None
OBUFT[a]	√	√	LVTTL [b]	4	F	3.3	None
OBUFT[a]	√	√	LVTTL[b]	6	F	3.3	None
OBUFT[a]	√	√	LVTTL[b]	8	F	3.3	None
OBUFT[a]	√	√	LVTTL[b]	12	S	3.3	None
OBUFT[a]	√	√	LVTTL[b]	16	S	3.3	None
OBUFT[a]	√	√	LVTTL[b]	2	S	3.3	None
OBUFT[a]	√	√	LVTTL[b]	24	S	3.3	None
OBUFT[a]	√	√	LVTTL[b]	4	S	3.3	None
OBUFT[a]	√	√	LVTTL[b]	6	S	3.3	None
OBUFT[a]	√	√	LVTTL[b]	8	S	3.3	None
OBUFT[a]	√	√	PCI33_3	N / A	N / A	3.3	None
OBUFT[a]	√	√	PCI66_3	N / A	N / A	3.3	None
OBUFT[a]	√		PCIX	N / A	N / A	3.3	None
OBUFT[a]	√	√	SSTL18_I	N / A	N / A	1.8	None
OBUFT[a]	√	√	SSTL18_I_DCI	N / A	N / A	1.8	Split
OBUFT[a]	√	√	SSTL18_II	N / A	N / A	1.8	None

Virtex-II, Virtex-II Pro OBUFT_*selectIO* IOSTANDARD Attributes

			Attribute Values				
Component	**Virtex-II**	**Virtex-II Pro**	**IOSTANDARD**	**Drive**	**Slew**	**Output VCCO**	**Terminate Type**
OBUFT[a]	√	√	SSTL18_II_DCI	N / A	N / A	1.8	Split
OBUFT[a]	√	√	SSTL2_I	N / A	N / A	2.5	None
OBUFT[a]	√	√	SSTL2_I_DCI	N / A	N / A	2.5	Split
OBUFT[a]	√	√	SSTL2_II	N / A	N / A	2.5	None
OBUFT[a]	√	√	SSTL2_II_DCI	N / A	N / A	2.5	Split
OBUFT[a]	√		SSTL3_I	N / A	N / A	3.3	None
OBUFT[a]	√		SSTL3_I_DCI	N / A	N / A	3.3	Split
OBUFT[a]	√		SSTL3_II	N / A	N / A	3.3	None
OBUFT[a]	√		SSTL3_II_DCI	N / A	N / A	3.3	Split

[a]Attach an IOSTANDARD attribute to an OBUFT and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the output for the I/O standard associated with that value.

[b] The LVTTL, LVCMOS15, LVCMOS18, LVCMOS25, LVCMOS33 attributes also require a slew value (FAST or SLOW) and DRIVE value. See the FAST, SLOW, and DRIVE attribute descriptions in the *Xilinx Constraints Guide* for valid values for each architecture.

OBUFT and its variants have selectable drive and slew rates using the DRIVE and FAST or SLOW constraints. The defaults are DRIVE=12 mA and SLOW slew.

When T is Low, data on the input of the buffer is transferred to the output. When T is High, the output is high impedance (off or Z state). OBUFTs isolate the internal circuit and provide extra drive current for signals leaving a chip. An OBUFT_*selectIO* output is connected to an OPAD or an IOPAD.

Usage

For HDL, these design elements are instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OBUFT_selectIO should be placed
-- after architecture statement but before begin keyword

component OBUFT_selectIO
   port (O  : out STD_ULOGIC;
         I  : in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OBUFT_selectIO
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OBUFT_selectIO should be
-- placed in architecture after the begin keyword
```

```
OBUFT_selectIO_INSTANCE_NAME : OBUFT_selectIO
      port map  (O => user_O,
                 I => user_I,
                 T => user_T);
```

Verilog Instantiation Template

```
OBUFT_selectIO instance_name (.O (user_O),
                              .I (user_I),
                              .T (user_T));
```

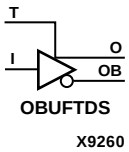
Commonly Used Constraints

DRIVE, IOBDELAY, SLEW, IOSTANDARD

OBUFTDS

3-State Differential Signaling Output Buffer with Active Low Output Enable and Selectable I/O Interface

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



OBUFTDS is a single 3-state, differential signaling output buffer with active Low enable and a Select I/O interface.

When T is Low, data on the input of the buffer is transferred to the output (O) and inverted output (OB). When T is High, both outputs are high impedance (off or Z state).

Inputs		Outputs	
I	T	O	OB
X	1	Z	Z
0	0	0	1
1	0	1	0

The IOSTANDARD attribute values listed in the following table can be applied to an IBUFGDS component to provide selectIO interface capability. See the *Xilinx Constraints Guide* for information using these attributes. The hardware implementation of the I/O standards requires that you follow a set of usage rules for the SelectI/O buffer components. See the "SelectI/O Usage Rules" under the IBUF_*selectIO* section for information on using these IOSTANDARD attributes.

Component	IOSTANDARD (Attribute Value)	Virtex-II	Virtex-II Pro	Output VCCO	VREF	Terminate Type
OBUFTDS[a]	BLVDS_25	√	√	2.5	N/A	None
OBUFTDS[a]	LDT_25	√	√	2.5	N/A	None
OBUFTDS[a]	LVDS_25 (default)	√	√	2.5	N/A	None
OBUFTDS[a]	LVDS_33*	√		3.3	N/A	None
OBUFTDS[a]	LVPECL_25*		√	2.5	N/A	None
OBUFTDS[a]	LVPECL_33*	√		3.3	N/A	None
OBUFTDS[a]	LVDSEXT_25	√	√	2.5	N/A	None
OBUFTDS[a]	LVDSEXT_33*	√		3.3	N/A	None
OBUFTDS[a]	ULVDS_25	√	√	2.5	N/A	None

[a]Attach an IOSTANDARD attribute to an OBUFTDS and assign the value indicated in the "IOSTANDARD (Attribute Value)" column to program the outputs for the I/O standard associated with that value.

Usage

For HDL, these design elements are instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OBUFTDS should be placed
-- after architecture statement but before begin keyword

component OBUFTDS
   port (O  : out STD_ULOGIC;
         OB : out STD_ULOGIC;
         I  : in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OBUFTDS
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OBUFTDS should be
-- placed in architecture after the begin keyword

OBUFTDS_INSTANCE_NAME : OBUFTDS
      port map  (O => user_O,
                 OB => user_OB,
                 I => user_I,
                 T => user_T);
```

Verilog Instantiation Template

```
OBUFTDS instance_name (.O (user_O),
                       .OB (user_OB),
                       .I (user_I),
                       .T (user_T));
```

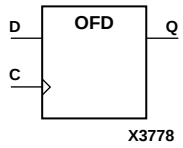
Commonly Used Constraints

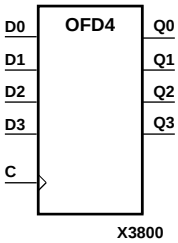
IOSTANDARD, IOBDELAY

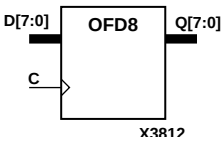
OFD, 4, 8, 16

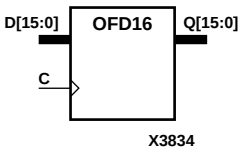
Single- and Multiple-Output D Flip-Flops

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OFD	Macro	Macro	Macro	Macro	Macro	Macro
OFD4, OFD8, OFD16	Macro	Macro	Macro	Macro	Macro	Macro









OFD, OFD4, OFD8, and OFD16 are single and multiple output D flip-flops.

The outputs (for example, Q3 – Q0) are connected to OPADs or IOPADs. The data on the D inputs is loaded into the flip-flops during the Low-to-High clock (C) transition and appears on the Q outputs.

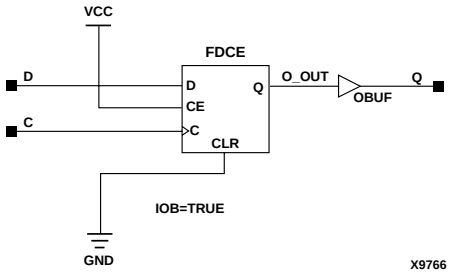
The flip-flops are asynchronously cleared with Low outputs when power is applied, or when global reset is active.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

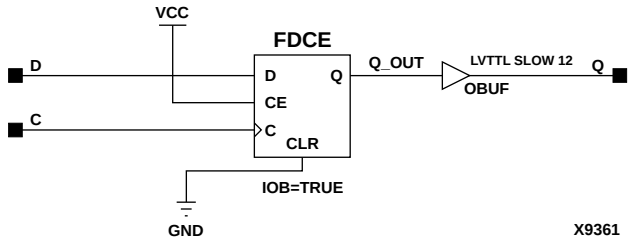
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

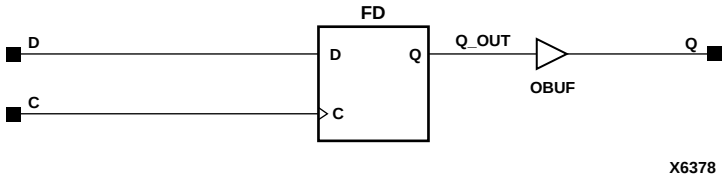
Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1



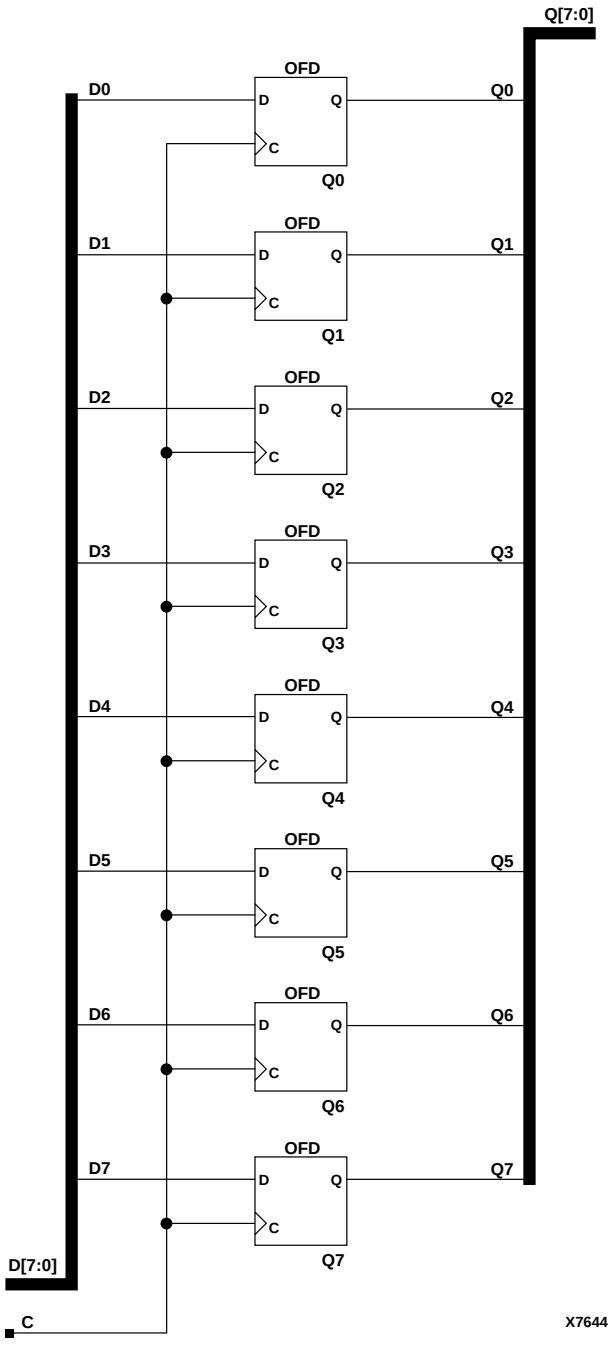
OFD Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



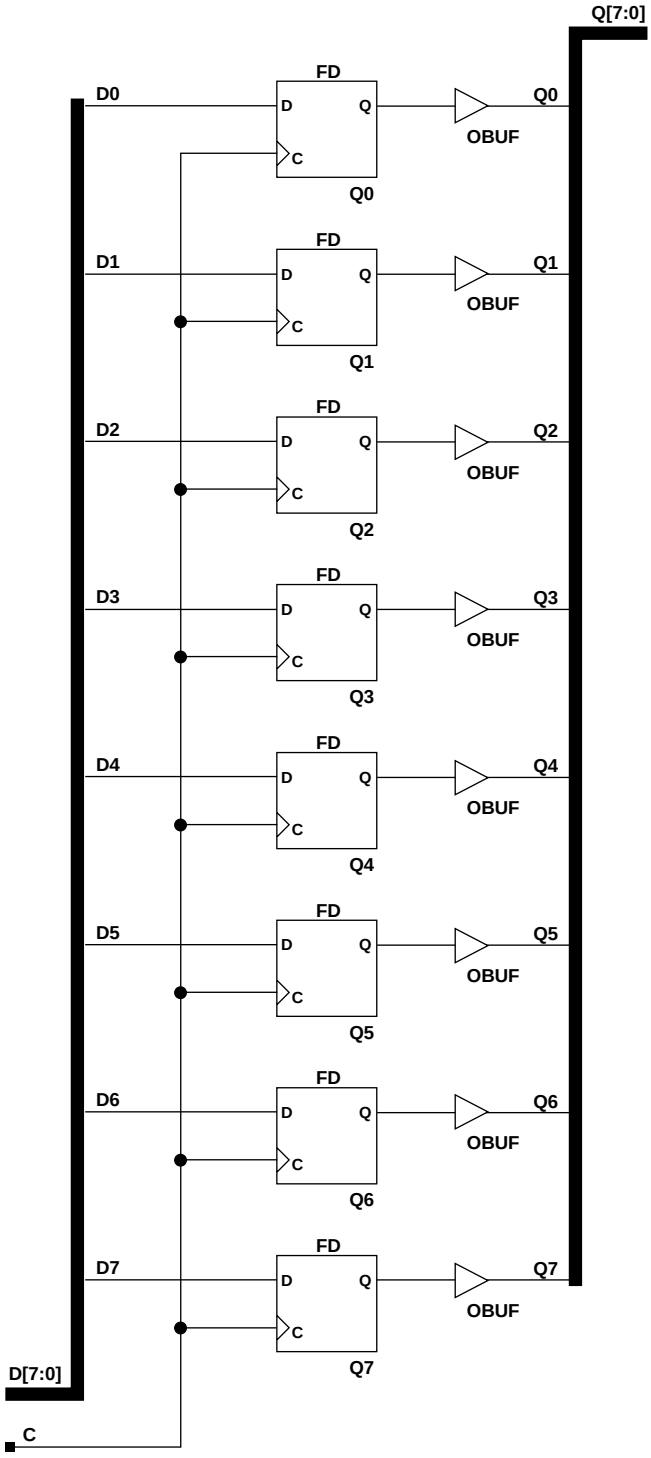
OFD Implementation Virtex-II, Virtex-II Pro



OFD Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



OFD8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



X7648

OFD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

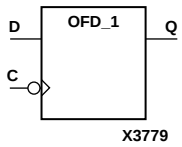
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFD, you would infer an FD and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFD_1

Output D Flip-Flop with Inverted Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



OFD_1 is located in an input / output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output.

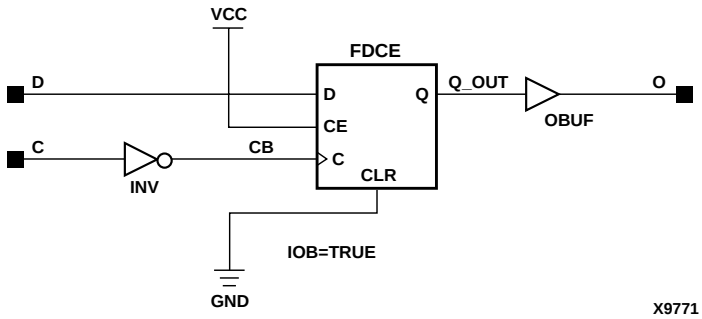
The flip-flop is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
D	↓	d

d = state of referenced input one setup time prior to active clock transition



OFD_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

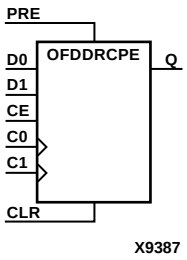
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFD_1, you would infer an FD_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFDDRCPE

Dual Data Rate Output D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



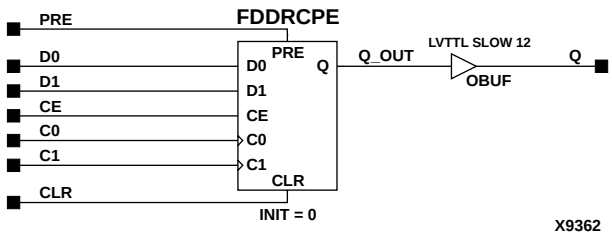
OFDDRCPE is a dual data rate (DDR) output D flip-flop with clock enable (CE) and asynchronous preset (PRE) and clear (CLR). It consists of one output buffer and one dual data rate flip-flop (FDDRCPE).

When the asynchronous PRE is High and CLR is Low, the Q output is preset High. When CLR is High, Q is set Low. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C1 clock transition.

The INIT attribute does not apply to OFDDRCPE components.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

Inputs							Outputs
C0	C1	CE	D0	D1	CLR	PRE	Q
X	X	X	X	X	1	0	0
X	X	X	X	X	0	1	1
X	X	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1



OFDDRCPE Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, the OFDDRCPE design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OFDDRCPE should be placed
-- after architecture statement but before begin keyword

component OFDDRCPE
   port (Q  : out STD_ULOGIC;
         C0 : in STD_ULOGIC;
         C1 : in STD_ULOGIC;
         CE : in STD_ULOGIC;
         CLR: in STD_ULOGIC;
         D0 : in STD_ULOGIC;
         D1 : in STD_ULOGIC;
         PRE: in STD_ULOGIC);
end component;

-- Component Attribute specification for OFDDRCPE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OFDDRCPE should be placed
-- in architecture after the begin keyword

OFDDRCPE_INSTANCE_NAME : OFDDRCPE
      port map(Q   => user_O,
               C0  => user_C0,
               C1  => user_C1,
               CE  => user_CE,
               CLR => user_CLR,
               D0  => user_D0,
               D1  => user_D1,
               PRE => user_PRE);
```

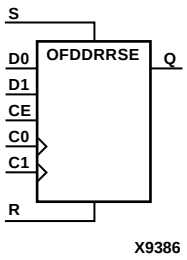
Verilog Instantiation Template

```
OFDDRCPE instance_name (.Q  (user_O),
                        .C0 (user_C0),
                        .C1  (user_C1),
                        .CE  (user_CE),
                        .CLR (user_CLR),
                        .D0  (user_D0),
                        .D1  (user_D1),
                        .PRE (user_I));
```

OFDDRRSE

Dual Data Rate Output D Flip-Flop with Synchronous Reset and Set and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



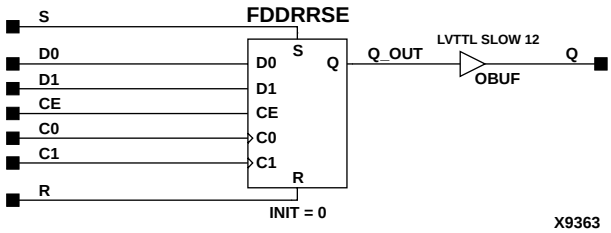
OFDDRRSE is a dual data rate (DDR) output D flip-flop with synchronous reset (R) and set (S) and clock enable (CE). It consists of one output buffer and one dual data rate flip-flop (FDDRRSE).

On a Low-to-High clock transition (C0 or C1), a High R input resets the Q output Low; a Low R input with a High S input sets Q High. When both R and S are Low and clock enable is High, data on the D0 input is loaded into the flip-flop on a Low-to-High C0 clock transition and data on the D1 input is loaded into the flip-flop on a Low-to-High C1 clock transition.

The flip-flops are asynchronously cleared with Low outputs when power is applied, or when global reset is active.

The INIT attribute does not apply to OFDDRRSE components.

Inputs							Outputs
C0	C1	CE	D0	D1	R	S	Q
↑	X	X	X	X	1	0	0
↑	X	X	X	X	0	1	1
↑	X	X	X	X	1	1	0
X	↑	X	X	X	1	0	0
X	↑	X	X	X	0	1	1
X	↑	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1



OFDDRRSE Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, the OFDDRRSE design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OFDDRRSE should be placed
-- after architecture statement but before begin keyword

component OFDDRRSE
   port (Q  : out STD_ULOGIC;
         C0 : in STD_ULOGIC;
         C1 : in STD_ULOGIC;
         CE : in STD_ULOGIC;
         D0 : in STD_ULOGIC;
         D1 : in STD_ULOGIC;
         R  : in STD_ULOGIC;
         S  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OFDDRRSE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OFDDRRSE should be placed
-- in architecture after the begin keyword

OFDDRRSE_INSTANCE_NAME : OFDDRRSE
      port map(Q   => user_O,
               C0  => user_C0,
               C1  => user_C1,
               CE  => user_CE,
               D0  => user_D0,
               D1  => user_D1,
               R   => user_R,
               S   => user_S);
```

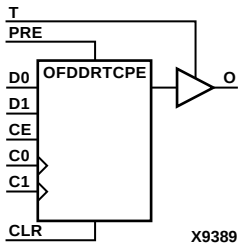
Verilog Instantiation Template

```
OFDDRRSE instance_name (.Q (user_Q),
                        .C0 (user_C0),
                        .C1 (user_C1),
                        .CE (user_CE),
                        .D0 (user_D0),
                        .D1 (user_D1),
                        .R (user_R),
                        .S (user_S));
```

OFDDRTCPE

Dual Data Rate D Flip-Flop with Active-Low 3-State Output Buffer, Clock Enable, and Asynchronous Preset and Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



OFDDRTCPE is a dual data rate (DDR) D flip-flop with clock enable (CE) and asynchronous preset and clear whose output is enabled by a 3-state buffer. It consists of a dual data rate flip-flop (FDDRCPE) and a 3-state output buffer (OBUFT). The data output (O) of the flip-flop is connected to the input of the output buffer (OBUFT). The output of the OBUFT is connected to an OPAD or IOPAD.

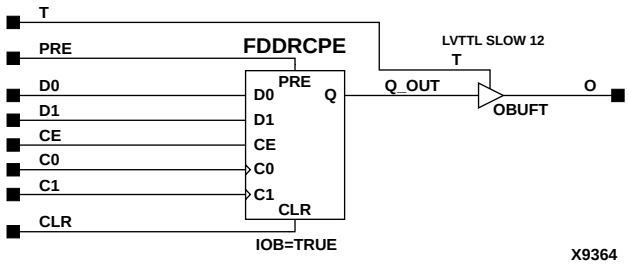
When the active-Low enable input (T) is Low, output is enabled and the data on the flip-flop's Q output appears on the OBUFT's O output. When the asynchronous PRE is High and CLR is Low, the O output is preset High. When CLR is High, O is set Low. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C1 clock transition.

When T is High, outputs are high impedance (Off). When CE is Low and T is Low, the outputs do not change.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

The INIT attribute does not apply to OFDDRTCPE components.

Inputs								Outputs
C0	C1	CE	D0	D1	CLR	PRE	T	O
X	X	X	X	X	X	X	1	Z
X	X	X	X	X	1	0	0	0
X	X	X	X	X	0	1	0	1
X	X	X	X	X	1	1	0	0
X	X	0	X	X	0	0	0	No Chg
↑	X	1	D0	X	0	0	0	D0
X	↑	1	X	D1	0	0	0	D1



OFDDRTCPE Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, the OFDDRTCPE design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OFDDRTCPE should be placed
-- after architecture statement but before begin keyword

component OFDDRTCPE
   port (O  : out STD_ULOGIC;
         C0 : in STD_ULOGIC;
         C1 : in STD_ULOGIC;
         CE : in STD_ULOGIC;
         CLR: in STD_ULOGIC;
         D0 : in STD_ULOGIC;
         D1 : in STD_ULOGIC;
         PRE: in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OFDDRTCPE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OFDDRTCPE should be placed
-- in architecture after the begin keyword

OFDDRTCPE_INSTANCE_NAME : OFDDRTCPE
      port map(O   => user_O,
               C0  => user_C0,
               C1  => user_C1,
               CE  => user_CE,
               CLR  => user_CLR,
               D0  => user_D0,
               D1  => user_D1,
               PRE => user_PRE,
               T   => user_T);
```

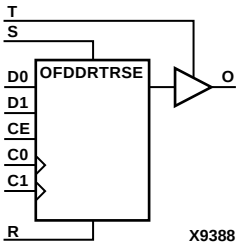
Verilog Instantiation Template

```
OFDDRTCPE instance_name (.O (user_O),
                         .C0 (user_C0),
                         .C1 (user_C1),
                         .CE (user_CE),
                         .CLR (user_CLR),
                         .D0 (user_D0),
                         .D1 (user_D1),
                         .PRE (user_PRE),
                         .T (user_T));
```

OFDDRTRSE

Dual Data Rate D Flip-Flop with Active-Low 3-State Output Buffer, Synchronous Reset and Set, and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



OFDDRTRSE is a dual data rate (DDR) D flip-flop with clock enable (CE) and synchronous reset and set whose output is enabled by a 3-state buffer. It consists of a dual data rate flip-flop (FDDRRSE) and a 3-state output buffer (OBUFT). The data output (O) of the flip-flop is connected to the input of the output buffer (OBUFT). The output of the OBUFT is connected to an OPAD or IOPAD.

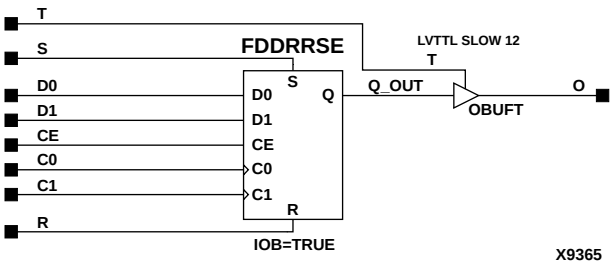
When the active-Low enable input (T) is Low, output is enabled and the data on the flip-flop's Q output appears on the OBUFT's O output. On a Low-to-High clock transition (C0 or C1), a High R input resets the Q output Low; a Low R input with a High S input sets O High. When both R and S are Low and clock enable is High, data on the D0 input is loaded into the flip-flop on a Low-to-High C0 clock transition and data on the D1 input is loaded into the flip-flop on a Low-to-High C1 clock transition.

When T is High, outputs are high impedance (Off). When CE is Low and T is Low, the outputs do not change.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

The INIT attribute does not apply to OFDDRTRSE components.

Inputs								Outputs
C0	C1	CE	D0	D1	R	S	T	O
X	X	X	X	X	X	X	1	Z
↑	X	X	X	X	1	0	0	0
↑	X	X	X	X	0	1	0	1
↑	X	X	X	X	1	1	0	0
X	↑	X	X	X	1	0	0	0
X	↑	X	X	X	0	1	0	1
X	↑	X	X	X	1	1	0	0
X	X	0	X	X	0	0	0	No Chg
↑	X	1	D0	X	0	0	0	D0
X	↑	1	X	D1	0	0	0	D1



OFDDRTRSE Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, the OFDDRTRSE design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OFDDRTRSE should be placed
-- after architecture statement but before begin keyword

component OFDDRTRSE
   port (O  : out STD_ULOGIC;
         C0 : in STD_ULOGIC;
         C1 : in STD_ULOGIC;
         CE : in STD_ULOGIC;
         D0 : in STD_ULOGIC;
         D1 : in STD_ULOGIC;
         R  : in STD_ULOGIC;
         S  : in STD_ULOGIC;
         T  : in STD_ULOGIC);
end component;

-- Component Attribute specification for OFDDRTRSE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for OFDDRTRSE should be placed
-- in architecture after the begin keyword

OFDDRTRSE_INSTANCE_NAME : OFDDRTRSE
      port map(O   => user_O,
               C0  => user_C0,
               C1  => user_C1,
               CE  => user_CE,
               D0  => user_D0,
               D1  => user_D1,
               R   => user_R,
               S   => user_S,
               T   => user_T);
```

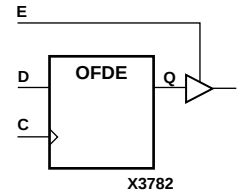
Verilog Instantiation Template

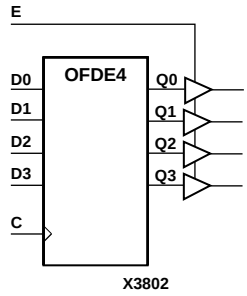
```
OFDDRTRSE instance_name (.O (user_O),
                         .C0 (user_C0),
                         .C1 (user_C1),
                         .CE (user_CE),
                         .D0 (user_D0),
                         .D1 (user_D1),
                         .R (user_R),
                         .S (user_S),
                         .T (user_T));
```

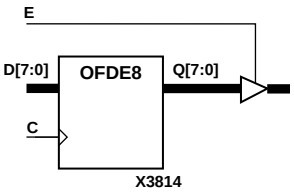
OFDE, 4, 8, 16

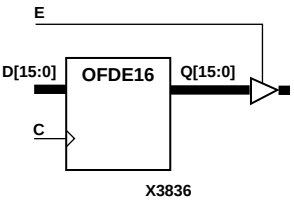
D Flip-Flops with Active-High Enable Output Buffers

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OFDE, OFDE4, OFDE8, OFDE16	Macro	Macro	Macro	Macro	Macro	Macro









OFDE, OFDE4, OFDE8, and OFDE16 are single or multiple D flip-flops whose outputs are enabled by 3-state buffers. The flip-flop data outputs (Q) are connected to the inputs of output buffers (OBUFE). The OBUFE outputs (O) are connected to OPADs or IOPADs. The data on the data inputs (D) is loaded into the flip-flops during the Low-to-High clock (C) transition. When the active-High enable inputs (E) are High, the data on the flip-flop outputs (Q) appears on the O outputs. When E is Low, outputs are high impedance (Z state or Off).

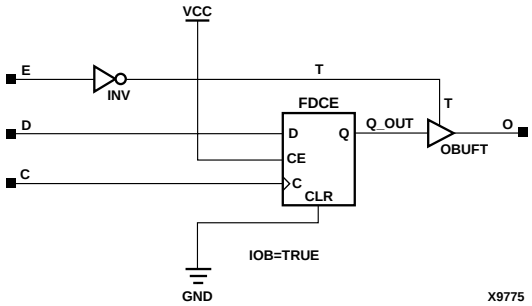
The flip-flops are asynchronously cleared with Low outputs when power is applied, or when global reset is active.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

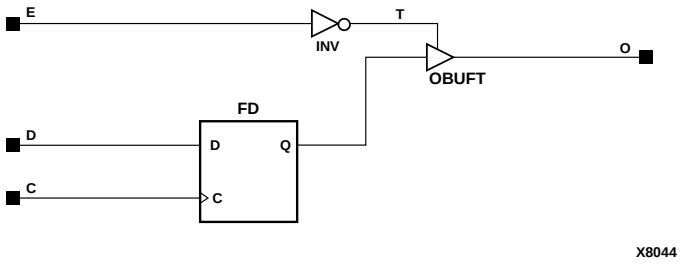
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

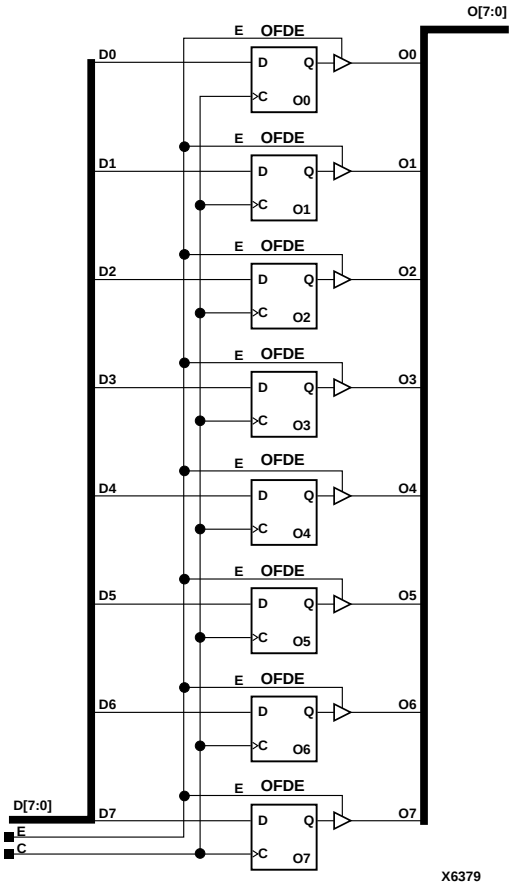
Inputs			Outputs
E	D	C	O
0	X	X	Z
1	1	↑	1
1	0	↑	0



OFDE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



OFDE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



OFDE8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

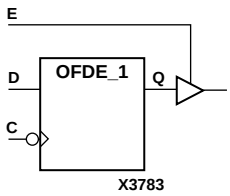
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDE, you would infer an FDE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFDE_1

D Flip-Flop with Active-High Enable Output Buffer and Inverted Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



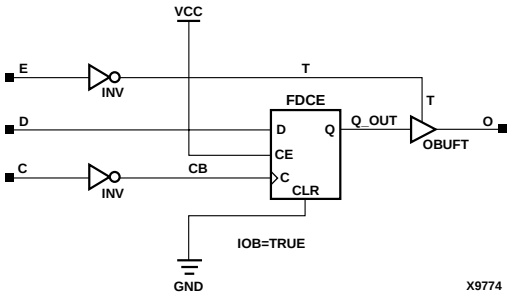
OFDE_1 and its output buffer are located in an input / output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer or OBUFE. The output of the OBUFE is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-High enable input (E) is High, the data on the flip-flop output (Q) appears on the O output. When E is Low, the output is high impedance (Z state or Off).

The flip-flop is asynchronously cleared with Low output when power is applied, or when global reset is active.

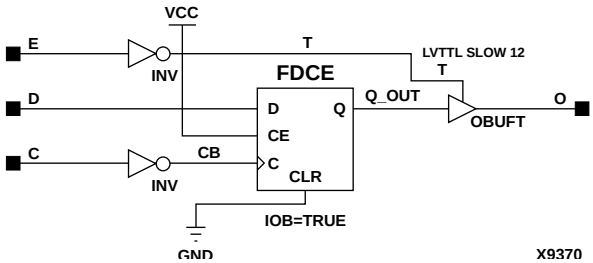
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
E	D	C	O
0	X	X	Z
1	1	↓	1
1	0	↓	0



OFDE_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDE_1 Implementation Virtex-II, Virtex-II Pro

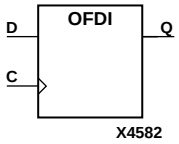
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDE_1, you would infer an FDE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFDI

Output D Flip-Flop (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



OFDI is contained in an input / output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q).

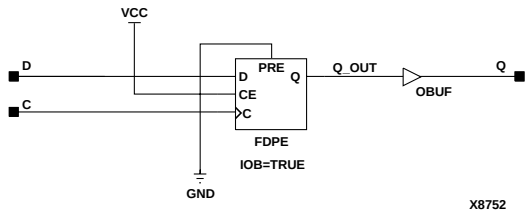
The flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

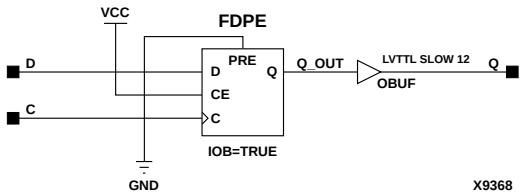
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
D	↑	d

d = state of referenced input one setup time prior to active clock transition



OFDI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDI Implementation Virtex-II, Virtex-II Pro

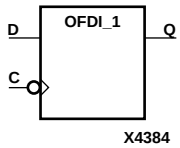
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDI, you would infer an FDP and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFDI_1

Output D Flip-Flop with Inverted Clock (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



OFDI_1 exists in an input / output block (IOB). The D flip-flop output (Q) is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output.

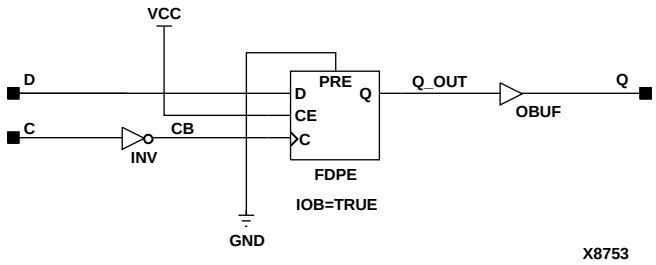
The flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

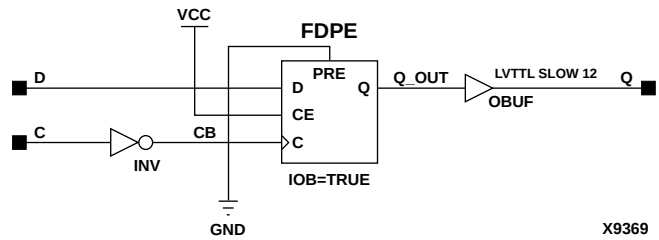
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
D	↓	d

d = state of referenced input one setup time prior to the active clock transition



OFDI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDI_1 Implementation Virtex-II, Virtex-II Pro

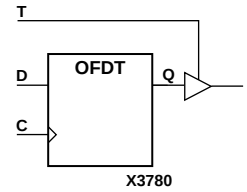
Usage

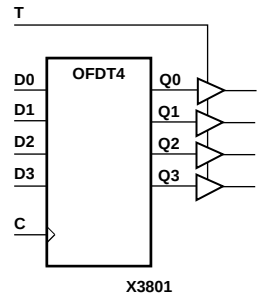
This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDI_1, you would infer an FDP_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

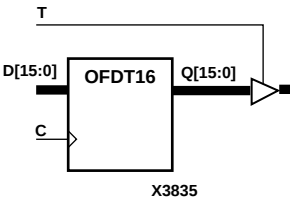
OFDT, 4, 8, 16

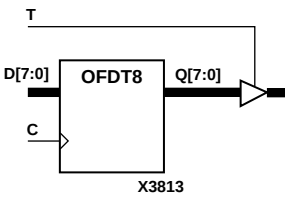
Single and Multiple D Flip-Flops with Active-Low 3-State Output Enable Buffers

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OFDT	Macro	Macro	Macro	Macro	Macro	Macro
OFDT4, OFDT8, OFDT16	Macro	Macro	Macro	Macro	Macro	Macro









OFDT, OFDT4, OFDT8, and OFDT16 are single or multiple D flip-flops whose outputs are enabled by a 3-state buffers. The data outputs (Q) of the flip-flops are connected to the inputs of output buffers (OBUFT). The outputs of the OBUFTs (O) are connected to OPADs or IOPADs. The data on the data inputs (D) is loaded into the flip-flops during the Low-to-High clock (C) transition. When the active-Low enable inputs (T) are Low, the data on the flip-flop outputs (Q) appears on the O outputs. When T is High, outputs are high impedance (Off).

The flip-flops are asynchronously cleared with Low outputs, when power is applied, or when global reset is active.

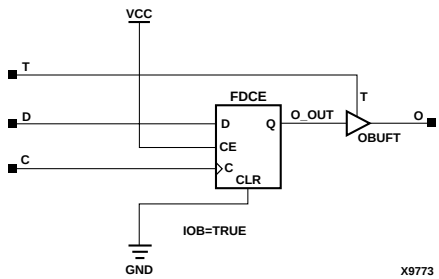
For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

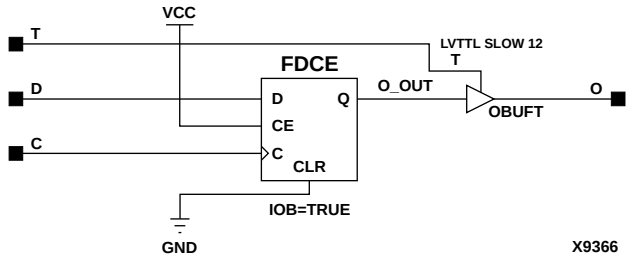
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
T	D	C	O
1	X	X	Z
0	D	↑	d

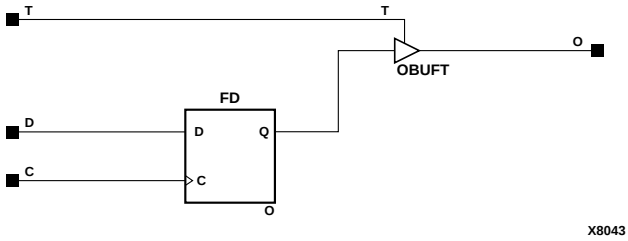
d = state of referenced input one setup time prior to active clock transition



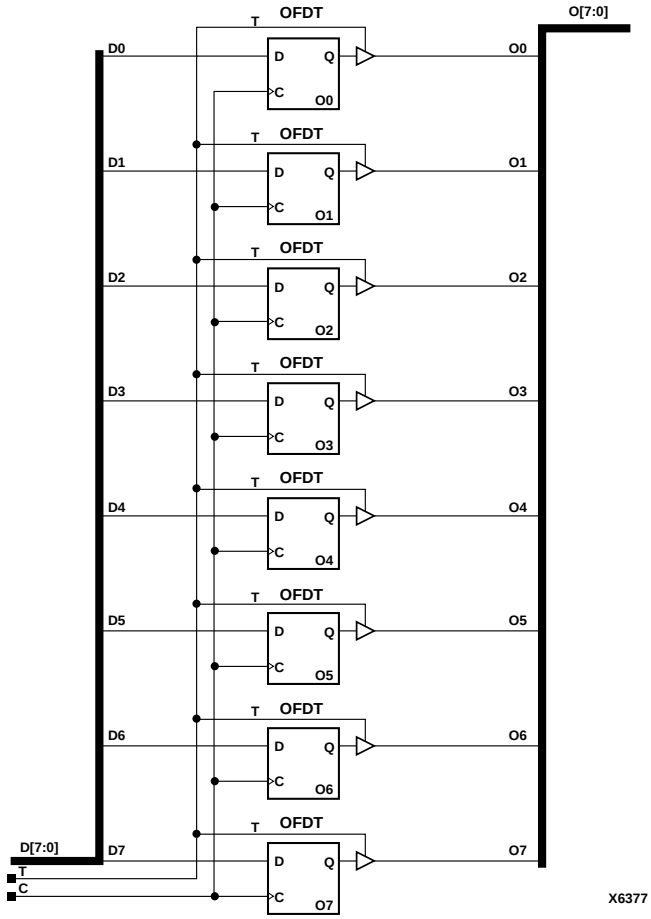
OFDT Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDT Implementation Virtex-II, Virtex-II Pro



OFDT Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



OFDT8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

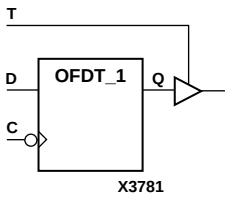
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDT, you would infer an FDCE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFDT_1

D Flip-Flop with Active-Low 3-State Output Buffer and Inverted Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



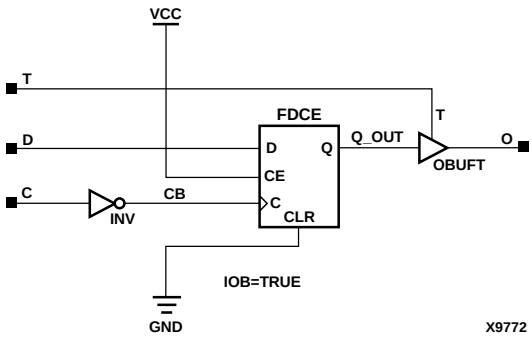
OFDT_1 and its output buffer are located in an input / output block (IOB). The flip-flop data output (Q) is connected to the input of an output buffer (OBUFT). The OBUFT output is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-Low enable input (T) is Low, the data on the flip-flop output (Q) appears on the O output. When T is High, the output is high impedance (Off).

The flip-flop is asynchronously cleared with Low output when power is applied, or when global reset is active.

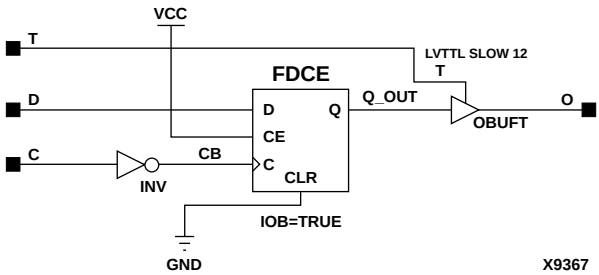
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
T	D	C	O
1	X	X	Z
0	1	↓	1
0	0	↓	0



OFDT_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDT_1 Implementation Virtex-II, Virtex-II Pro

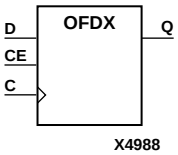
Usage

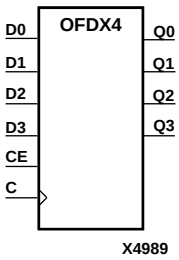
This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDT_1, you would infer an FDCE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

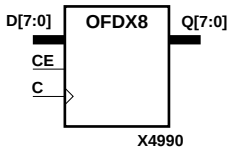
OFDX, 4, 8, 16

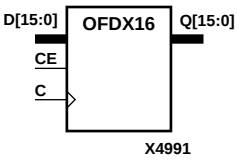
Single- and Multiple-Output D Flip-Flops with Clock Enable

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OFDX	Macro	Macro	Macro	N/A	N/A	N/A
OFDX4, OFDX8, OFDX16	Macro	Macro	Macro	N/A	N/A	N/A









OFDX, OFDX4, OFDX8, and OFDX16 are single and multiple output D flip-flops. The Q outputs are connected to OPADs or IOPADs. The data on the D inputs is loaded into the flip-flops during the Low-to-High clock (C) transition and appears on the Q outputs. When CE is Low, flip-flop outputs do not change.

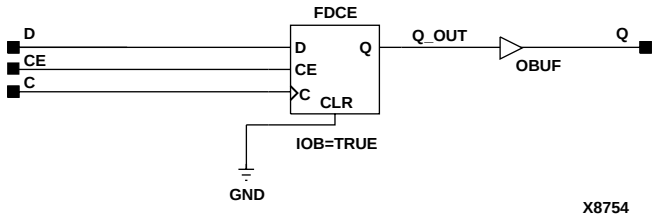
The flip-flops are asynchronously cleared with Low outputs, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set/reset (GSR) is active.

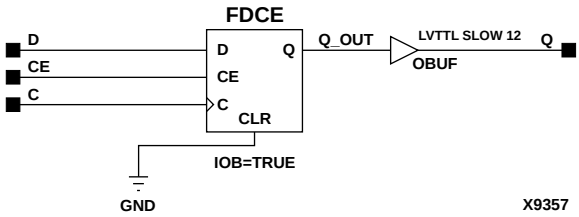
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
1	Dn	↑	dn
0	X	X	No Chg

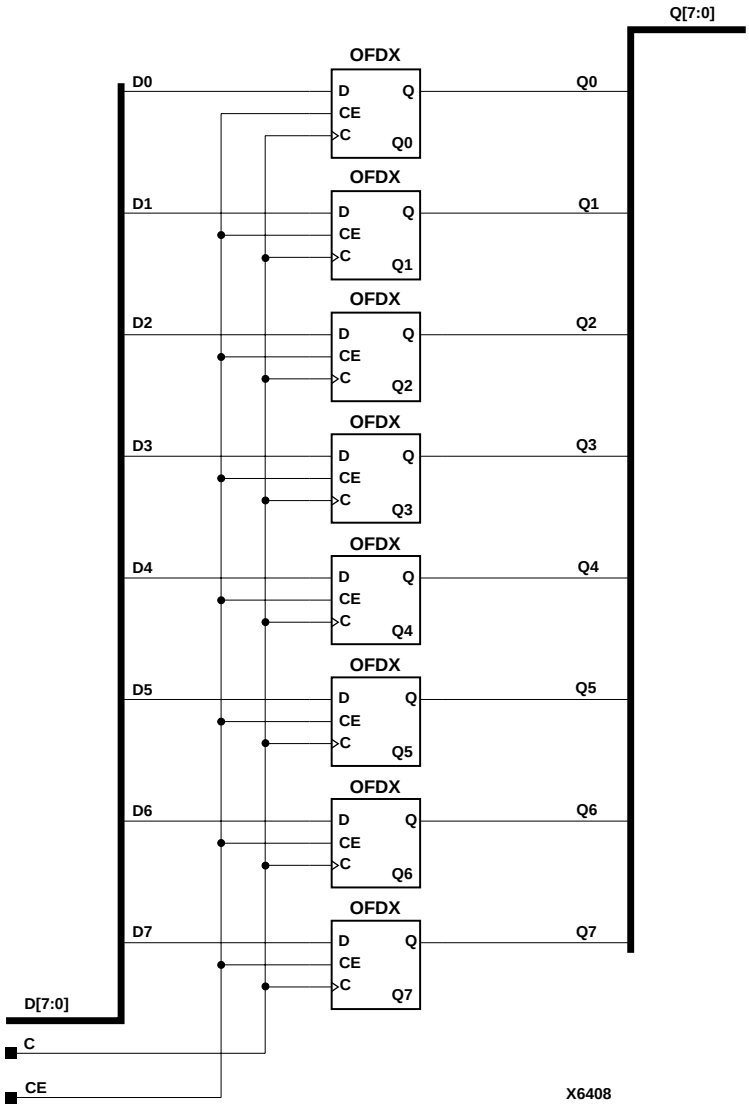
dn = state of referenced input one setup time prior to active clock transition



OFDX Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDX Implementation Virtex-II, Virtex-II Pro



OFDX8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

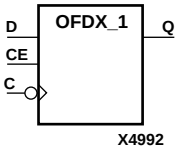
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDX, you would infer an FDCE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFDX_1

Output D Flip-Flop with Inverted Clock and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



OFDX_1 is located in an input / output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output. When the CE pin is Low, the output (Q) does not change.

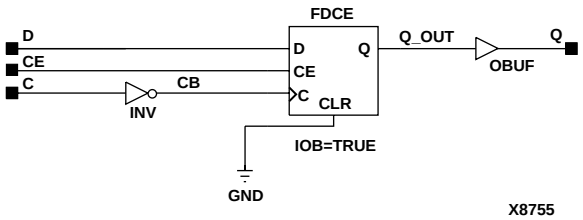
The flip-flop is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

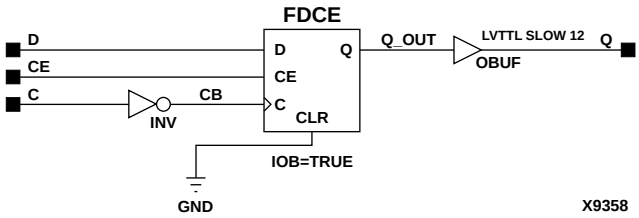
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↓	d
0	X	X	No Chg

d = state of referenced input one setup time prior to active clock transition



OFDX_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDX_1 Implementation Virtex-II, Virtex-II Pro

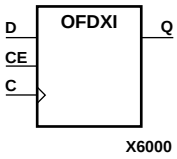
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDX_1, you would infer an FDCE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFDXI

Output D Flip-Flop with Clock Enable (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



OFDXI is contained in an input / output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). When CE is Low, the output does not change.

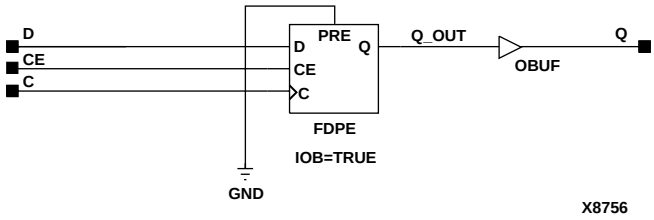
The flip-flop is asynchronously preset with High output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

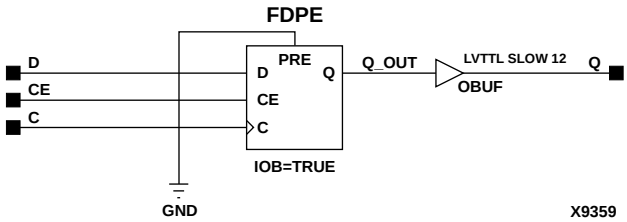
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↑	d
0	X	X	No Chg

d = state of referenced input one setup time prior to active clock transition



OFDXI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDXI Implementation Virtex-II, Virtex-II Pro

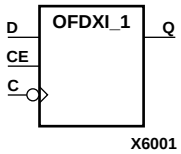
Usage

This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDXI, you would infer an FDPE and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

OFDXI_1

Output D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



OFDXI_1 is located in an input / output block (IOB). The D flip-flop output (Q) is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output. When CE is Low, the output (Q) does not change.

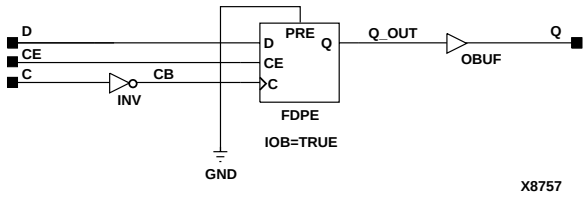
The flip-flop is asynchronously preset with High output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

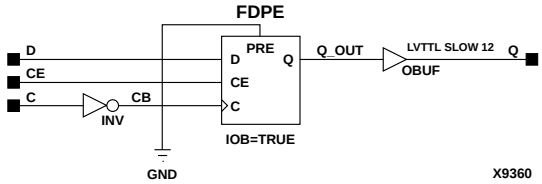
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↓	d
0	X	X	No Chg

d = state of referenced input one setup time prior to active clock transition



OFDXI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDXI_1 Implementation Virtex-II, Virtex-II Pro

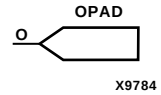
Usage

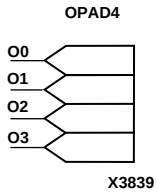
This component is inside of the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDXI_1, you would infer an FDPE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option –pr o to pack all output registers into the IOBs.

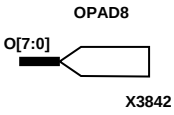
OPAD, 4, 8, 16

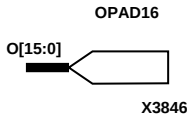
Single- and Multiple-Output Pads

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OPAD	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OPAD4, OPAD8, OPAD16	Macro	Macro	Macro	Macro	Macro	Macro



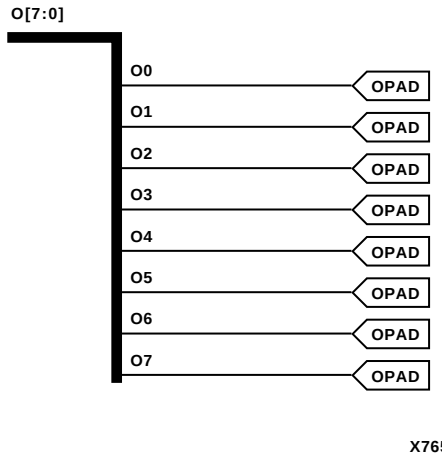






OPAD, OPAD4, OPAD8, and OPAD16 are single and multiple output pads. An OPAD connects a device pin to an output signal of a PLD. It is internally connected to an input/output block (IOB), which is configured by the software as an OBUF, an OBUFT, an OBUFE, an OFD, or an OFDT.

See the appropriate CAE tool interface user guide for details on assigning pin location and identification.



OPAD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, it is not necessary to use these elements in the design. They will be added automatically.

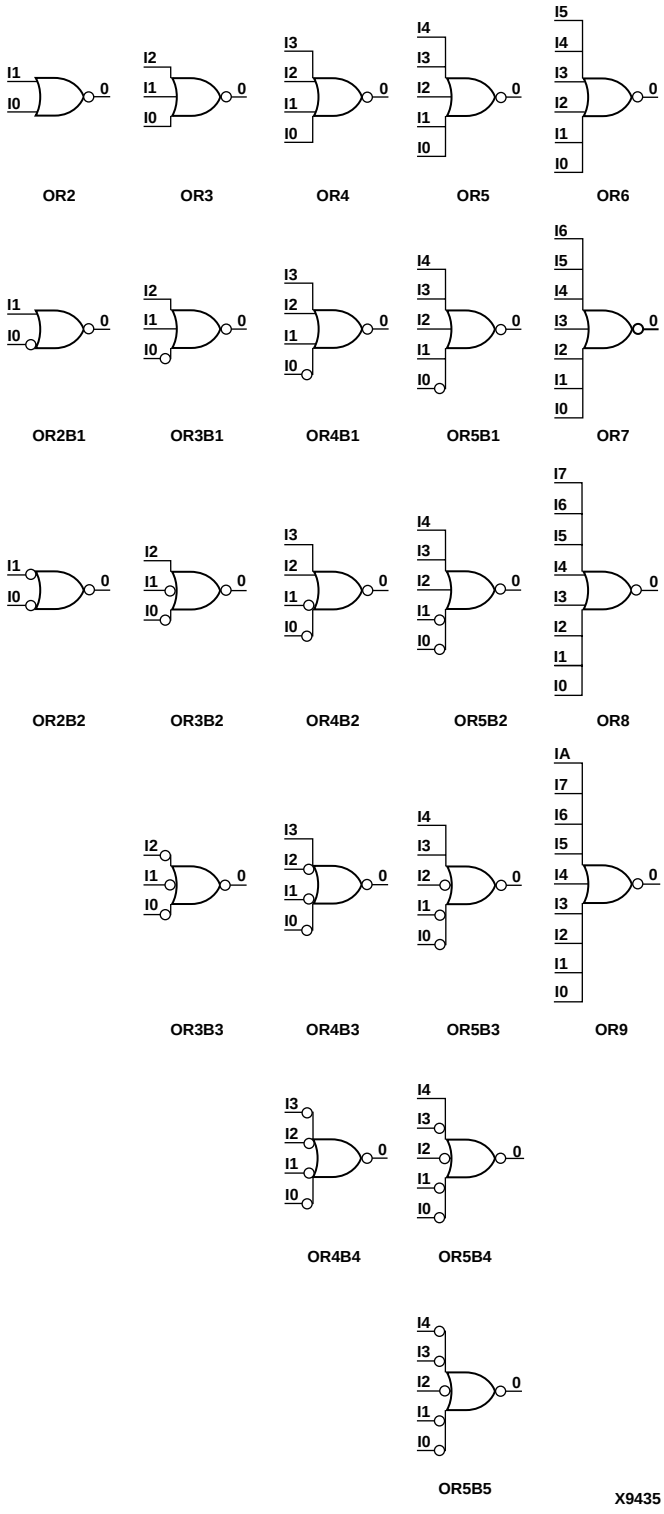
Commonly Used Constraints

IOBDELAY, PULLDOWN

OR2-9

2- to 9-Input OR Gates with Inverted and Non-Inverted Inputs

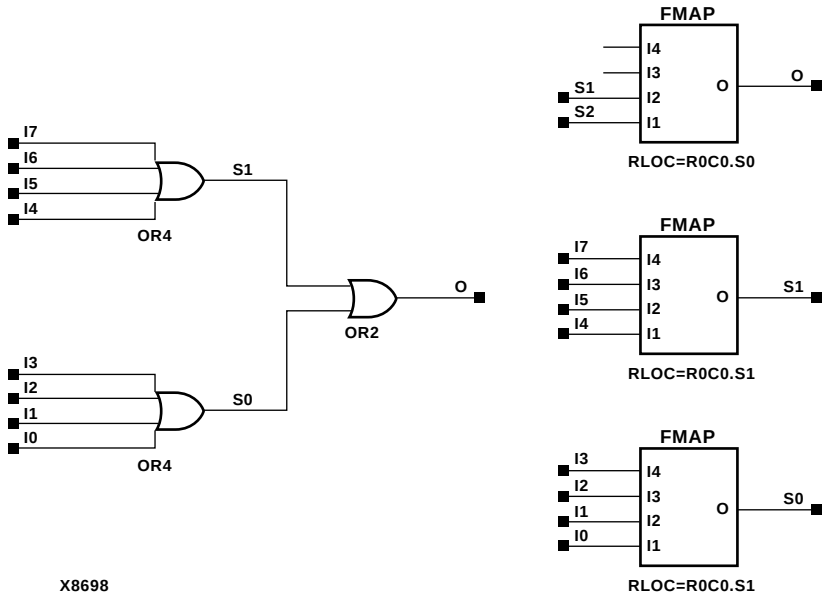
Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
OR2, OR2B1, OR2B2, OR3, OR3B1, OR3B2, OR3B3, OR4, OR4B1, OR4B2, OR4B3, OR4B4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR5, OR5B1, OR5B2, OR5B3, OR5B4, OR5B5	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR6, OR7, OR8, OR9	Macro	Macro	Macro	Primitive	Primitive	Primitive



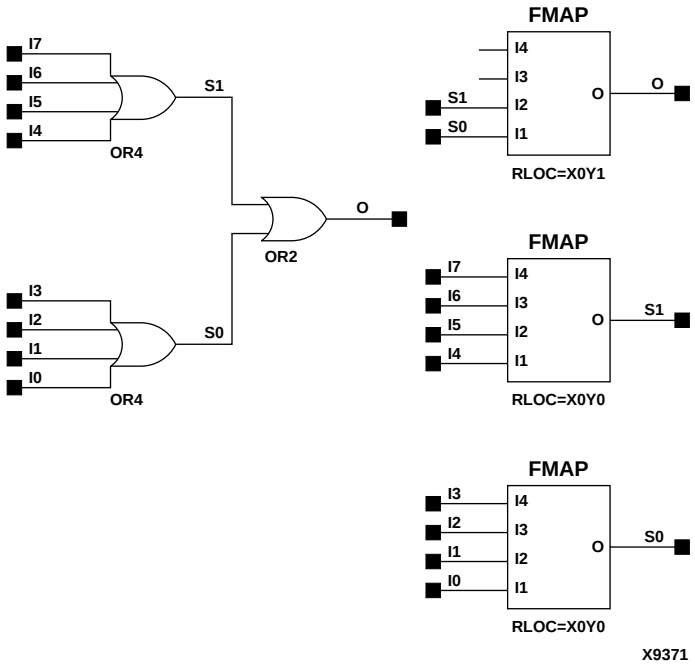
OR Gate Representations

The OR function is performed in the Configurable Logic Block (CLB) function generators for Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro. OR functions of up to five inputs are available in any combination of inverting and non-inverting inputs. OR functions of six to nine inputs are available with only non-inverting inputs. To invert some or all inputs, use external inverters. Since each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.

See the "OR12, 16" section for information on additional OR functions for the Spartan-II, Spartan-IIE, Virtex, and Virtex-E.



OR8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OR8 Implementation Virtex-II, Virtex-II Pro

Usage

OR2 through OR5 are primitives that can be inferred or instantiated. OR6 through OR9 are macros which can be inferred.

VHDL Inference Code

```
architecture Behavioral of or6 is

begin

process (I0, I1, I2, I3, I4, I5)
begin
       O <= (I0 or I1 or I2 or I3 or I4 or I5);
end process;

end Behavioral;
```

Verilog Inference Code (OR6)

```
always @(I0 or I1 or I2 or I3 or I4 or I5)
begin
       O <= (I0 || I1 || I2 || I3 || I4 || I5);
end
```

VHDL Instantiation Template for OR5

Following is the VHDL code for OR5. To instantiate OR2, remove I2, I3, and I4. To instantiate OR3, remove I3 and I4, For OR4, remove I4. OR2B1, and OR2B2 have the same code as OR2. OR3B1, 3B2, and 3B3 have the same code as OR3 and so forth.

```
-- Component Declaration for OR5 should be placed
-- after architecture statement but before begin keyword

component OR5
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         I2 : in STD_ULOGIC;
         I3 : in STD_ULOGIC;
         I4 : in STD_ULOGIC);
end component;

-- Component Attribute specification for OR5
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for OR5 should be placed
-- in architecture after the begin keyword

OR5_INSTANCE_NAME : OR5
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3,
                 I4 => user_I4);
```

Verilog Instantiation Template for OR5

```
OR5 OR5_instance_name (.O (user_O),
                       .I0 (user_I0),
                       .I1 (user_I1),
                       .I2 (user_I2),
                       .I3 (user_I3),
                       .I4 (user_I4));
```

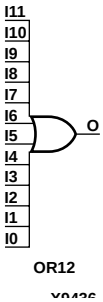
Commonly Used Constraints

None

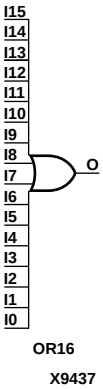
OR12, 16

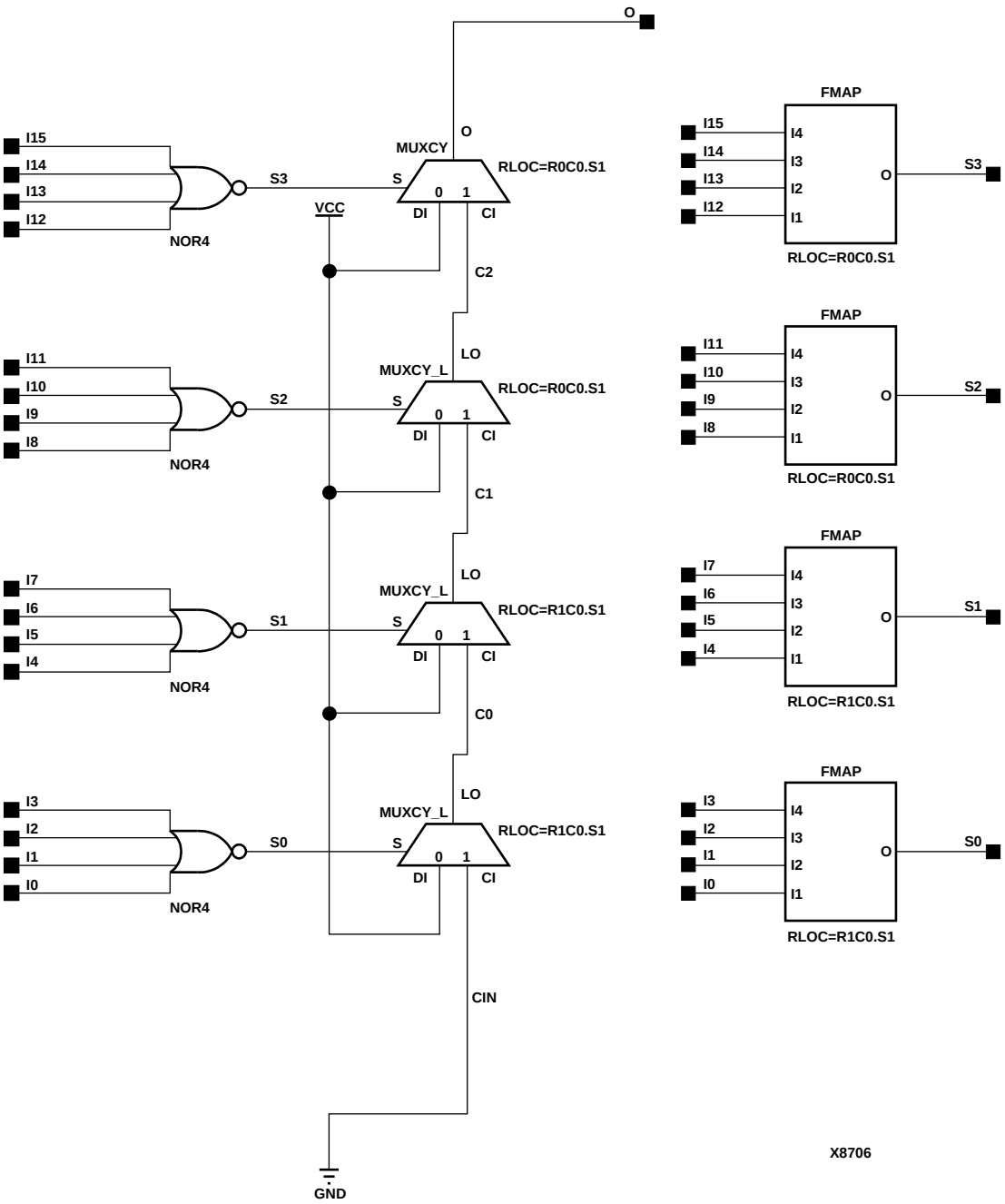
12- and 16-Input OR Gates with Non-Inverted Inputs

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A

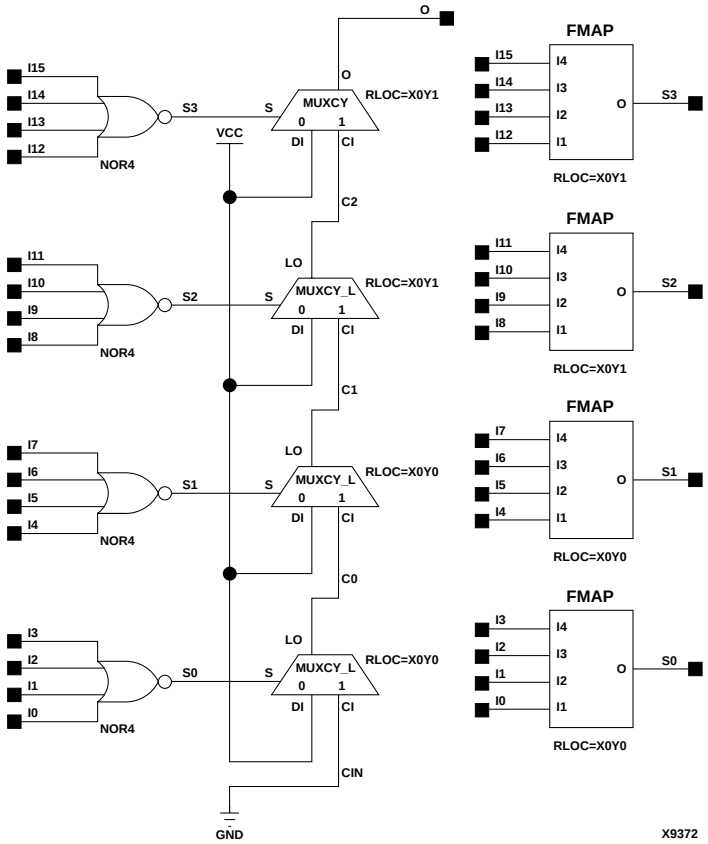


See the "OR2-9" section for information on OR functions.





OR16 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OR16 Implementation Virtex-II, Virtex-II Pro

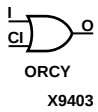
Usage

For HDL, OR12 and OR16 are macros that are inferred.See the "OR2-9" section for information about inferring OR gates.

ORCY

OR with Carry Logic

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



ORCY is a special OR with general O output used for generating faster and smaller arithmetic functions.

Each Virtex-II and Virtex-II Pro slice contains a dedicated 2-input OR gate that ORs together carry out values for a series of horizontally adjacent carry chains. The OR gate gets one input external to the slice and the other input from the output of the high order carry mux. The OR gate's output drives the next slice's OR gate horizontally across the die.

Only MUXCY outputs can drive the signal on the CI pin. Only ORCY outputs or logic zero can drive the I pin.

Usage

For HDL, the ORCY design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for ORCY should be placed
-- after architecture statement but before begin keyword

component ORCY
   port (O  : out STD_ULOGIC;
         CI : in STD_ULOGIC;
         I  : in STD_ULOGIC);
end component;

-- Component Attribute specification for ORCY
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ORCY should be placed
-- in architecture after the begin keyword

ORCY_INSTANCE_NAME : ORCY
      port map  (O => user_O,
                 CI => user_CI,
                 I => user_I);
```

Verilog Instantiation Template

```
ORCY instance_name (.O (user_O),
                    .CI (user_CI),
                    .I (user_I));
```

Commonly Used Constraints

None

PPC405

Primitive for the Power PC Core

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A

*Not supported for Virtex-II. Only supported for Virtex-II Pro.

The PowerPC 405 embedded core is a 32-bit RISC core integrating a PowerPC 405 CPU, separate instruction and data caches, a JTAG port, trace FIFO, multiple timers, and a memory management unit (MMU). Integrated on-chip memory (OCM) controllers provide dedicated interfaces between Block SelectRAM memory and the processor core instruction and data paths for high-speed access. The PowerPC 405 core implements the PowerPC User Instruction Set.

For complete information about the PowerPC 405, see the following documents:

- Virtex-II Pro Datasheet
- Virtex-II Pro Handbook
- The PowerPC 405 Core Processor Block Manual
- The PowerPC 405 User Guide

The following table lists the inputs and outputs of the primitive. For detailed information about the pinouts, see the DS083 Virtex-II Pro Data Sheet.

Inputs	Outputs
BRAMDSOCMCLK	C405CPMCORESLEEPREQ
BRAMDSOCMRDDBUS [0:31]	C405CPMMSRCE
BRAMISOCMCLK	C405CPMMSREE
BRAMISOCMRDDBUS [0:63]	C405CPMTIMERIRQ
CPMC405CLOCK	C405CPMTIMERRESETREQ
CPMC405CORECLKINACTIVE	C405DBGMSRWE
CPMC405CPUCLKEN	C405DBGSTOPACK
CPMC405JTAGCLKEN	C405DBGWBCOMPLETE
CPMC405TIMERCLKEN	C405DBGWBFULL
CPMC405TIMERTICK	C405DBGWBIAR[0:29]
DBGC405DEBUGHALT	C405DCRABUS [0:9]
DBGC405EXTBUSHOLDACK	C405DCRDBUSOUT [0:31]
DBGC405UNCONDDEBUGEVENT	C405DCRREAD
DCRC405ACK	C405DCRWRITE
DCRC405DBUSIN [0:31]	C405JTGCAPTUREDR
DSARCVALUE [0:7]	C405JTGEXTEST
DSCNTLVALUE [0:7]	C405JTGPGMOUT
EICC405CRITINPUTIRQ	C405JTGSHIFTDR

Inputs	Outputs
EICC405EXTINPUTIRQ	C405JTGTDO
ISARCVALUE [0:7]	C405JTGTDOEN
ISCNTLVALUE [0:7]	C405JTGUPDATEDR
JTGC405BNDSCANTDO	C405PLBDCUABORT
JTGC405TCK	C405PLBDCUABUS [0:31]
JTGC405TDI	C405PLBDCUBE [0:7]
JTGC405TMS	C405PLBDCUCACHEABLE
JTGC405TRSTNEG	C405PLBDCUGUARDED
MCBCPUCLKEN	C405PLBDCUPRIORITY [0:1]
MCBJTAGEN	C405PLBDCUREQUEST
MCBTIMEREN	C405PLBDCURNW
MCPPCRST	C405PLBDCUSIZE2
PLBC405DCUADDRACK	C405PLBDCUU0ATTR
PLBC405DCUBUSY	C405PLBDCUWRDBUS [0:63]
PLBC405DCUERR	C405PLBDCUWRITETHRU
PLBC405DCURDDACK	C405PLBICUABORT
PLBC405DCURDDBUS [0:63]	C405PLBICUABUS [0:29]
PLBC405DCURDWDADDR [1:3]	C405PLBICUCACHEABLE
PLBC405DCUSSIZE1	C405PLBICUPRIORITY [0:1]
PLBC405DCUWRDACK	C405PLBICUREQUEST
PLBC405ICUADDRACK	C405PLBICUSIZE [2:3]
PLBC405ICUBUSY	C405PLBICUU0ATTR
PLBC405ICUERR	C405RSTCHIPRESETREQ
PLBC405ICURDDACK	C405RSTCORERESETREQ
PLBC405ICURDDBUS [0:63]	C405RSTSYSRESETREQ
PLBC405ICURDWDADDR [1:3]	C405TRCCYCLE
PLBC405ICUSSIZE1	C405TRCEVENEXECUTIONSTATUS [0:1]
PLBCLK	C405TRCODDEXECUTIONSTATUS [0:1]
RSTC405RESETCHIP	C405TRCTRACESTATUS [0:3]
RSTC405RESETCORE	C405TRCTRIGGEREVENTOUT
RSTC405RESETSYS	C405TRCTRIGGEREVENTTYPE [0:10]
TIEC405DETERMINISTICMULT	C405XXXMACHINECHECK
TIEC405DISOPERANDFWD	DSOCMBRAMABUS [8:29]
TIEC405MMUEN	DSOCMBRAMBYTEWRITE [0:3]
TIEDSOCMDCRADDR [0:7]	DSOCMBRAMEN
TIEISOCMDCRADDR [0:7]	DSOCMBRAMWRDBUS [0:31]
TRCC405TRACEDISABLE	DSOCMBUSY
TRCC405TRIGGEREVENTIN	ISOCMBRAMEN
	ISOCMBRAMEVENWRITEEN
	ISOCMBRAMODDWRITEEN
	ISOCMBRAMRDABUS [8:28]

Inputs	Outputs
	ISOCMBRAMWRABUS [8:28]
	ISOCMBRAMWRDBUS [0:31]

Usage

For HDL, the PPC405 design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for PPC405 should be placed
-- after architecture statement but before begin keyword

component PPC405

  port(C405CPMCORESLEEPREQ      : out STD_ULOGIC;
       C405CPMMSRCE             : out STD_ULOGIC;
       C405CPMMSREE             : out STD_ULOGIC;
       C405CPMTIMERIRQ          : out STD_ULOGIC;
       C405CPMTIMERRESETREQ     : out STD_ULOGIC;
       C405DBGMSRWE             : out STD_ULOGIC;
       C405DBGSTOPACK           : out STD_ULOGIC;
       C405DBGWBCOMPLETE        : out STD_ULOGIC;
       C405DBGWBFULL            : out STD_ULOGIC;
       C405DBGWBIAR             : out STD_LOGIC_VECTOR (29 downto 0);
       C405DCRABUS              : out STD_LOGIC_VECTOR (9 downto 0);
       C405DCRDBUSOUT           : out STD_LOGIC_VECTOR (31 downto 0);
       C405DCRREAD              : out STD_ULOGIC;
       C405DCRWRITE             : out STD_ULOGIC;
       C405JTGCAPTUREDR         : out STD_ULOGIC;
       C405JTGEXTEST            : out STD_ULOGIC;
       C405JTGPGMOUT            : out STD_ULOGIC;
       C405JTGSHIFTDR           : out STD_ULOGIC;
       C405JTGTDO               : out STD_ULOGIC;
       C405JTGTDOEN             : out STD_ULOGIC;
       C405JTGUPDATEDR          : out STD_ULOGIC;
       C405PLBDCUABORT          : out STD_ULOGIC;
       C405PLBDCUABUS           : out STD_LOGIC_VECTOR (31 downto 0);
       C405PLBDCUBE             : out STD_LOGIC_VECTOR (7 downto 0);
       C405PLBDCUCACHEABLE      : out STD_ULOGIC;
       C405PLBDCUGUARDED        : out STD_ULOGIC;
       C405PLBDCUPRIORITY       : out STD_LOGIC_VECTOR (1 downto 0);
       C405PLBDCUREQUEST        : out STD_ULOGIC;
       C405PLBDCURNW            : out STD_ULOGIC;
       C405PLBDCUSIZE2          : out STD_ULOGIC;
       C405PLBDCUU0ATTR         : out STD_ULOGIC;
       C405PLBDCUWRDBUS         : out STD_LOGIC_VECTOR (63 downto 0);
       C405PLBDCUWRITETHRU      : out STD_ULOGIC;
       C405PLBICUABORT          : out STD_ULOGIC;
       C405PLBICUABUS           : out STD_LOGIC_VECTOR (29 downto 0);
       C405PLBICUCACHEABLE      : out STD_ULOGIC;
       C405PLBICUPRIORITY       : out STD_LOGIC_VECTOR (1 downto 0);
```

```
C405PLBICUREQUEST        : out STD_ULOGIC;
C405PLBICUSIZE           : out STD_LOGIC_VECTOR (3 downto 2);
C405PLBICUU0ATTR         : out STD_ULOGIC;
C405RSTCHIPRESETREQ      : out STD_ULOGIC;
C405RSTCORERESETREQ      : out STD_ULOGIC;
C405RSTSYSRESETREQ       : out STD_ULOGIC;
C405TRCCYCLE             : out STD_ULOGIC;
C405TRCEVENEXECUTIONSTATUS: out STD_LOGIC_VECTOR (1 downto 0);
C405TRCODDEXECUTIONSTATUS: out STD_LOGIC_VECTOR (1 downto 0);
C405TRCTRACESTATUS       : out STD_LOGIC_VECTOR (3 downto 0);
C405TRCTRIGGEREVENTOUT   : out STD_ULOGIC;
C405TRCTRIGGEREVENTTYPE  : out STD_LOGIC_VECTOR (10 downto 0);
C405XXXMACHINECHECK      : out STD_ULOGIC;
DSOCMBRAMABUS            : out STD_LOGIC_VECTOR (29 downto 8);
DSOCMBRAMBYTEWRITE       : out STD_LOGIC_VECTOR (3 downto 0);
DSOCMBRAMEN              : out STD_ULOGIC;
DSOCMBRAMWRDBUS          : out STD_LOGIC_VECTOR (31 downto 0);
DSOCMBUSY                : out STD_ULOGIC;
ISOCMBRAMEN              : out STD_ULOGIC;
ISOCMBRAMEVENWRITEEN     : out STD_ULOGIC;
ISOCMBRAMODDWRITEEN      : out STD_ULOGIC;
ISOCMBRAMRDABUS          : out STD_LOGIC_VECTOR (28 downto 8);
ISOCMBRAMWRABUS          : out STD_LOGIC_VECTOR (28 downto 8);
ISOCMBRAMWRDBUS          : out STD_LOGIC_VECTOR (31 downto 0);
BRAMDSOCMCLK             : in STD_ULOGIC;
BRAMDSOCMRDDBUS          : in STD_LOGIC_VECTOR (31 downto 0);
BRAMISOCMCLK             : in STD_ULOGIC;
BRAMISOCMRDDBUS          : in STD_LOGIC_VECTOR (63 downto 0);
CPMC405CLOCK             : in STD_ULOGIC;
CPMC405CORECLKINACTIVE   : in STD_ULOGIC;
CPMC405CPUCLKEN          : in STD_ULOGIC;
CPMC405JTAGCLKEN         : in STD_ULOGIC;
CPMC405TIMERCLKEN        : in STD_ULOGIC;
CPMC405TIMERTICK         : in STD_ULOGIC;
DBGC405DEBUGHALT         : in STD_ULOGIC;
DBGC405EXTBUSHOLDACK     : in STD_ULOGIC;
DBGC405UNCONDDEBUGEVENT  : in STD_ULOGIC;
DCRC405ACK               : in STD_ULOGIC;
DCRC405DBUSIN            : in STD_LOGIC_VECTOR (31 downto 0);
DSARCVALUE               : in STD_LOGIC_VECTOR (7 downto 0);
DSCNTLVALUE              : in STD_LOGIC_VECTOR (7 downto 0);
EICC405CRITINPUTIRQ      : in STD_ULOGIC;
EICC405EXTINPUTIRQ       : in STD_ULOGIC;
ISARCVALUE               : in STD_LOGIC_VECTOR (7 downto 0);
ISCNTLVALUE              : in STD_LOGIC_VECTOR (7 downto 0);
JTGC405BNDSCANTDO        : in STD_ULOGIC;
JTGC405TCK               : in STD_ULOGIC;
JTGC405TDI               : in STD_ULOGIC;
JTGC405TMS               : in STD_ULOGIC;
JTGC405TRSTNEG           : in STD_ULOGIC;
MCBCPUCLKEN              : in STD_ULOGIC;
MCBJTAGEN                : in STD_ULOGIC;
MCBTIMEREN               : in STD_ULOGIC;
MCPPCRST                 : in STD_ULOGIC;
```

```
       PLBC405DCUADDRACK        : in STD_ULOGIC;
       PLBC405DCUBUSY           : in STD_ULOGIC;
       PLBC405DCUERR            : in STD_ULOGIC;
       PLBC405DCURDDACK         : in STD_ULOGIC;
       PLBC405DCURDDBUS         : in STD_LOGIC_VECTOR (63 downto 0);
       PLBC405DCURDWDADDR       : in STD_LOGIC_VECTOR (3 downto 1);
       PLBC405DCUSSIZE1         : in STD_ULOGIC;
       PLBC405DCUWRDACK         : in STD_ULOGIC;
       PLBC405ICUADDRACK        : in STD_ULOGIC;
       PLBC405ICUBUSY           : in STD_ULOGIC;
       PLBC405ICUERR            : in STD_ULOGIC;
       PLBC405ICURDDACK         : in STD_ULOGIC;
       PLBC405ICURDDBUS         : in STD_LOGIC_VECTOR (63 downto 0);
       PLBC405ICURDWDADDR       : in STD_LOGIC_VECTOR (3 downto 1);
       PLBC405ICUSSIZE1         : in STD_ULOGIC;
       PLBCLK                   : in STD_ULOGIC;
       RSTC405RESETCHIP         : in STD_ULOGIC;
       RSTC405RESETCORE         : in STD_ULOGIC;
       RSTC405RESETSYS          : in STD_ULOGIC;
       TIEC405DETERMINISTICMULT : in STD_ULOGIC;
       TIEC405DISOPERANDFWD     : in STD_ULOGIC;
       TIEC405MMUEN             : in STD_ULOGIC;
       TIEDSOCMDCRADDR          : in STD_LOGIC_VECTOR (7 downto 0);
       TIEISOCMDCRADDR          : in STD_LOGIC_VECTOR (7 downto 0);
       TRCC405TRACEDISABLE      : in STD_ULOGIC;
       TRCC405TRIGGEREVENTINE   : in STD_ULOGIC);
end component;

-- Component Attribute specification for PPC405
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for PPC405 should be placed
-- in architecture after the begin keyword

PPC405_INSTANCE_NAME : PPC405
    port map(C405CPMCORESLEEPREQ  => user_C405CPMCORESLEEPREQ,
             C405CPMMSRCE         => user_C405CPMMSRCE,
             C405CPMMSREE         => user_C405CPMMSREE,
             C405CPMTIMERIRQ      => user_C405CPMTIMERIRQ,
             C405CPMTIMERRESETREQ => user_ C405CPMTIMERRESETREQ,
             C405DBGMSRWE         => user_C405DBGMSRWE,
             C405DBGSTOPACK       => user_C405DBGSTOPACK,
             C405DBGWBCOMPLETE  => user_C405DBGWBCOMPLETE,
             C405DBGWBFULL      => user_C405DBGWBFULL,
             C405DBGWBIAR       => user_C405DBGWBIAR,
             C405DCRABUS        => user_C405DCRABUS,
             C405DCRDBUSOUT     => user_C405DCRDBUSOUT,
             C405DCRREAD        => user_C405DCRREAD,
             C405DCRWRITE       => user_C405DCRWRITE,
             C405JTGCAPTUREDR   => user_C405JTGCAPTUREDR,
```

```
C405JTGEXTEST       => user_C405JTGEXTEST,
C405JTGPGMOUT       => user_C405JTGPGMOUT,
C405JTGSHIFTDR      => user_C405JTGSHIFTDR,
C405JTGTDO          => user_C405JTGTDO,
C405JTGTDOEN        => user_C405JTGTDOEN,
C405JTGUPDATEDR     => user_C405JTGUPDATEDR,
C405PLBDCUABORT     => user_C405PLBDCUABORT,
C405PLBDCUABUS      => user_C405PLBDCUABUS,
C405PLBDCUBE        => user_C405PLBDCUBE,
C405PLBDCUCACHEABLE => user_C405PLBDCUCACHEABLE,
C405PLBDCUGUARDED   => user_C405PLBDCUGUARDED,
C405PLBDCUPRIORITY  => user_C405PLBDCUPRIORITY,
C405PLBDCUREQUEST   => user_C405PLBDCUREQUEST,
C405PLBDCURNW       => user_C405PLBDCURNW,
C405PLBDCUSIZE2     => user_C405PLBDCUSIZE2,
C405PLBDCUU0ATTR    => user_C405PLBDCUU0ATTR,
C405PLBDCUWRDBUS    => user_C405PLBDCUWRDBUS,
C405PLBDCUWRITETHRU => user_C405PLBDCUWRITETHRU,
C405PLBICUABORT     => user_C405PLBICUABORT,
C405PLBICUABUS      => user_C405PLBICUABUS,
C405PLBICUCACHEABLE => user_C405PLBICUCACHEABLE,
C405PLBICUPRIORITY  => user_C405PLBICUPRIORITY,
C405PLBICUREQUEST   => user_C405PLBICUREQUEST,
C405PLBICUSIZE      => user_C405PLBICUSIZE,
C405PLBICUU0ATTR    => user_C405PLBICUU0ATTR,
C405RSTCHIPRESETREQ => user_C405RSTCHIPRESETREQ,
C405RSTCORERESETREQ => user_C405RSTCORERESETREQ,
C405RSTSYSRESETREQ  => user_C405RSTSYSRESETREQ,
C405TRCCYCLE        => user_C405TRCCYCLE,
C405TRCEVENEXECUTIONSTATUS=> user_C405TRCEVENEXECUTIONSTATUS,
C405TRCODDEXECUTIONSTATUS => user_C405TRCODDEXECUTIONSTATUS,
C405TRCTRACESTATUS        => user_C405TRCTRACESTATUS,
C405TRCTRIGGEREVENTOUT    => user_C405TRCTRIGGEREVENTOUT,
C405TRCTRIGGEREVENTTYPE   => user_C405TRCTRIGGEREVENTTYPE,
C405XXXMACHINECHECK       => user_C405XXXMACHINECHECK,
DSOCMBRAMABUS             => user_DSOCMBRAMABUS,
DSOCMBRAMBYTEWRITE        => user_DSOCMBRAMBYTEWRITE,
DSOCMBRAMEN               => user_DSOCMBRAMEN,
DSOCMBRAMWRDBUS           => user_DSOCMBRAMWRDBUS,
DSOCMBUSY                 => user_DSOCMBUSY,
ISOCMBRAMEN               => user_ISOCMBRAMEN,
ISOCMBRAMEVENWRITEEN      => user_ISOCMBRAMEVENWRITEEN,
ISOCMBRAMODDWRITEEN       => user_ISOCMBRAMODDWRITEEN,
ISOCMBRAMRDABUS           => user_ISOCMBRAMRDABUS,
ISOCMBRAMWRABUS           => user_ISOCMBRAMWRABUS,
ISOCMBRAMWRDBUS           => user_ISOCMBRAMWRDBUS,
BRAMDSOCMCLK               => user_BRAMDSOCMCLK,
BRAMDSOCMRDDBUS            => user_BRAMDSOCMRDDBUS,
BRAMISOCMCLK               => user_BRAMISOCMCLK,
BRAMISOCMRDDBUS            => user_BRAMISOCMRDDBUS,
CPMC405CLOCK               => user_CPMC405CLOCK,
CPMC405CORECLKINACTIVE     => user_CPMC405CORECLKINACTIVE,
CPMC405CPUCLKEN            => user_CPMC405CPUCLKEN,
CPMC405JTAGCLKEN           => user_CPMC405JTAGCLKEN,
```

```
CPMC405TIMERCLKEN          => user_CPMC405TIMERCLKEN,
CPMC405TIMERTICK           => user_CPMC405TIMERTICK,
DBGC405DEBUGHALT           => user_DBGC405DEBUGHALT,
DBGC405EXTBUSHOLDACK       => user_DBGC405EXTBUSHOLDACK,
DBGC405UNCONDDEBUGEVENT    => user_DBGC405UNCONDDEBUGEVENT,
DCRC405ACK                 => user_DCRC405ACK,
DCRC405DBUSIN              => user_DCRC405DBUSIN,
DSARCVALUE                 => user_DSARCVALUE,
DSCNTLVALUE                => user_DSCNTLVALUE,
EICC405CRITINPUTIRQ        => user_EICC405CRITINPUTIRQ,
EICC405EXTINPUTIRQ         => user_EICC405EXTINPUTIRQ,
ISARCVALUE                 => user_ISARCVALUE,
ISCNTLVALUE                => user_ISCNTLVALUE,
JTGC405BNDSCANTDO          => user_JTGC405BNDSCANTDO,
JTGC405TCK                 => user_JTGC405TCK,
JTGC405TDI                 => user_JTGC405TDI,
JTGC405TMS                 => user_JTGC405TMS,
JTGC405TRSTNEG             => user_JTGC405TRSTNEG,
MCBCPUCLKEN                => user_MCBCPUCLKEN,
MCBJTAGEN                  => user_MCBJTAGEN,
MCBTIMEREN                 => user_MCBTIMEREN,
MCPPCRST                   => user_MCPPCRST,
PLBC405DCUADDRACK          => user_PLBC405DCUADDRACK,
PLBC405DCUBUSY             => user_PLBC405DCUBUSY,
PLBC405DCUERR              => user_PLBC405DCUERR,
PLBC405DCURDDACK           => user_PLBC405DCURDDACK,
PLBC405DCURDDBUS           => user_PLBC405DCURDDBUS,
PLBC405DCURDWDADDR         => user_PLBC405DCURDWDADDR,
PLBC405DCUSSIZE1           => user_PLBC405DCUSSIZE1,
PLBC405DCUWRDACK           => user_PLBC405DCUWRDACK,
PLBC405ICUADDRACK          => user_PLBC405ICUADDRACK,
PLBC405ICUBUSY             => user_PLBC405ICUBUSY,
PLBC405ICUERR              => user_PLBC405ICUERR,
PLBC405ICURDDACK           => user_PLBC405ICURDDACK,
PLBC405ICURDDBUS           => user_PLBC405ICURDDBUS,
PLBC405ICURDWDADDR         => user_PLBC405ICURDWDADDR,
PLBC405ICUSSIZE1           => user_PLBC405ICUSSIZE1,
PLBCLK                     => user_PLBCLK,
RSTC405RESETCHIP           => user_RSTC405RESETCHIP,
RSTC405RESETCORE           => user_RSTC405RESETCORE,
RSTC405RESETSYS            => user_RSTC405RESETSYS,
TIEC405DETERMINISTICMULT   => user_TIEC405DETERMINISTICMULT,
TIEC405DISOPERANDFWD       => user_TIEC405DISOPERANDFWD,
TIEC405MMUEN               => user_TIEC405MMUEN,
TIEDSOCMDCRADDR            => user_TIEDSOCMDCRADDR,
TIEISOCMDCRADDR            => user_TIEISOCMDCRADDR,
TRCC405TRACEDISABLE         => user_TRCC405TRACEDISABLE,
TRCC405TRIGGEREVENTINE      => user_TRCC405TRIGGEREVENTINE);
```

Verilog Instantiation Template

```
PPC405 instance_name (.C405CPMCORESLEEPREQ(user_C405CPMCORESLEEPREQ),
                      .C405CPMMSRCE(user_C405CPMMSRCE),
                      .C405CPMMSREE(user_C405CPMMSREE),
                      .C405CPMTIMERIRQ (user_C405CPMTIMERIRQ),
                      .C405CPMTIMERRESETREQ (user_ C405CPMTIMERRESETREQ),
                      .C405DBGMSRWE (user_C405DBGMSRWE),
                      .C405DBGSTOPACK (user_C405DBGSTOPACK),
                      .C405DBGWBCOMPLETE (user_C405DBGWBCOMPLETE),
                      .C405DBGWBFULL (user_C405DBGWBFULL),
                      .C405DBGWBIAR (user_C405DBGWBIAR),
                      .C405DCRABUS(user_C405DCRABUS),
                      .C405DCRDBUSOUT (user_C405DCRDBUSOUT),
                      .C405DCRREAD (user_C405DCRREAD),
                      .C405DCRWRITE (user_C405DCRWRITE),
                      .C405JTGCAPTUREDR (user_C405JTGCAPTUREDR),
                      .C405JTGEXTEST (user_C405JTGEXTEST),
                      .C405JTGPGMOUT (user_C405JTGPGMOUT),
                      .C405JTGSHIFTDR (user_C405JTGSHIFTDR),
                      .C405JTGTDO (user_C405JTGTDO),
                      .C405JTGTDOEN (user_C405JTGTDOEN),
                      .C405JTGUPDATEDR (user_C405JTGUPDATEDR),
                      .C405PLBDCUABORT (user_C405PLBDCUABORT),
                      .C405PLBDCUABUS (user_C405PLBDCUABUS),
                      .C405PLBDCUBE (user_C405PLBDCUBE),
                      .C405PLBDCUCACHEABLE (user_C405PLBDCUCACHEABLE),
                      .C405PLBDCUGUARDED (user_C405PLBDCUGUARDED),
                      .C405PLBDCUPRIORITY (user_C405PLBDCUPRIORITY),
                      .C405PLBDCUREQUEST (user_C405PLBDCUREQUEST),
                      .C405PLBDCURNW (user_C405PLBDCURNW),
                      .C405PLBDCUSIZE2 (user_C405PLBDCUSIZE2),
                      .C405PLBDCUU0ATTR (user_C405PLBDCUU0ATTR),
                      .C405PLBDCUWRDBUS (user_C405PLBDCUWRDBUS),
                      .C405PLBDCUWRITETHRU (user_C405PLBDCUWRITETHRU),
                      .C405PLBICUABORT (user_C405PLBICUABORT),
                      .C405PLBICUABUS (user_C405PLBICUABUS),
                      .C405PLBICUCACHEABLE (user_C405PLBICUCACHEABLE),
                      .C405PLBICUPRIORITY (user_C405PLBICUPRIORITY),
                      .C405PLBICUREQUEST (user_C405PLBICUREQUEST),
                      .C405PLBICUSIZE (user_C405PLBICUSIZE),
                      .C405PLBICUU0ATTR (user_C405PLBICUU0ATTR),
                      .C405RSTCHIPRESETREQ (user_C405RSTCHIPRESETREQ),
                      .C405RSTCORERESETREQ (user_C405RSTCORERESETREQ),
                      .C405RSTSYSRESETREQ (user_C405RSTSYSRESETREQ),
                      .C405TRCCYCLE (user_C405TRCCYCLE),
                      .C405TRCEVENEXECUTIONSTATUS (user_C405TRCEVENEXECUTIONSTATUS),
                      .C405TRCODDEXECUTIONSTATUS (user_C405TRCODDEXECUTIONSTATUS),
                      .C405TRCTRACESTATUS (user_C405TRCTRACESTATUS),
                      .C405TRCTRIGGEREVENTOUT (user_C405TRCTRIGGEREVENTOUT),
                      .C405TRCTRIGGEREVENTTYPE (user_C405TRCTRIGGEREVENTTYPE),
                      .C405XXXMACHINECHECK (user_C405XXXMACHINECHECK),
                      .DSOCMBRAMABUS (user_DSOCMBRAMABUS),
                      .DSOCMBRAMBYTEWRITE (user_DSOCMBRAMBYTEWRITE),
```

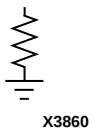
```
.DSOCMBRAMEN (user_DSOCMBRAMEN),
.DSOCMBRAMWRDBUS(user_DSOCMBRAMWRDBUS),
.DSOCMBUSY (user_DSOCMBUSY),
.ISOCMBRAMEN (user_ISOCMBRAMEN),
.ISOCMBRAMEVENWRITEEN (user_ISOCMBRAMEVENWRITEEN),
.ISOCMBRAMODDWRITEEN (user_ISOCMBRAMODDWRITEEN),
.ISOCMBRAMRDABUS (user_ISOCMBRAMRDABUS),
.ISOCMBRAMWRABUS (user_ISOCMBRAMWRABUS),
.ISOCMBRAMWRDBUS (user_ISOCMBRAMWRDBUS),
.BRAMDSOCMCLK (user_BRAMDSOCMCLK),
.BRAMDSOCMRDDBUS (user_BRAMDSOCMRDDBUS),
.BRAMISOCMCLK (user_BRAMISOCMCLK),
.BRAMISOCMRDDBUS (user_BRAMISOCMRDDBUS),
.CPMC405CLOCK (user_CPMC405CLOCK),
.CPMC405CORECLKINACTIVE (user_CPMC405CORECLKINACTIVE),
.CPMC405CPUCLKEN (user_CPMC405CPUCLKEN),
.CPMC405JTAGCLKEN (user_CPMC405JTAGCLKEN),
.CPMC405TIMERCLKEN (user_CPMC405TIMERCLKEN),
.CPMC405TIMERTICK (user_CPMC405TIMERTICK),
.DBGC405DEBUGHALT (user_DBGC405DEBUGHALT),
.DBGC405EXTBUSHOLDACK (user_DBGC405EXTBUSHOLDACK),
.DBGC405UNCONDDEBUGEVENT (user_DBGC405UNCONDDEBUGEVENT),
.DCRC405ACK (user_DCRC405ACK),
.DCRC405DBUSIN (user_DSARCVALUE),
.DSCNTLVALUE (user_DSCNTLVALUE),
.EICC405CRITINPUTIRQ (user_EICC405CRITINPUTIRQ),
.EICC405EXTINPUTIRQ (user_EICC405EXTINPUTIRQ),
.ISARCVALUE (user_ISARCVALUE),
.ISCNTLVALUE (user_ISCNTLVALUE),
.JTGC405BNDSCANTDO (user_JTGC405BNDSCANTDO),
.JTGC405TCK (user_JTGC405TCK),
.JTGC405TDI (user_JTGC405TDI),
.JTGC405TMS (user_JTGC405TMS),
.JTGC405TRSTNEG (user_JTGC405TRSTNEG),
.MCBCPUCLKEN (user_MCBCPUCLKEN),
.MCBJTAGEN (user_MCBJTAGEN),
.MCBTIMEREN (user_MCBTIMEREN),
.MCPPCRST (user_MCPPCRST),
.PLBC405DCUADDRACK (user_PLBC405DCUADDRACK),
.PLBC405DCUBUSY (user_PLBC405DCUBUSY),
.PLBC405DCUERR (user_PLBC405DCUERR),
.PLBC405DCURDDACK (user_PLBC405DCURDDACK),
.PLBC405DCURDDBUS (user_PLBC405DCURDDBUS),
.PLBC405DCURDWDADDR (user_PLBC405DCURDWDADDR),
.PLBC405DCUSSIZE1 (user_PLBC405DCUSSIZE1),
.PLBC405DCUWRDACK (user_PLBC405DCUWRDACK),
.PLBC405ICUADDRACK (user_PLBC405ICUADDRACK),
.PLBC405ICUBUSY (user_PLBC405ICUBUSY),
.PLBC405ICUERR (user_PLBC405ICUERR),
.PLBC405ICURDDACK (user_PLBC405ICURDDACK),
.PLBC405ICURDDBUS (user_PLBC405ICURDDBUS),
.PLBC405ICURDWDADDR (user_PLBC405ICURDWDADDR),
.PLBC405ICUSSIZE1 (user_PLBC405ICUSSIZE1),
.PLBCLK (user_PLBCLK),
```

```
.RSTC405RESETCHIP (user_RSTC405RESETCHIP),
.RSTC405RESETCORE (user_RSTC405RESETCORE),
.RSTC405RESETSYS (user_RSTC405RESETSYS),
.TIEC405DETERMINISTICMULT (user_TIEC405DETERMINISTICMULT),
.TIEC405DISOPERANDFWD (user_TIEC405DISOPERANDFWD),
.TIEC405MMUEN (user_TIEC405MMUEN),
.TIEDSOCMDCRADDR (user_TIEDSOCMDCRADDR),
.TIEISOCMDCRADDR (user_TIEISOCMDCRADDR),
.TRCC405TRACEDISABLE (user_TRCC405TRACEDISABLE),
.TRCC405TRIGGEREVENTINE (user_TRCC405TRIGGEREVENTINE));
```

PULLDOWN

Resistor to GND for Input Pads

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	Primitive



PULLDOWN resistor elements are connected to input, output, or bidirectional pads to guarantee a logic Low level for nodes that might float.

Usage

For HDL, the PULLDOWN design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for PULLDOWN should be placed
-- after architecture statement but before begin keyword

component PULLDOWN
   port (O  : out STD_ULOGIC);
end component;

-- Component Attribute specification for PULLDOWN
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for PULLDOWN should be placed
-- in architecture after the begin keyword

PULLDOWN_INSTANCE_NAME : PULLDOWN
      port map  (O => user_O);
```

Verilog Instantiation Template

```
PULLDOWN instance_name (.O (user_O));
```

Commonly Used Constraints

None

PULLUP

Resistor to VCC for Input PADs, Open-Drain, and 3-State Outputs

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	Primitive

X3861

The pull-up elements establish a High logic level for open-drain elements and macros (DECODE, WAND, WORAND) or 3-state nodes (TBUF) when all the drivers are off.

The buffer outputs are connected together as a wired-AND to form the output (O). When all the inputs are High, the output is off. To establish an output High level, a PULLUP resistor(s) is tied to output (O). One PULLUP resistor uses the least power, two pull-up resistors achieve the fastest Low-to-High speed.

To indicate two PULLUP resistors, append a DOUBLE parameter to the pull-up symbol attached to the output (O) node. See the appropriate CAE tool interface user guide for details.

The PULLUP element is ignored in XC9500/XV/XL designs. Internal 3-state nodes (from BUFE or BUFT) in XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II are always pulled up when not driven.

Usage

For HDL, the PULLUP design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for PULLUP should be placed
-- after architecture statement but before begin keyword

component PULLUP
   port (O  : out STD_ULOGIC);
end component;

-- Component Attribute specification for PULLUP
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for PULLUP should be placed
-- in architecture after the begin keyword

PULLUP_INSTANCE_NAME : PULLUP
      port map  (O => user_O);
```

Verilog Instantiation Template

```
PULLUP instance_name (.O (user_O));
```

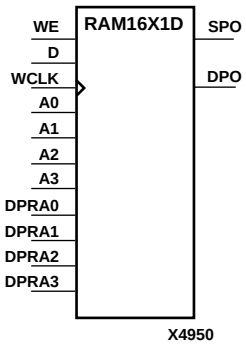
Commonly Used Constraints

DOUBLE, HBLKNM

RAM16X1D

16-Deep by 1-Wide Static Dual Port Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



RAM16X1D is a 16-word by 1-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D	D	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note The write process is not affected by the address on the read address port.

Specifying Initial Contents of a RAM

You can use the INIT attribute to specify an initial value directly on the symbol if the RAM is 1 bit wide and 16, 32, 64, or 128 bits deep. The value must be a hexadecimal number, for example, INIT=ABAC. If the INIT attribute is not specified, the RAM is initialized with zero.

For Virtex, Virtex-E, Spartan-II, and Spartan-IIE, lower INIT values get mapped to the G function generator and upper INIT values get mapped to the F function generator.

Refer to the "INIT" section of the *Constraints Guide* for more information on the INIT attribute.

For Virtex-II, and Virtex-II Pro, wide RAMs (2, 4, and 8-bit wide single port synchronous RAMs with a WCLK) can also be initialized. These RAMs, however, require INIT_xx attributes. See "Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM" in the RAM16X2S section for more information on initializing Virtex-II wide RAM.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM16X1D should be placed
-- after architecture statement but before begin keyword

component RAM16X1D
   -- synthesis translate_off
       generic (INIT : bit_vector   := X"16");
   -- synthesis translate_on
port (DPO   : out STD_ULOGIC;
      SPO   : out STD_ULOGIC;
      A0    : in STD_ULOGIC;
      A1    : in STD_ULOGIC;
      A2    : in STD_ULOGIC;
      A3    : in STD_ULOGIC;
      D     : in STD_ULOGIC;
      DPRA0 : in STD_ULOGIC;
      DPRA1 : in STD_ULOGIC;
      DPRA2 : in STD_ULOGIC;
      DPRA3 : in STD_ULOGIC;
      WCLK  : in STD_ULOGIC;
      WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM16X1D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM16X1D should be placed
-- in architecture after the begin keyword

RAM16X1D_INSTANCE_NAME : RAM16X1D
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (DPO    => user_DPO,
                 SPO    => user_SPO,
```

```
A0      => user_A0,
A1      => user_A1,
A2      => user_A2,
A3      => user_A3,
D       => user_D,
DPRA0   => user_DPRA0,
DPRA1   => user_DPRA1,
DPRA2   => user_DPRA2,
DPRA3   => user_DPRA3,
WCLK    => user_WCLK,
WE      => user_WE);
```

Verilog Instantiation Template

```
RAM16X1D instance_name (.DPO (user_DPO),
                        .SPO (user_SPO),
                        .A0(user_A0),
                        .A1(user_A1),
                        .A2(user_A2),
                        .A3(user_A3),
                        .D (user_D),
                        .DPRA0(user_DPRA0),
                        .DPRA1(user_DPRA1),
                        .DPRA2(user_DPRA2),
                        .DPRA3(user_DPRA3),
                        .WCLK(user_WCLK),
                        .WE (user_WE));

defparam RAM16X1D_instance_name.INIT = hex_value;
```

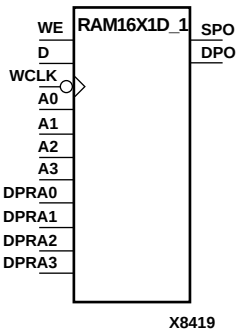
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X1D_1

16-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



RAM16X1D_1 is a 16-word by 1-bit static dual port random access memory with synchronous write capability and negative-edge clock. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM16X1D_1 during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↓	D	D	data_d
1 (read)	↑	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM16X1D_1 should be placed --
-- after architecture statement but before begin keyword

component RAM16X1D_1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"16");
   -- synthesis translate_on
      port (DPO : out STD_ULOGIC;
            SPO : out STD_ULOGIC;
            A0    : in STD_ULOGIC;
            A1    : in STD_ULOGIC;
            A2    : in STD_ULOGIC;
            A3    : in STD_ULOGIC;
            D     : in STD_ULOGIC;
            DPRA0 : in STD_ULOGIC;
            DPRA1 : in STD_ULOGIC;
            DPRA2 : in STD_ULOGIC;
            DPRA3 : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM16X1D_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM16X1D_1 should be placed
-- in architecture after the begin keyword

RAM16X1D_1_INSTANCE_NAME : RAM16X1D_1
   -- synthesis translate_off
   generic map (INIT => hex_value);
   -- synthesis translate_on
      port map  (DPO    => user_DPO,
                 SPO    => user_SPO,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 D      => user_D,
                 DPRA0  => user_DPRA0,
                 DPRA1  => user_DPRA1,
                 DPRA2  => user_DPRA2,
                 DPRA3  => user_DPRA3,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM16X1D_1 instance_name (.DPO (user_DPO),
                          .SPO (user_SPO),
                          .A0(user_A0),
                          .A1(user_A1),
                          .A2(user_A2),
                          .A3(user_A3),
                          .D (user_D),
                          .DPRA0(user_DPRA0),
                          .DPRA1(user_DPRA1),
                          .DPRA2(user_DPRA2),
                          .DPRA3(user_DPRA3),
                          .WCLK(user_WCLK),
                          .WE (user_WE));

defparam user_instance_name.INIT = hex_value;
```

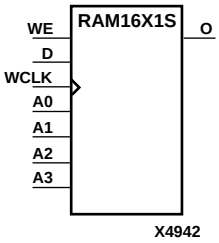
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X1S

16-Deep by 1-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



RAM16X1S is a 16-word by 1-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM16X1S during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE(mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM16X1S should be placed
-- after architecture statement but before begin keyword

component RAM16X1S
   -- synthesis translate_off
   generic (INIT: bit_vector := X"16");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A3 : in STD_ULOGIC;
            D  : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM16X1S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM16X1S should be placed
-- in architecture after the begin keyword

RAM16X1S_INSTANCE_NAME : RAM16X1S
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM16X1S instance_name (.O (user_O),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .D (user_D),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT = hex_value;
```

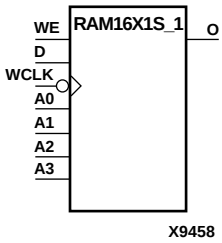
Commonly Used Constraints

BEL, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X1S_1

16-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N / A	N / A	N / A



RAM16X1S_1 is a 16-word by 1-bit static random access memory with synchronous write capability and negative-edge clock. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM16X1S_1 during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE(mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A3 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM16X1S_1 should be placed
-- after architecture statement but before begin keyword

component RAM16X1S_1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"16");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A3 : in STD_ULOGIC;
            D  : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM16X1S_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM16X1S_1 should be placed
-- in architecture after the begin keyword

RAM16X1S_1_INSTANCE_NAME : RAM16X1S_1
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM16X1S_1 instance_name (.O (user_O),
                          .A0 (user_A0),
                          .A1 (user_A1),
                          .A2 (user_A2),
                          .A3 (user_A3),
                          .D (user_D),
                          .WCLK (user_WCLK),
                          .WE (user_WE));

defparam user_instance_name.INIT = hex_value;
```

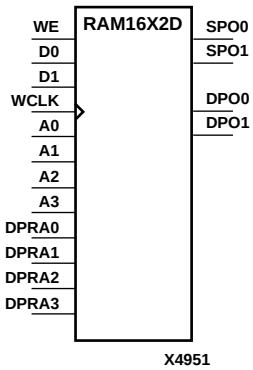
Commonly Used Constraints

BEL, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X2D

16-Deep by 2-Wide Static Dual Port Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



RAM16X2D is a 16-word by 2-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO1 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X2D cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D1-D0	SPO1-SPO0	DPO1-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D1-D0	D1-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note The write process is not affected by the address on the read address port.

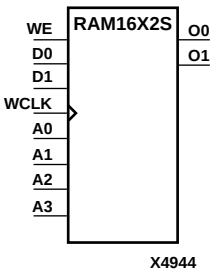
User

For HDL, this design element is inferred. See the *XST User Guide* for details.

RAM16X2S

16-Deep by 2-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Primitive	N/A	N/A	N/A



RAM16X2S is a 16-word by 2-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O1 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Virtex-II and Virtex-II Pro, the initial contents of RAM16X2S cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

For Virtex-II and Virtex-II Pro, you can use the INIT_00 and INIT_01 properties to specify the initial contents of RAM16X2S as described in "Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM" in this section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D1-D0	O1-O0
0 (read)	X	X	Data
1(read)	0	X	Data
1(read)	1	X	Data
1(write)	↑	D1-D0	D1-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM

You can use the INIT_xx properties to specify the initial contents of a Virtex-II and Virtex-II Pro wide RAM. INIT_00 initializes the RAM cells corresponding to the O0 output, INIT_01 initializes the cells corresponding to the O1 output, etc. For example, a RAM16X2S instance is initialized by INIT_00 and INIT_01 containing 4 hex characters each. A RAM16X8S instance is initialized by eight properties INIT_00 through INIT_07 containing 4 hex characters each. A RAM64x2S instance is completely initialized by two properties INIT_00 and INIT_01 containing 16 hex characters each. See the INIT_xx section of the *Constraints Guide* for more information on the INIT_xx attribute.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM16X2S should be placed
-- after architecture statement but before begin keyword

component RAM16X2S
   -- synthesis translate_off
   generic (INIT_00: bit_vector := X"16";
            INIT_01: bit_vector := X"16");
   -- synthesis translate_on
      port (O0    : out STD_ULOGIC;
            O1    : out STD_ULOGIC;
            A0    : in STD_ULOGIC;
            A1    : in STD_ULOGIC;
            A2    : in STD_ULOGIC;
            A3    : in STD_ULOGIC;
            D0    : in STD_ULOGIC;
            D1    : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM16X2S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM16X2S should be placed
-- in architecture after the begin keyword

RAM16X2S_INSTANCE_NAME : RAM16X2S
   -- synthesis translate_off
   generic map(INIT_00 => hex_value;
               INIT_01 => hex_value);
   -- synthesis translate_on
      port map  (O0     => user_O0,
                 O1     => user_O1,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 D0     => user_D0,
                 D1     => user_D1,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM16X2S instance_name (.O0 (user_O0),
                        .O1 (user_O1),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3)
                        .D0 (user_D0),
                        .D1 (user_D1),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT_00 = hex_value;
defparam user_instance_name.INIT_01 = hex_value;
```

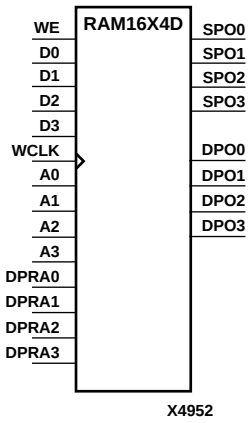
Commonly Used Constraints

BEL, INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X4D

16-Deep by 4-Wide Static Dual Port Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



RAM16X4D is a 16-word by 4-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO3 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D3 – D0) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X4D cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D3-D0	SPO3-SPO0	DPO3-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D3-D0	D3-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note The write process is not affected by the address on the read address port.

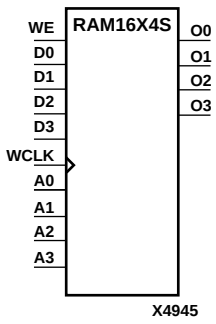
Usage

For HDL, this design element must be inferred. For information on how to infer RAM, please refer to the *XST User Guide*.

RAM16X4S

16-Deep by 4-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Primitive	N / A	N / A	N / A



RAM16X4S is a 16-word by 4-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D3 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O3 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Virtex-II and Virtex-II Pro, the initial contents of RAM16X4S cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

For Virtex-II and Virtex-II Pro, you can use INIT_00 through INIT_03 to specify the initial contents of RAM16X4S as described in the "Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM"section in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D3 – D0	O3 – O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D3-D0	D3-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM16X4S should be placed
-- after architecture statement but before begin keyword

component RAM16X4S
   -- synthesis translate_off
   generic (INIT_00 : bit_vector := X"16";
            INIT_01 : bit_vector := X"16";
            INIT_02 : bit_vector := X"16";
            INIT_03 : bit_vector := X"16");
   -- synthesis translate_on
      port (O0    : out STD_ULOGIC;
            O1    : out STD_ULOGIC;
            O2    : out STD_ULOGIC;
            O3    : out STD_ULOGIC;
            A0    : in STD_ULOGIC;
            A1    : in STD_ULOGIC;
            A2    : in STD_ULOGIC;
            A3    : in STD_ULOGIC;
            D0    : in STD_ULOGIC;
            D1    : in STD_ULOGIC;
            D2    : in STD_ULOGIC;
            D3    : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM16X4S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM16X4S should be placed
-- in architecture after the begin keyword

RAM16X4S_INSTANCE_NAME : RAM16X4S
   -- synthesis translate_off
   generic map(INIT_00 => hex_value,
               INIT_01 => hex_value,
               INIT_02 => hex_value,
               INIT_03 => hex_value);
   -- synthesis translate_on
      port map  (O0     => user_O0,
                 O1     => user_O1,
                 O2     => user_O2,
```

```
                    O3     => user_O3,
                    A0     => user_A0,
                    A1     => user_A1,
                    A2     => user_A2,
                    A3     => user_A3,
                    D0     => user_D0,
                    D1     => user_D1,
                    D2     => user_D2,
                    D3     => user_D3,
                    WCLK   => user_WCLK,
                    WE     => user_WE);
```

Verilog Instantiation Template

```
RAM16X4S instance_name (.O0 (user_O0),
                        .O1 (user_O1),
                        .O2 (user_O2),
                        .O3 (user_O3),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .D0 (user_D0),
                        .D1 (user_D1),
                        .D2 (user_D2),
                        .D3 (user_D3),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT_00 = hex_value;
defparam user_instance_name.INIT_01 = hex_value;
defparam user_instance_name.INIT_02 = hex_value;
defparam user_instance_name.INIT_03 = hex_value;
```

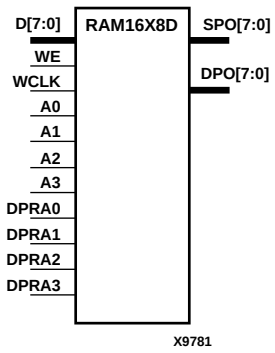
Commonly Used Constraints

BEL, INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X8D

16-Deep by 8-Wide Static Dual Port Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	N / A	N / A	N / A



RAM16X8D is a 16-word by 8-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO7 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D7 – D0) into the word selected by the 4-bit write address (A3 – A0). For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X8D cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D7-D0	SP7-SPO0	DPO7-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D7-D0	D7-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

The write process is not affected by the address on the read address port.

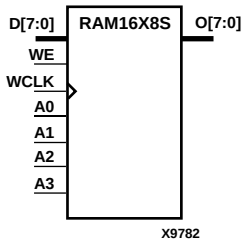
Usage

For HDL, this design element must be inferred. For information on how to infer RAM, please refer to the *XST User Guide*.

RAM16X8S

16-Deep by 8-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Primitive	N / A	N / A	N / A



RAM16X8S is a 16-word by 8-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on data inputs (D7 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O7 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Virtex-II and Virtex-II Pro, the initial contents of RAM16X8S cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

For Virtex-II and Virtex-II Pro, you can use INIT_00 through INIT_07 to specify the initial contents of RAM16X8S as described in the "Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM" section in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D7-D0	O7-O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D7-D0	D7-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM16X8S should be placed
-- after architecture statement but before begin keyword

component RAM16X8S
   -- synthesis translate_off
   generic (INIT_00 : bit_vector := X"16";
            INIT_01 : bit_vector := X"16";
            INIT_02 : bit_vector := X"16";
            INIT_03 : bit_vector := X"16";
            INIT_04 : bit_vector := X"16";
            INIT_05 : bit_vector := X"16";
            INIT_06 : bit_vector := X"16";
            INIT_07 : bit_vector := X"16");
   -- synthesis translate_on
      port (O0    : out STD_ULOGIC;
            A0    : in STD_ULOGIC;
            A1    : in STD_ULOGIC;
            A2    : in STD_ULOGIC;
            A3    : in STD_ULOGIC;
            D     : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM16X8S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for RAM16X8S should be placed
-- in architecture after the begin keyword

RAM16X8S_INSTANCE_NAME : RAM16X8S
   -- synthesis translate_off
   generic map(INIT_00 => hex_value,
               INIT_01 => hex_value,
               INIT_02 => hex_value,
               INIT_03 => hex_value,
               INIT_04 => hex_value,
               INIT_05 => hex_value,
               INIT_06 => hex_value,
               INIT_07 => hex_value);
   -- synthesis translate_on
      port map  (O0     => user_O0,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM16X8S instance_name (.O0 (user_O0),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .D (user_D),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT_00 = hex_value;
defparam user_instance_name.INIT_01 = hex_value;
defparam user_instance_name.INIT_02 = hex_value;
defparam user_instance_name.INIT_03 = hex_value;
defparam user_instance_name.INIT_04 = hex_value;
defparam user_instance_name.INIT_05 = hex_value;
defparam user_instance_name.INIT_06 = hex_value;
defparam user_instance_name.INIT_07 = hex_value;
```

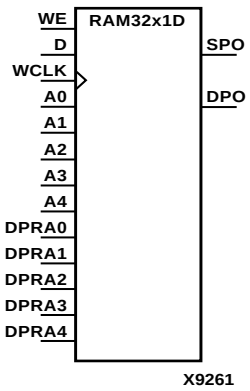
Commonly Used Constraints

BEL, INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM32X1D

32-Deep by 1-Wide Static Dual Static Port Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



RAM32X1D is a 32-word by 1-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA4 – DPRA0) and the write address (A4 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM32X1D during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D	D	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A4-A0

data_d = word addressed by bits DPRA4-DPRA0

The SPO output reflects the data in the memory cell addressed by A4 – A0. The DPO output reflects the data in the memory cell addressed by DPRA4 – DPRA0.

Note The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM32X1D should be placed
-- after architecture statement but before begin keyword

component RAM32X1D
   -- synthesis translate_off
       generic (INIT : bit_vector   := X"32");
   -- synthesis translate_on
port (DPO   : out STD_ULOGIC;
      SPO   : out STD_ULOGIC;
      A0    : in STD_ULOGIC;
      A1    : in STD_ULOGIC;
      A2    : in STD_ULOGIC;
      A3    : in STD_ULOGIC;
      A4    : in STD_ULOGIC;
      D     : in STD_ULOGIC;
      DPRA0 : in STD_ULOGIC;
      DPRA1 : in STD_ULOGIC;
      DPRA2 : in STD_ULOGIC;
      DPRA3 : in STD_ULOGIC;
      DPRA4 : in STD_ULOGIC;
      WCLK  : in STD_ULOGIC;
      WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM32X1D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM32X1D should be placed
-- in architecture after the begin keyword

RAM32X1D_INSTANCE_NAME : RAM32X1D
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (DPO    => user_DPO,
                 SPO    => user_SPO,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 D      => user_D,
```

```
                    DPRA0  => user_DPRA0,
                    DPRA1  => user_DPRA1,
                    DPRA2  => user_DPRA2,
                    DPRA3  => user_DPRA3,
                    DPRA4  => user_DPRA4,
                    WCLK   => user_WCLK,
                    WE     => user_WE);
```

Verilog Instantiation Template

```
RAM32X1D instance_name (.DPO (user_DPO),
                        .SPO (user_SPO),
                        .A0(user_A0),
                        .A1(user_A1),
                        .A2(user_A2),
                        .A3(user_A3),
                        .A4(user_A4),
                        .D (user_D),
                        .DPRA0(user_DPRA0),
                        .DPRA1(user_DPRA1),
                        .DPRA2(user_DPRA2),
                        .DPRA3(user_DPRA3),
                        .DPRA4(user_DPRA4),
                        .WCLK(user_WCLK),
                        .WE (user_WE));

defparam RAM32X1D_instance_name.INIT = hex_value;
```

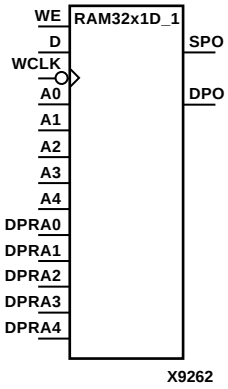
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM32X1D_1

32-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



RAM32X1D_1 is a 32-word by 1-bit static dual port random access memory with synchronous write capability and a negative-edge clock. The device has two separate address ports: the read address (DPRA4 – DPRA0) and the write address (A4 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit write address. For predictable performance, write address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM32X1D_1 during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↓	D	D	data_d
1 (read)	↑	X	data_a	data_d

data_a = word addressed by bits A4-A0

data_d = word addressed by bits DPRA4-DPRA0

The SPO output reflects the data in the memory cell addressed by A4 – A0. The DPO output reflects the data in the memory cell addressed by DPRA4 – DPRA0.

Note The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM32X1D_1 should be placed
-- after architecture statement but before begin keyword

component RAM32X1D_1
   -- synthesis translate_off
       generic (INIT : bit_vector   := X"32");
   -- synthesis translate_on
port (DPO : out STD_ULOGIC;
      SPO : out STD_ULOGIC;
      A0    : in STD_ULOGIC;
      A1    : in STD_ULOGIC;
      A2    : in STD_ULOGIC;
      A3    : in STD_ULOGIC;
      A4    : in STD_ULOGIC;
      D     : in STD_ULOGIC;
      DPRA0 : in STD_ULOGIC;
      DPRA1 : in STD_ULOGIC;
      DPRA2 : in STD_ULOGIC;
      DPRA3 : in STD_ULOGIC;
      DPRA4 : in STD_ULOGIC;
      WCLK  : in STD_ULOGIC;
      WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM32X1D_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM32X1D_1 should be placed
-- in architecture after the begin keyword

RAM32X1D_1_INSTANCE_NAME : RAM32X1D_1
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (DPO    => user_DPO,
                 SPO    => user_SPO,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 D      => user_D,
```

```
                    DPRA0  => user_DPRA0,
                    DPRA1  => user_DPRA1,
                    DPRA2  => user_DPRA2,
                    DPRA3  => user_DPRA3,
                    DPRA4  => user_DPRA4,
                    WCLK   => user_WCLK,
                    WE     => user_WE);
```

Verilog Instantiation Template

```
RAM32X1D_1 instance_name (.DPO (user_DPO),
                          .SPO (user_SPO),
                          .A0(user_A0),
                          .A1 (user_A1),
                          .A2 (user_A2),
                          .A3 (user_A3),
                          .A4 (user_A4),
                          .D (user_D),
                          .DPRA0(user_DPRA0),
                          .DPRA1(user_DPRA1),
                          .DPRA2(user_DPRA2),
                          .DPRA3(user_DPRA3),
                          .DPRA4(user_DPRA4),
                          .WCLK (user_WCLK),
                          .WE (user_WE));

defparam RAM32X1D_1_instance_name.INIT = hex_value;
```

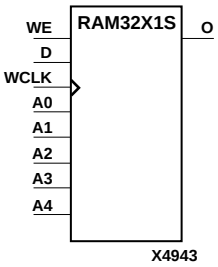
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM32X1S

32-Deep by 1-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N / A	N / A	N / A



RAM32X1S is a 32-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM32X1S during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM32X1S should be placed
-- after architecture statement but before begin keyword

component RAM32X1S
   -- synthesis translate_off
   generic (INIT: bit_vector := X"32");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            D  : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM32X1S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for RAM32X1S should be placed
-- in architecture after the begin keyword

RAM32X1S_INSTANCE_NAME : RAM32X1S
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM32X1S instance_name (.O (user_O),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .A4 (user_A4),
                        .D (user_D),
                        .WCLK (user_WCLK),
                        .WE (user_WE));
defparam user_instance_name.INIT = hex_value;
```

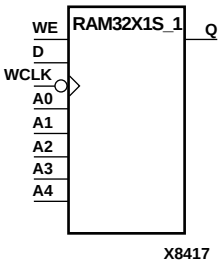
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM32X1S_1

32-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N / A	N / A	N / A



RAM32X1S_1 is a 32-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM32X1S_1 during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM32X1S_1 should be placed
-- after architecture statement but before begin keyword

component RAM32X1S_1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"32");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            D  : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM32X1S_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM32X1S_1 should be placed
-- in architecture after the begin keyword

RAM32X1S_1_INSTANCE_NAME : RAM32X1S_1
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM32X1S_1 instance_name (.O (user_O),
                          .A0 (user_A0),
                          .A1 (user_A1),
                          .A2 (user_A2),
                          .A3 (user_A3),
                          .A4 (user_A4),
                          .D (user_D),
                          .WCLK (user_WCLK),
                          .WE (user_WE));

defparam user_instance_name.INIT = hex_value;
```

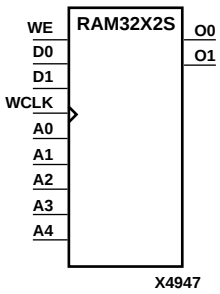
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM32X2S

32-Deep by 2-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Primitive	N / A	N / A	N / A



RAM32X2S is a 32-word by 2-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O1 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Virtex-II and Virtex-II Pro, the initial contents of RAM32X2S cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

For Virtex-II and Virtex-II Pro, you can use the INIT_00 and INIT_01 properties to specify the initial contents of RAM32X2S as described in "Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM" in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D0-D1	O0-O1
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D1-D0	D1-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM32X2S should be placed
-- after architecture statement but before begin keyword

component RAM32X2S
   -- synthesis translate_off
   generic (INIT_00: bit_vector := X"32";
            INIT_01: bit_vector := X"32");
   -- synthesis translate_on
      port (O0    : out STD_ULOGIC;
            O1    : out STD_ULOGIC;
            A0    : in STD_ULOGIC;
            A1    : in STD_ULOGIC;
            A2    : in STD_ULOGIC;
            A3    : in STD_ULOGIC;
            A4    : in STD_ULOGIC;
            D0    : in STD_ULOGIC;
            D1    : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM32X2S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM32X2S should be placed
-- in architecture after the begin keyword

RAM32X2S_INSTANCE_NAME : RAM32X2S
   -- synthesis translate_off
   generic map(INIT_00 => hex_value,
               INIT_01 => hex_value);
   -- synthesis translate_on
      port map  (O0     => user_O0,
                 O1     => user_O1,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 D0     => user_D0,
                 D1     => user_D1,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM32X2S instance_name (.O0 (user_O0),
                        .O1 (user_O1),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .A4 (user_A4),
                        .D0 (user_D0),
                        .D1 (user_D1),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT_00 = hex_value;
defparam user_instance_name.INIT_01 = hex_value;
```

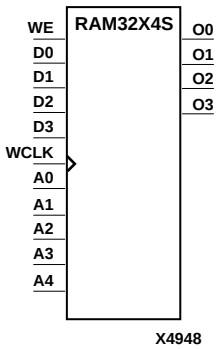
Commonly Used Constraints

INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM32X4S

32-Deep by 4-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Primitive	N/A	N/A	N/A



RAM32X4S is a 32-word by 4-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data inputs (D3 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O3 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Virtex-II and Virtex-II Pro, the initial contents of RAM32X4S cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

For Virtex-II and Virtex-II Pro, you can use the INIT_00 through INIT_03 properties to specify the initial contents of RAM32X4S as described in "Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM" in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE	WCLK	D3-D0	O3-O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D3-D0	D3-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM32X4S should be placed
-- after architecture statement but before begin keyword

component RAM32X4S
   -- synthesis translate_off
   generic (INIT_00 : bit_vector := X"32";
            INIT_01 : bit_vector := X"32";
            INIT_02 : bit_vector := X"32";
            INIT_03 : bit_vector := X"32");
   -- synthesis translate_on
      port (O0    : out STD_ULOGIC;
            O1    : out STD_ULOGIC;
            O2    : out STD_ULOGIC;
            O3    : out STD_ULOGIC;
            A0    : in STD_ULOGIC;
            A1    : in STD_ULOGIC;
            A2    : in STD_ULOGIC;
            A3    : in STD_ULOGIC;
            A4    : in STD_ULOGIC;
            D0    : in STD_ULOGIC;
            D1    : in STD_ULOGIC;
            D2    : in STD_ULOGIC;
            D3    : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM32X4S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM32X4S should be placed
-- in architecture after the begin keyword

RAM32X4S_INSTANCE_NAME : RAM32X4S
   -- synthesis translate_off
   generic map(INIT_00 => hex_value,
               INIT_01 => hex_value,
               INIT_02 => hex_value,
               INIT_03 => hex_value);
   -- synthesis translate_on
      port map  (O0     => user_O0,
                 O1     => user_O1,
                 O2     => user_O2,
                 O3     => user_O3,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
```

```
A4      => user_A4,
D0      => user_D0,
D1      => user_D1,
D2      => user_D2,
D3      => user_D3,
WCLK    => user_WCLK,
WE      => user_WE);
```

Verilog Instantiation Template

```
RAM32X4S instance_name (.O0 (user_O0),
                        .O1 (user_O1),
                        .O2 (user_O2),
                        .O3 (user_O3),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .A4 (user_A4),
                        .D0 (user_D0),
                        .D1 (user_D1),
                        .D2 (user_D2),
                        .D3 (user_D3),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT_00 = hex_value;
defparam user_instance_name.INIT_01 = hex_value;
defparam user_instance_name.INIT_02 = hex_value;
defparam user_instance_name.INIT_03 = hex_value;
```

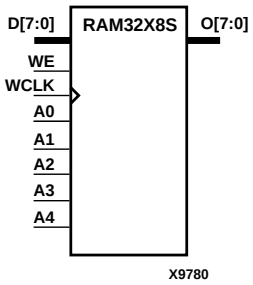
Commonly Used Constraints

INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM32X8S

32-Deep by 8-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Primitive	N / A	N / A	N / A



RAM32X8S is a 32-word by 8-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data inputs (D7 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O7 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

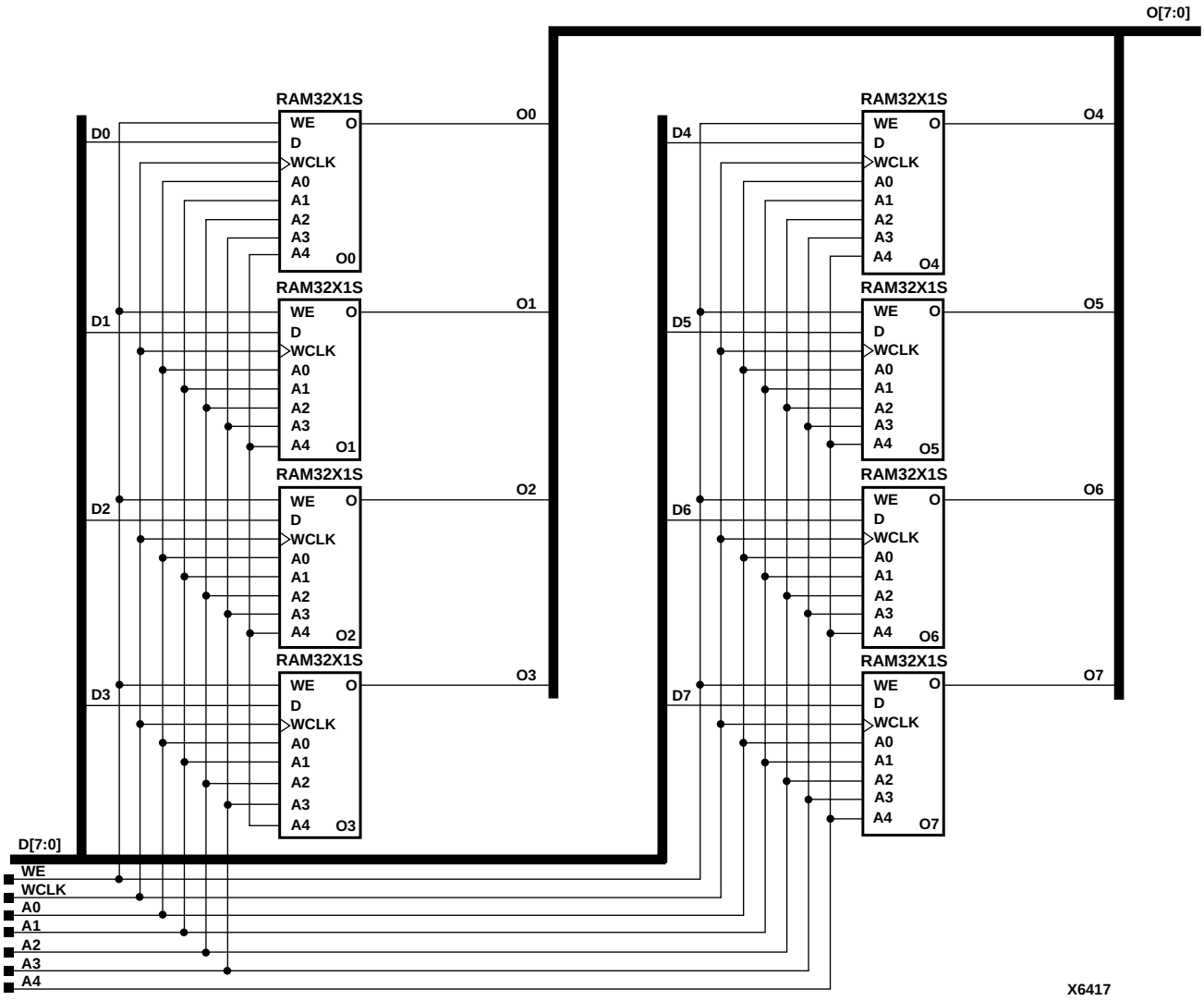
Except for Virtex-II and Virtex-II Pro, the initial contents of RAM32X8S cannot be specified directly. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

For Virtex-II and Virtex-II Pro, you can use the INIT_00 through INIT_07 properties to specify the initial contents of RAM32X8S as described in "Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM" in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D7-D0	O7-O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D7-D0	D7-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0



RAM32X8S Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM32X8S should be placed
-- after architecture statement but before begin keyword

component RAM32X8S
   -- synthesis translate_off
   generic (INIT_00 : bit_vector := X"32";
            INIT_01 : bit_vector := X"32";
            INIT_02 : bit_vector := X"32";
            INIT_03 : bit_vector := X"32";
            INIT_04 : bit_vector := X"32";
            INIT_05 : bit_vector := X"32";
            INIT_06 : bit_vector := X"32";
            INIT_07 : bit_vector := X"32");
   -- synthesis translate_on
      port (O0    : out STD_ULOGIC;
            A0    : in STD_ULOGIC;
            A1    : in STD_ULOGIC;
            A2    : in STD_ULOGIC;
            A3    : in STD_ULOGIC;
            A4    : in STD_ULOGIC;
            D     : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM32X8S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM32X8S should be placed
-- in architecture after the begin keyword

RAM32X8S_INSTANCE_NAME : RAM32X8S
   -- synthesis translate_off
   generic map(INIT_00 => hex_value,
            INIT_01 => hex_value,
            INIT_02 => hex_value,
            INIT_03 => hex_value,
            INIT_04 => hex_value,
            INIT_05 => hex_value,
            INIT_06 => hex_value,
            INIT_07 => hex_value);
   -- synthesis translate_on
      port map  (O0     => user_O0,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
```

```
                    D      => user_D,
                    WCLK   => user_WCLK,
                    WE     => user_WE);
```

Verilog Instantiation Template

```
RAM32X8S instance_name (.O0 (user_O0),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .A4 (user_A4),
                        .D (user_D),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT_00 = hex_value;
defparam user_instance_name.INIT_01 = hex_value;
defparam user_instance_name.INIT_02 = hex_value;
defparam user_instance_name.INIT_03 = hex_value;
defparam user_instance_name.INIT_04 = hex_value;
defparam user_instance_name.INIT_05 = hex_value;
defparam user_instance_name.INIT_06 = hex_value;
defparam user_instance_name.INIT_07 = hex_value;
```

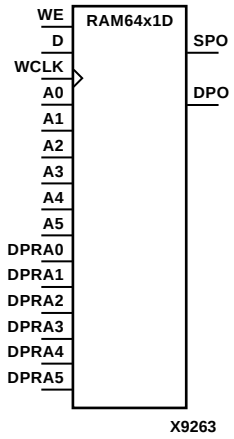
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, INIT_xx, LOC, RLOC, U_SET, XBLKNM

RAM64X1D

64-Deep by 1-Wide Dual Port Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



RAM64X1D is a 64-word by 1-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA5 – DPRA0) and the write address (A5 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 6-bit (A0 - A5) write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM64X1D during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D	D	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A5-A0

data_d = word addressed by bits DPRA5-DPRA0

The SPO output reflects the data in the memory cell addressed by A5 – A0. The DPO output reflects the data in the memory cell addressed by DPRA5 – DPRA0.

Note The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM64X1D should be placed
-- after architecture statement but before begin keyword

component RAM64X1D
   -- synthesis translate_off
       generic (INIT : bit_vector   := X"64");
   -- synthesis translate_on
port (DPO : out STD_ULOGIC;
      SPO : out STD_ULOGIC;
      A0    : in STD_ULOGIC;
      A1    : in STD_ULOGIC;
      A2    : in STD_ULOGIC;
      A3    : in STD_ULOGIC;
      A4    : in STD_ULOGIC;
      A5    : in STD_ULOGIC;
      D     : in STD_ULOGIC;
      DPRA0 : in STD_ULOGIC;
      DPRA1 : in STD_ULOGIC;
      DPRA2 : in STD_ULOGIC;
      DPRA3 : in STD_ULOGIC;
      DPRA4 : in STD_ULOGIC;
      DPRA5 : in STD_ULOGIC;
      WCLK  : in STD_ULOGIC;
      WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM64X1D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM64X1D should be placed
-- in architecture after the begin keyword

RAM64X1D_INSTANCE_NAME : RAM64X1D
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (DPO    => user_DPO,
                 SPO    => user_SPO,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
```

```
A4      => user_A4,
A5      => user_A5,
D       => user_D,
DPRA0   => user_DPRA0,
DPRA1   => user_DPRA1,
DPRA2   => user_DPRA2,
DPRA3   => user_DPRA3,
DPRA4   => user_DPRA4,
DPRA5   => user_DPRA5,
WCLK    => user_WCLK,
WE      => user_WE);
```

Verilog Instantiation Template

```
RAM64X1D instance_name (.DPO (user_DPO),
                        .SPO (user_SPO),
                        .A0(user_A0),
                        .A1(user_A1),
                        .A2(user_A2),
                        .A3(user_A3),
                        .A4(user_A4),
                        .A5(user_A5),
                        .D (user_D),
                        .DPRA0(user_DPRA0),
                        .DPRA1(user_DPRA1),
                        .DPRA2(user_DPRA2),
                        .DPRA3(user_DPRA3),
                        .DPRA4(user_DPRA4),
                        .DPRA5(user_DPRA5),
                        .WCLK(user_WCLK),
                        .WE (user_WE));

defparam RAM64X1D_instance_name.INIT = hex_value;
```

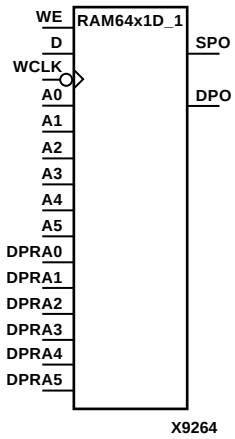
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM64X1D_1

64-Deep by 1-Wide Dual Port Static Synchronous RAM with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A



RAM64X1D_1 is a 64-word by 1-bit static dual port random access memory with synchronous write capability and a negative-edge clock. The device has two separate address ports: the read address (DPRA5 – DPRA0) and the write address (A5 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 6-bit (A0 - A5) write address. For predictable performance, write address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM64X1D_1 during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↓	D	D	data_d
1 (read)	↑	X	data_a	data_d

data_a = word addressed by bits A5-A0

data_d = word addressed by bits DPRA5-DPRA0

The SPO output reflects the data in the memory cell addressed by A5 – A0. The DPO output reflects the data in the memory cell addressed by DPRA5 – DPRA0.

Note The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM64X1D_1 should be placed
-- after architecture statement but before begin keyword

component RAM64X1D_1
   -- synthesis translate_off
       generic (INIT : bit_vector   := X"64");
   -- synthesis translate_on
port (DPO : out STD_ULOGIC;
      SPO : out STD_ULOGIC;
      A0    : in STD_ULOGIC;
      A1    : in STD_ULOGIC;
      A2    : in STD_ULOGIC;
      A3    : in STD_ULOGIC;
      A4    : in STD_ULOGIC;
      A5    : in STD_ULOGIC;
      D     : in STD_ULOGIC;
      DPRA0 : in STD_ULOGIC;
      DPRA1 : in STD_ULOGIC;
      DPRA2 : in STD_ULOGIC;
      DPRA3 : in STD_ULOGIC;
      DPRA4 : in STD_ULOGIC;
      DPRA5 : in STD_ULOGIC;
      WCLK  : in STD_ULOGIC;
      WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM64X1D_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM64X1D_1 should be placed
-- in architecture after the begin keyword

RAM64X1D_1_INSTANCE_NAME : RAM64X1D_1
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (DPO    => user_DPO,
                 SPO    => user_SPO,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 A5     => user_A5,
                 D      => user_D,
                 DPRA0  => user_DPRA0,
                 DPRA1  => user_DPRA1,
                 DPRA2  => user_DPRA2,
```

```
                    DPRA3  => user_DPRA3,
                    DPRA4  => user_DPRA4,
                    DPRA5  => user_DPRA5,
                    WCLK   => user_WCLK,
                    WE     => user_WE);
```

Verilog Instantiation Template

```
RAM64X1D_1 instance_name (.DPO (user_DPO),
                          .SPO (user_SPO),
                          .A0(user_A0),
                          .A1(user_A1),
                          .A2(user_A2),
                          .A3(user_A3),
                          .A4(user_A4),
                          .A5(user_A5),
                          .D (user_D),
                          .DPRA0(user_DPRA0),
                          .DPRA1(user_DPRA1),
                          .DPRA2(user_DPRA2),
                          .DPRA3(user_DPRA3),
                          .DPRA4(user_DPRA4),
                          .DPRA5(user_DPRA5),
                          .WCLK(user_WCLK),
                          .WE (user_WE));

defparam RAM64X1D_1_instance_name.INIT = hex_value;
```

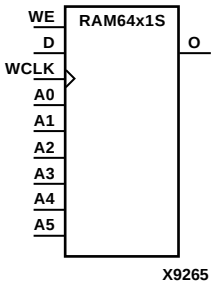
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM64X1S

64-Deep by 1-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



RAM64X1S is a 64-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 6-bit address (A5 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM64X1S during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A5 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM64X1S should be placed
-- after architecture statement but before begin keyword

component RAM64X1S
   -- synthesis translate_off
   generic (INIT: bit_vector := X"64");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            A5 : in STD_ULOGIC;
            D  : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM64X1S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM64X1S should be placed
-- in architecture after the begin keyword

RAM64X1S_INSTANCE_NAME : RAM64X1S
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 A5     => user_A5,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM64X1S instance_name (.O (user_O),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .A4 (user_A4),
                        .A5 (user_A5),
                        .D (user_D),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT = hex_value;
```

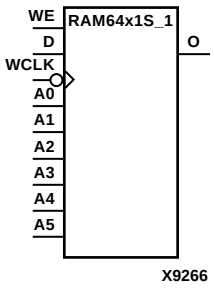
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM64X1S_1

64-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A



RAM64X1S_1 is a 64-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 6-bit address (A5 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM32X1S_1 during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A5 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM64X1S_1 should be placed
-- after architecture statement but before begin keyword

component RAM64X1S_1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"64");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            A5 : in STD_ULOGIC;
            D  : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM64X1S_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM64X1S_1 should be placed
-- in architecture after the begin keyword

RAM64X1S_1_INSTANCE_NAME : RAM64X1S_1
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 A5     => user_A5,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM64X1S_1 instance_name (.O (user_O),
                          .A0 (user_A0),
                          .A1  (user_A1),
                          .A2  (user_A2),
                          .A3  (user_A3),
                          .A4  (user_A4),
                          .A5 (user_A5),
                          .D (user_D),
                          .WCLK (user_WCLK),
                          .WE (user_WE));

defparam user_instance_name.INIT = hex_value;
```

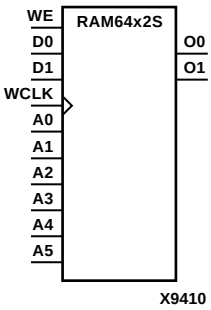
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM64X2S

64-Deep by 2-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



RAM64X2S is a 64-word by 2-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 6-bit address (A5 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O1 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can use the INIT_00 and INIT_01 properties to specify the initial contents of RAM64X2S as described in "Specifying Initial Contents of a Virtex-II and Virtex-II Pro Wide RAM" in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D0-D1	O0-O1
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D1-D0	D1-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM64X2S should be placed
-- after architecture statement but before begin keyword

component RAM64X2S
   -- synthesis translate_off
   generic (INIT_00: bit_vector := X"64";
            INIT_01: bit_vector := X"64");
   -- synthesis translate_on
      port (O0    : out STD_ULOGIC;
            O1    : out STD_ULOGIC;
            A0    : in STD_ULOGIC;
            A1    : in STD_ULOGIC;
            A2    : in STD_ULOGIC;
            A3    : in STD_ULOGIC;
            A4    : in STD_ULOGIC;
            A5    : in STD_ULOGIC;
            D0    : in STD_ULOGIC;
            D1    : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM64X2S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM64X2S should be placed
-- in architecture after the begin keyword

RAM64X2S_INSTANCE_NAME : RAM64X2S
   -- synthesis translate_off
   generic map(INIT_00 => hex_value,
               INIT_01 => hex_value);
   -- synthesis translate_on
      port map  (O0     => user_O0,
                 O1     => user_O1,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 A5     => user_A5,
                 D0     => user_D0,
                 D1     => user_D1,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM64X2S instance_name (.O0 (user_O0),
                        .O1 (user_O1),
                        .A0 (user_A0),
                        .A1 (user_A1),
                        .A2 (user_A2),
                        .A3 (user_A3),
                        .A4 (user_A4),
                        .A5 (user_A5),
                        .D0 (user_D0),
                        .D1 (user_D1),
                        .WCLK (user_WCLK),
                        .WE (user_WE));

defparam user_instance_name.INIT_00 = hex_value;
defparam user_instance_name.INIT_01 = hex_value;
```

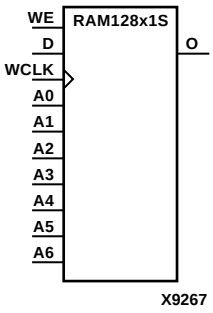
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM128X1S

128-Deep by 1-Wide Static Synchronous RAM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



RAM128X1S is a 128-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 7-bit address (A6 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM128X1S during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A6 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM128X1S should be placed
-- after architecture statement but before begin keyword

component RAM128X1S
   -- synthesis translate_off
   generic (INIT: bit_vector := X"128");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            A5 : in STD_ULOGIC;
            A6 : in STD_ULOGIC;
            D  : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM128X1S
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM128X1S should be placed
-- in architecture after the begin keyword

RAM128X1S_INSTANCE_NAME : RAM128X1S
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 A5     => user_A5,
                 A6     => user_A6,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM128X1S instance_name (.O (user_O),
                         .A0 (user_A0),
                         .A1 (user_A1),
                         .A2 (user_A2),
                         .A3 (user_A3),
                         .A4 (user_A4),
                         .A5(user_A5),
                         .A6(user_A6),
                         .D (user_D),
                         .WCLK (user_WCLK),
                         .WE (user_WE));

defparam user_instance_name.INIT = hex_value;
```

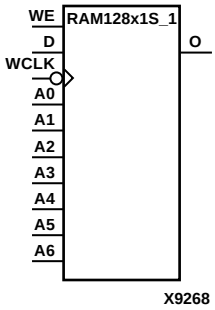
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM128X1S_1

128-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	Primitive	N / A	N / A	N / A



RAM128X1S_1 is a 128-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 7-bit address (A6 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM128X1S_1 during configuration using the INIT attribute. See "Specifying Initial Contents of a RAM" in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A6 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template

```
-- Component Declaration for RAM128X1S_1 should be placed
-- after architecture statement but before begin keyword

component RAM128X1S_1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"128");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            A5 : in STD_ULOGIC;
            A6 : in STD_ULOGIC;
            D  : in STD_ULOGIC;
            WCLK  : in STD_ULOGIC;
            WE    : in STD_ULOGIC);
end component;

-- Component Attribute specification for RAM128X1S_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for RAM128X1S_1 should be placed
-- in architecture after the begin keyword

RAM128X1S_1_INSTANCE_NAME : RAM128X1S_1
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 A5     => user_A5,
                 A6     => user_A6,
                 D      => user_D,
                 WCLK   => user_WCLK,
                 WE     => user_WE);
```

Verilog Instantiation Template

```
RAM128X1S_1 instance_name (.O (user_O),
                           .A0 (user_A0),
                           .A1 (user_A1),
                           .A2 (user_A2),
                           .A3 (user_A3),
                           .A4 (user_A4),
                           .A5(user_A5),
                           .A6(user_A6),
                           .D (user_D),
                           .WCLK (user_WCLK),
                           .WE (user_WE));

defparam user_instance_name.INIT = hex_value;
```

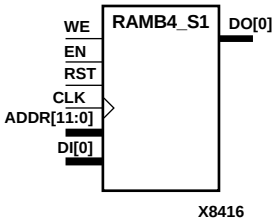
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

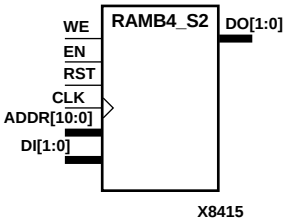
RAMB4_S*n*

4096-Bit Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 8, or 16 Bits

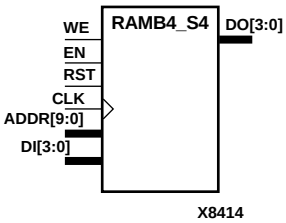
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	N/A	N/A	N/A	N/A



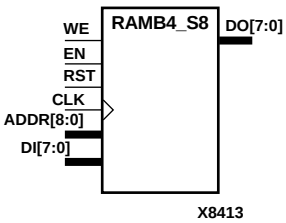
X8416



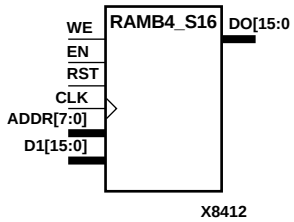
X8415



X8414



X8413



X8412

RAMB4_S1, RAMB4_S2, RAMB4_S4, RAMB4_S8, and RAMB4_S16 are dedicated random access memory blocks with synchronous write capability. They provide the capability for fast, discrete, large blocks of RAM in each Virtex, Virtex-E, Spartan-II, and Spartan-IIE device.The RAMB4_Sn cell configurations are listed in the following table.

Component	Depth	Width	Address Bus	Data Bus
RAMB4_S1	4096	1	(11:0)	(0:0)
RAMB4_S2	2048	2	(10:0)	(1:0)
RAMB4_S4	1024	4	(9:0)	(3:0)
RAMB4_S8	512	8	(8:0)	(7:0)
RAMB4_S16	256	16	(7:0)	(15:0)

The enable (EN) pin controls read, write, and reset. When EN is Low, no data is written and the output (DO) retains the last state. When EN is High and reset (RST) is High, DO is cleared during the Low-to-High clock (CLK) transition; if write enable (WE) is High, the memory contents reflect the data at DI. When EN is High and WE is Low, the data stored in the RAM address (ADDR) is read during the Low-to-High clock transition. When EN and WE are High, the data on the data input (DI) is loaded into the word selected by the write address (ADDR) during the Low-to-High clock transition and the data output (DO) reflects the selected (addressed) word.

The above description assumes an active High EN, WE, RST, and CLK. However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB4 port is absorbed into the block and does not use a CLB resource.

RAMB4_Sn's may be initialized during configuration. See "Specifying Initial Contents of a Block RAM" below.

Block RAM output registers are asynchronously cleared, output Low, when power is applied. The initial contents of the block RAM are not altered.

Virtex, Virtex-E, Spartan-II, and Spartan-IIE simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Mode selection is shown in the following truth table.

Inputs						Outputs	
EN	RST	WE	CLK	ADDR	DI	DO	RAM Contents
0	X	X	X	X	X	No Chg	No Chg
1	1	0	↑	X	X	0	No Chg
1	1	1	↑	addr	data	0	RAM(addr) =>data
1	0	0	↑	addr	X	RAM(addr)	No Chg
1	0	1	↑	addr	data	data	RAM(addr) =>data

addr=RAM address

RAM(addr)=RAM contents at address ADDR

data=RAM input data

Specifying Initial Contents of a Block RAM

You can use the INIT_xx attributes to specify an initial value during device configuration. The initialization of each RAMB4_Sn is set by 16 initialization attributes (INIT_00 through INIT_0F) of 64 hex values for a total of 4096 bits. See the INIT_xx section of the *Constraints Guide* for more information on these attributes.

If any INIT_0x attribute is not specified, it is configured as zeros. Partial initialization strings are padded with zeros to the left.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template for RAMB4_S*n*

```
-- Component Declaration for RAMB4_Sn
-- Should be placed after architecture statement but before begin keyword
component RAMB4_Sn
  -- synthesis translate_off
  generic (
      INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_08 : bit_vector :=
```

```
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_09 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
);
  -- synthesis translate_on
  port (DO    : out STD_LOGIC_VECTOR (0 downto 0);
        ADDR  : in STD_LOGIC_VECTOR (11 downto 0);
        CLK   : in STD_ULOGIC;
        DI    : in STD_LOGIC_VECTOR (0 downto 0);
        EN    : in STD_ULOGIC;
        RST   : in STD_ULOGIC;
        WE    : in STD_ULOGIC);

end component;

-- Component Attribute Specification for RAMB4_Sn
-- Should be placed after architecture declaration but before the begin keyword

-- Put attributes, if necessary


-- Component Instantiation for RAMB4_Sn
-- Should be placed in architecture after the begin keyword

RAMB4_Sn_INSTANCE_NAME : RAMB4_Sn
  -- synthesis translate_off
  generic map (
       INIT_00 => hex_value,
       INIT_01 => hex_value,
       INIT_02 => hex_value,
       INIT_03 => hex_value,
       INIT_04 => hex_value,
       INIT_05 => hex_value,
       INIT_06 => hex_value,
       INIT_07 => hex_value,
       INIT_08 => hex_value,
       INIT_09 => hex_value,
       INIT_0A => hex_value,
       INIT_0B => hex_value,
       INIT_0C => hex_value,
       INIT_0D => hex_value,
       INIT_0E => hex_value,
```

```
      INIT_0F => hex_value);
  -- synopsys translate_on
   port map (DO => user_DO,
             ADDR => user_ADDR,
             CLK => user_CLK,
             DI => user_DI,
              EN => user_EN,
             RST => user_RST,
             WE => user_WE);
```

Verilog Instantiation Template for RAMB16_S1, S2, and S4

```
RAMB4_Sn user_instance_name (.DO (user_DO),
                             .ADDR (user_ADDR),
                             .CLK (user_CLK),
                             .DI (user_DI),
                             .EN (user_EN),
                             .RST (user_RST),
                             .WE(user_WE));

    defparam user_instance_name.INIT_00 = 256_bit_hex_value;
    defparam user_instance_name.INIT_01 = 256_bit_hex_value;
    defparam user_instance_name.INIT_02 = 256_bit_hex_value;
    defparam user_instance_name.INIT_03 = 256_bit_hex_value;
    defparam user_instance_name.INIT_04 = 256_bit_hex_value;
    defparam user_instance_name.INIT_05 = 256_bit_hex_value;
    defparam user_instance_name.INIT_06 = 256_bit_hex_value;
    defparam user_instance_name.INIT_07 = 256_bit_hex_value;
    defparam user_instance_name.INIT_08 = 256_bit_hex_value;
    defparam user_instance_name.INIT_09 = 256_bit_hex_value;
    defparam user_instance_name.INIT_0A = 256_bit_hex_value;
    defparam user_instance_name.INIT_0B = 256_bit_hex_value;
    defparam user_instance_name.INIT_0C = 256_bit_hex_value;
    defparam user_instance_name.INIT_0D = 256_bit_hex_value;
    defparam user_instance_name.INIT_0E = 256_bit_hex_value;
    defparam user_instance_name.INIT_0F = 256_bit_hex_value;
```

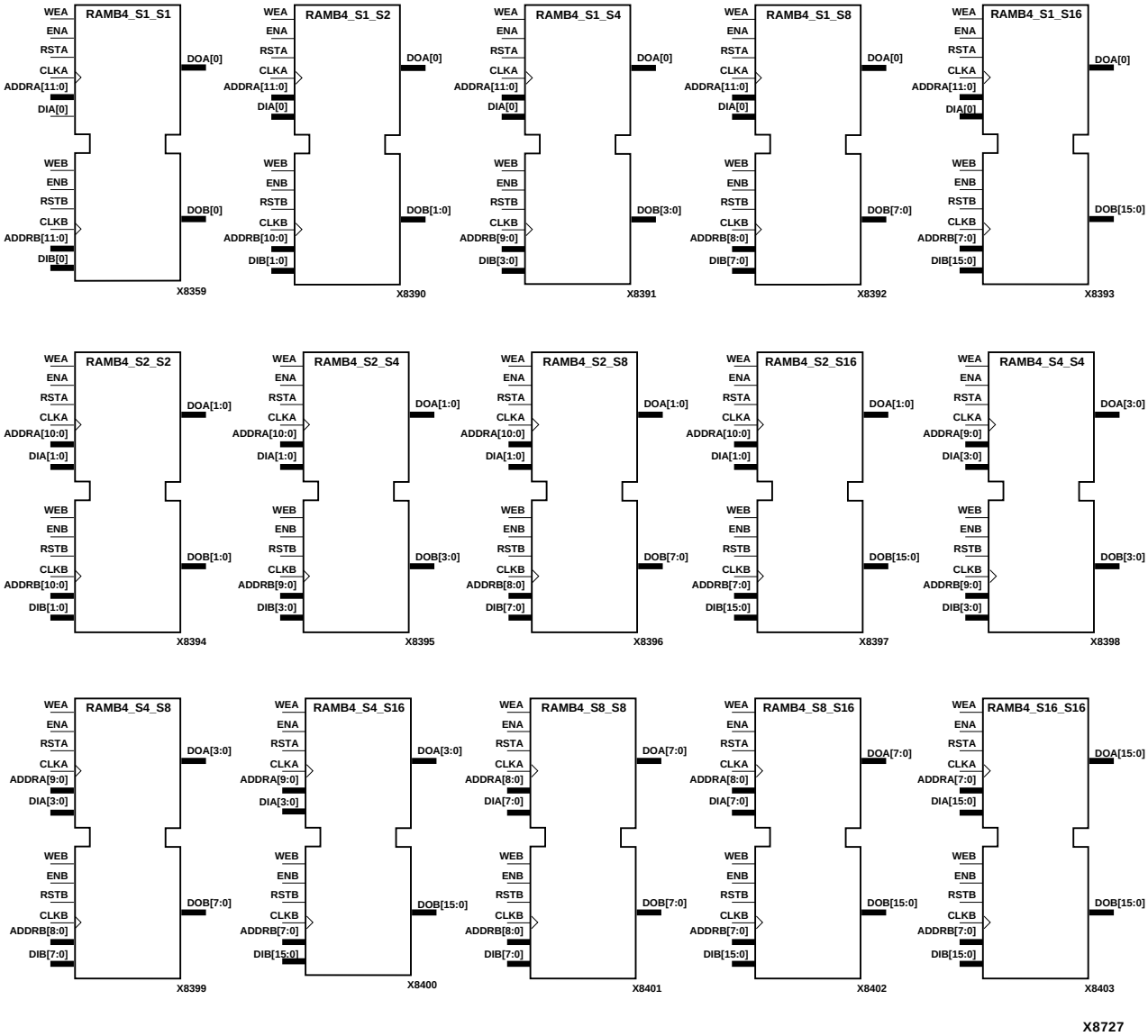
Commonly Used Constraints

INIT_xx

RAMB4_S*m*_S*n*

4096-Bit Dual-Port Synchronous Block RAM with Port Width (m or n) Configured to 1, 2, 4, 8, or 16 Bits

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	N/A	N/A	N/A	N/A



RAMB4_Sm_Sn Representations

The RAMB4_S*m*_S*n* components listed in the following table are 4096-bit dual-ported dedicated random access memory blocks with synchronous write capability. Each port is independent of the other while accessing the same set of 4096 memory cells. Each port is independently configured to a specific data width.

Component	Port A Depth	Port A Width	Port A ADDR	Port A DI	Port B Depth	Port B Width	Port B ADDR	Port B DI
RAMB4_S1_S1	4096	1	(11:0)	(0:0)	4096	1	(11:0)	(0:0)
RAMB4_S1_S2	4096	1	(11:0)	(0:0)	2048	2	(10:0)	(1:0)
RAMB4_S1_S4	4096	1	(11:0)	(0:0)	1024	4	(9:0)	(3:0)
RAMB4_S1_S8	4096	1	(11:0)	(0:0)	512	8	(8:0)	(7:0)
RAMB4_S1_S16	4096	1	(11:0)	(0:0)	256	16	(7:0)	(15:0)
RAMB4_S2_S2	2048	2	(10:0)	(1:0)	2048	2	(10:0)	(1:0)
RAMB4_S2_S4	2048	2	(10:0)	(1:0)	1024	4	(9:0)	(3:0)
RAMB4_S2_S8	2048	2	(10:0)	(1:0)	512	8	(8:0)	(7:0)
RAMB4_S2_S16	2048	2	(10:0)	(1:0)	256	16	(7:0)	(15:0)
RAMB4_S4_S4	1024	4	(9:0)	(3:0)	1024	4	(9:0)	(3:0)
RAMB4_S4_S8	1024	4	(9:0)	(3:0)	512	8	(8:0)	(7:0)
RAMB4_S4_S16	1024	4	(9:0)	(3:0)	256	16	(7:0)	(15:0)
RAMB4_S8_S8	512	8	(8:0)	(7:0)	512	8	(8:0)	(7:0)
RAMB4_S8_S16	512	8	(8:0)	(7:0)	256	16	(7:0)	(15:0)
RAMB4_S16_S16	256	16	(7:0)	(15:0)	256	16	(7:0)	(15:0)

ADDR=address bus for the port

DI=data input bus for the port

Each port is fully synchronous with independent clock pins. All port A input pins have setup time referenced to the CLKA pin and its data output bus DOA has a clock-to-out time referenced to the CLKA. All port B input pins have setup time referenced to the CLKB pin and its data output bus DOB has a clock-to-out time referenced to the CLKB.

The enable ENA pin controls read, write, and reset for port A. When ENA is Low, no data is written and the output (DOA) retains the last state. When ENA is High and reset (RSTA) is High, DOA is cleared during the Low-to-High clock (CLKA) transition; if write enable (WEA) is High, the memory contents reflect the data at DIA. When ENA is High and WEA is Low, the data stored in the RAM address (ADDRA) is read during the Low-to-High clock transition. When ENA and WEA are High, the data on the data input (DIA) is loaded into the word selected by the write address (ADDRA) during the Low-to-High clock transition and the data output (DOA) reflects the selected (addressed) word.

The enable ENB pin controls read, write, and reset for port B. When ENB is Low, no data is written and the output (DOB) retains the last state. When ENB is High and reset (RSTB) is High, DOB is cleared during the Low-to-High clock (CLKB) transition; if write enable (WEB) is High, the memory contents reflect the data at DIB. When ENB is High and WEB is Low, the data stored in the RAM address (ADDRB) is read during the Low-to-High clock transition. When ENB and WEB are High, the data on the data input (DIB) is loaded into the word selected by the write address (ADDRB) during the Low-to-High clock transition and the data output (DOB) reflects the selected (addressed) word.

The above descriptions assume active High control pins (ENA, WEA, RSTA, CLKA, ENB, WEB, RSTB, and CLKB). However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB4 port is absorbed into the block and does not use a CLB resource.

RAMB_S*m*_S*n*'s may be initialized during configuration. See the following truth table.

Block RAM output registers are asynchronously cleared, output Low, when power is applied. The initial contents of the block RAM are not altered.

Virtex, Virtex-E, Spartan-II, and Spartan-IIE simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Mode selection is shown in the following truth table.

Inputs						Outputs	
EN(A/B)	RST(A/B)	WE(A/B)	CLK(A/B)	ADDR(A/B)	DI(A/B)	DO(A/B)	RAM Contents
0	X	X	X	X	X	No Chg	No Chg
1	1	0	↑	X	X	0	No Chg
1	1	1	↑	addr	data	0	RAM(addr) =>data
1	0	0	↑	addr	X	RAM(addr)	No Chg
1	0	1	↑	addr	data	data	RAM(addr) =>data

addr=RAM address of port A/B

RAM(addr)=RAM contents at address ADDRA/ADDRB

data=RAM input data at pins DIA/DIB

Address Mapping

Each port accesses the same set of 4096 memory cells using an addressing scheme that is dependent on the width of the port. The physical RAM location that is addressed for a particular width is determined from the following formula.

```
Start=((ADDR port+1)*(Widthport)) -1
End=(ADDRport)*(Widthport)
```

The following table shows address mapping for each port width.

Port Address Mapping

Port Width	Port Addresses																	
1	4096	<-----	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
2	2048	<-----	07		06		05		04		03		02		01		00	
4	1024	<-----	03				02				01				00			
8	512	<-----	01								00							
16	256	<-----	00															

Port A and Port B Conflict Resolution

A RAMB4_S*m*_S*n* component is a true dual-ported RAM in that it allows simultaneous reads of the same memory cell. When one port is performing a write to a given memory cell, the other port should not address that memory cell (for a write or a read) within the clock-to-clock setup window.

If both ports write to the same memory cell simultaneously, violating the clock-to-setup requirement, the data stored will be invalid.

If one port attempts to read from the same memory cell that the other is simultaneously writing to, violating the clock setup requirement, the write will be successful but the data read will be invalid.

Specifying Initial Contents of a Block RAM

You can use the INIT_0x attributes to specify an initial value during device configuration. The initialization of each RAMB4_S*m*_S*n* is set by 16 initialization attributes (INIT_00 through INIT_0F) of 64 hex values for a total of 4096 bits. See the INIT_xx section of the *Constraints Guide* for more information on these attributes.

If any INIT_0x attribute is not specified, it is configured as zeros. Partial initialization strings are padded with zeros to the left.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template for RAMB4_S*m*_S*n*

```
-- Component Declaration for RAMB4_Sm_Sn
-- Should be placed after architecture statement but before begin keyword
component RAMB4_Sm_Sn
  -- synthesis translate_off
  generic (
       INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_08 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_09 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_0A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
);
-- synthesis translate_on
  port (DOA   : out STD_LOGIC_VECTOR (0 downto 0);
        DOB   : out STD_LOGIC_VECTOR (0 downto 0);
        ADDRA : in STD_LOGIC_VECTOR (11 downto 0);
        ADDRB : in STD_LOGIC_VECTOR (11 downto 0);
        CLKA  : in STD_ULOGIC;
        CLKB  : in STD_ULOGIC;
        DIA   : in STD_LOGIC_VECTOR (0 downto 0);
        DIB   : in STD_LOGIC_VECTOR (0 downto 0);
        ENA   : in STD_ULOGIC;
        ENB   : in STD_ULOGIC;
        RSTA  : in STD_ULOGIC;
        RSTB  : in STD_ULOGIC;
        WEA   : in STD_ULOGIC;
        WEB   : in STD_ULOGIC);

end component;

-- Component Attribute Specification for RAMB4_Sm_Sn
-- Should be placed after architecture declaration but before the begin keyword

-- Put attributes, if necessary


-- Component Instantiation for RAMB4_Sm_Sn
-- Should be placed in architecture after the begin keyword

RAMB4_Sm_Sn_INSTANCE_NAME : RAMB4_Sm_Sn
  -- synthesis translate_off
  generic map (
       INIT_00 => vector_value,
       INIT_01 => vector_value,
       INIT_02 => vector_value,
       INIT_03 => vector_value,
       INIT_04 => vector_value,
       INIT_05 => vector_value,
       INIT_06 => vector_value,
       INIT_07 => vector_value,
       INIT_08 => vector_value,
       INIT_09 => vector_value,
       INIT_0A => vector_value,
```

```
      INIT_0B => vector_value,
      INIT_0C => vector_value,
      INIT_0D => vector_value,
      INIT_0E => vector_value,
      INIT_0F => vector_value);
  -- synopsys translate_on
   port map (DOA => user_DOA,
             DOB => user_DOB,
             ADDRA => user_ADDRA,
             ADDRB => user_ADDRB,
             CLKA => user_CLKA,
             CLKB => user_CLKB,
             DIA => user_DIA,
             DIB => user_DIB,
             ENA => user_ENA,
             ENB => user_ENB,
             RSTA => user_RSTA,
             RSTB => user_RSTB,
             WEA => user_WEA,
             WEB => user_WEB);
```

Verilog Instantiation Template for RAMB16_S1, S2, and S4

```
RAMB4_Sm_Sn user_instance_name (.DOA (user_DOA),
                                .DOB (user_DOB),
                                .ADDRA (user_ADDRA),
                                .ADDRB (user_ADDRB),
                                .CLKA (user_CLKA),
                                .CLKB (user_CLKB),
                                .DIA (user_DIA),
                                .DIB (user_DIB),
                                .ENA (user_ENA),
                                .ENB (user_ENB),
                                .RSTA (user_RSTA),
                                .RSTB (user_RSTB),
                                .WEA (user_WEA),
                                .WEB (user_WEB));

    defparam user_instance_name.INIT_00 = 256_bit_hex_value;
    defparam user_instance_name.INIT_01 = 256_bit_hex_value;
    defparam user_instance_name.INIT_02 = 256_bit_hex_value;
    defparam user_instance_name.INIT_03 = 256_bit_hex_value;
    defparam user_instance_name.INIT_04 = 256_bit_hex_value;
    defparam user_instance_name.INIT_05 = 256_bit_hex_value;
    defparam user_instance_name.INIT_06 = 256_bit_hex_value;
    defparam user_instance_name.INIT_07 = 256_bit_hex_value;
    defparam user_instance_name.INIT_08 = 256_bit_hex_value;
    defparam user_instance_name.INIT_09 = 256_bit_hex_value;
    defparam user_instance_name.INIT_0A = 256_bit_hex_value;
    defparam user_instance_name.INIT_0B = 256_bit_hex_value;
    defparam user_instance_name.INIT_0C = 256_bit_hex_value;
    defparam user_instance_name.INIT_0D = 256_bit_hex_value;
    defparam user_instance_name.INIT_0E = 256_bit_hex_value;
    defparam user_instance_name.INIT_0F = 256_bit_hex_value;
```

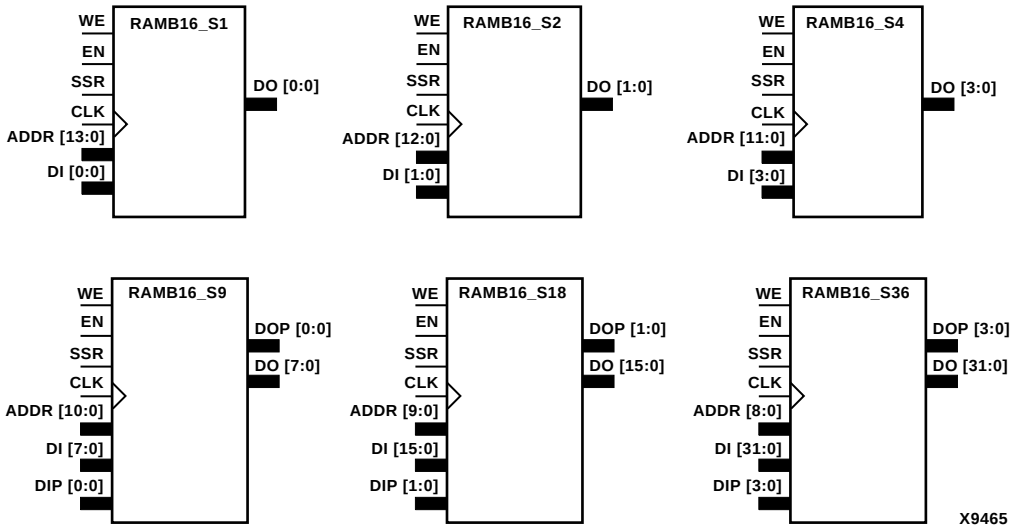
Commonly Used Constraints

INIT_xx

RAMB16_S*n*

16384-Bit Data Memory and 2048-Bit Parity Memory, Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 9, 18, or 36 Bits

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



RAMB16_S1 through RAMB16_S36 Representations

RAMB16_S1, RAMB16_S2, RAMB16_S4, RAMB16_S9, RAMB16_S18, and RAMB16_S36 are dedicated random access memory blocks with synchronous write capability. The block RAM port has 16384 bits of data memory. RAMB16_S9, RAMB16_S18, and RAMB16_S36 have an additional 2048 bits of parity memory. The RAMB16_Sn cell configurations are listed in the following table.

Component	Data Cells		Parity Cells		Address Bus	Data Bus	Parity Bus
	Depth	Width	Depth	Width			
RAMB16_S1	16384	1	-	-	(13:0)	(0:0)	-
RAMB16_S2	8192	2	-	-	(12:0)	(1:0)	-
RAMB16_S4	4096	4	-	-	(11:0)	(3:0)	-
RAMB16_S9	2048	8	2048	1	(10:0)	(7:0)	(0:0)
RAMB16_S18	1024	16	1024	2	(9:0)	(15:0)	(1:0)
RAMB16_S36	512	32	512	4	(8:0)	(31:0)	(3:0)

The enable (EN) pin controls read, write, and reset. When EN is Low, no data is written and the outputs (DO and DOP) retain the last state. When EN is High and reset (SSR) is High, DO and DOP are set to SRVAL during the Low-to-High clock (CLK) transition; if write enable (WE) is High, the memory contents reflect the data at DI and DIP. When SSR is Low, EN is High, and WE is Low, the data stored in the RAM address (ADDR) is read during the Low-to-High clock transition. The output value depends on the mode. By default WRITE_MODE=WRITE_FIRST, when EN and WE are High and SSR is Low, the data on the data inputs (DI and DIP) is loaded into the word selected by the write address (ADDR) during the Low-to-High clock transition. See the "Write Mode Selection" section for information on setting the WRITE_MODE.

The above description assumes an active High EN, WE, SSR, and CLK. However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB16 port is absorbed into the block and does not use a CLB resource.

Inputs								Outputs			
GSR	EN	SSR	WE	CLK	ADDR	DI	DIP	DO	DOP	RAM Contents	
										Data RAM	Parity RAM
1	X	X	X	X	X	X	X	INIT	INIT	No Chg	No Chg
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
0	1	1	0	↑	X	X	X	SRVAL	SRVAL	No Chg	No Chg
0	1	1	1	↑	addr	data	pdata	SRVAL	SRVAL	RAM(addr) =>data	RAM(addr) =>pdata
0	1	0	0	↑	addr	X	X	RAM(addr)	RAM(addr)	No Chg	No Chg
0	1	0	1	↑	addr	data	pdata	No Chg[a] RAM (addr)[b] data[c]	No Chg[a] RAM(addr)[b] pdata[c]	RAM(addr) =>data	RAM(addr) =>pdata

GSR=Global Set Reset signal

INIT=Value specified by the INIT attribute for data memory. Default is all zeros.
SRVAL=Value after assertion of SSR as specified by the SRVAL attribute.

addr=RAM address
RAM(addr)=RAM contents at address ADDR
data=RAM input data
pdata=RAM parity data

[a]WRITE_MODE=NO_CHANGE
[b]WRITE_MODE=READ_FIRST
[c]WRITE_MODE=WRITE_FIRST

Initializing Memory Contents of a Single-Port RAMB16

You can use the INIT_xx attributes to specify an initialization value for the memory contents of a RAMB16 during device configuration. The initialization of each RAMB16_Sn is set by 64 initialization attributes (INIT_00 through INIT_3F) of 64 hex values for a total of 16384 bits.

You can use the INITP_xx attributes to specify an initial value for the parity memory during device configuration or assertion. The initialization of the parity memory for ports configured for 9, 18, or 36 bits is set by 8 initialization attributes (INITP_00 through INITP_07) of 64 hex values for a total of 2048 bits.

If any INIT_xx or INITP_xx attribute is not specified, it is configured as zeros. Partial strings are padded with zeros to the left.

See the *Constraints Guide* for more information on these attributes.

Initializing the Output Register of a Single-Port RAMB16

In Virtex-II and Virtex-II Pro, each bit in the output register can be initialized at power on to either a 0 or 1. In addition, the initial state specified for power on can be different than the state that results from assertion of a set/reset. Two types of properties control initialization of the output register for a single-port RAMB16: INIT and SRVAL. The INIT attribute specifies the output register value at power on. You can use the SRVAL attribute to define the state resulting from assertion of the SSR (set/reset) input.

The INIT and SRVAL attributes specify the initialization value as a hexadecimal string. The value is dependent upon the port width. For example, for a RAMB16_S1 with port width equal to 1, the output register contains 1 bit. Therefore, the INIT or SRVAL value can only be specified as a 1 or 0. For RAMB16_S4 with port width equal to 4, the output register contains 4 bits. In this case, you can specify a hexadecimal value from 0 through F to initialize the 4 bits of the output register.

For those ports that include parity bits, the parity portion of the output register is specified in the high order bit position of the INIT or SRVAL value.

The INIT and SRVAL attributes default to zero if they are not set by the user.

See the *Constraints Guide* for more information on these attributes.

Write Mode Selection

The WRITE_MODE attribute controls RAMB16 memory and output contents. By default, the WRITE_MODE is set to WRITE_FIRST. This means that input is read, written to memory, and then passed to output. You can set the WRITE_MODE to READ_FIRST to read the memory contents, pass the memory contents to the outputs, and then write the input to memory. Or, you can set the WRITE_MODE to NO_CHANGE to have the input written to memory without changing the output.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template for RAMB16_S1, S2, and S4

```
-- Component Declaration for RAMB16_{S1 | S2 | S4}
-- Should be placed after architecture statement but before begin keyword
component RAMB16_{S1 | S2 | S4}
  -- synthesis translate_off
  generic (
      INIT : bit_vector   := X"0";
      INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_08 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_09 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_10 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_11 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_12 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_13 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_14 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_15 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_16 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_17 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_18 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_19 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_20 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_21 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_22 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_23 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_24 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_25 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_26 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_27 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_28 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_29 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_30 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_31 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_32 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_33 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_34 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_35 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_36 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_37 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_38 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_39 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       SRVAL : bit_vector  := X"0";
       WRITE_MODE : string := "WRITE_FIRST");

  -- synthesis translate_on
  port (DO     : out STD_LOGIC_VECTOR (0 downto 0)
        ADDR   : in STD_LOGIC_VECTOR (13 downto 0);
        CLK    : in STD_ULOGIC;
        DI     : in STD_LOGIC_VECTOR (0 downto 0);
        EN     : in STD_ULOGIC;
        SSR    : in STD_ULOGIC;
        WE     : in STD_ULOGIC);

end component;

-- Component Attribute Specification for RAMB16_{S1 | S2 | S4}
-- Should be placed after architecture declaration but before the begin keyword

-- Put attributes, if necessary


-- Component Instantiation for RAMB16_{S1 | S2 | S4}
-- Should be placed in architecture after the begin keyword

RAMB16_{S1 | S2 | S4}_INSTANCE_NAME : RAMB16_S1
  -- synthesis translate_off
  generic map (
```

```
INIT => bit_value,
INIT_00 => vector_value,
INIT_01 => vector_value,
INIT_02 => vector_value,
INIT_03 => vector_value,
INIT_04 => vector_value,
INIT_05 => vector_value,
INIT_06 => vector_value,
INIT_07 => vector_value,
INIT_08 => vector_value,
INIT_09 => vector_value,
INIT_0A => vector_value,
INIT_0B => vector_value,
INIT_0C => vector_value,
INIT_0D => vector_value,
INIT_0E => vector_value,
INIT_0F => vector_value,
INIT_10 => vector_value,
INIT_11 => vector_value,
INIT_12 => vector_value,
INIT_13 => vector_value,
INIT_14 => vector_value,
INIT_15 => vector_value,
INIT_16 => vector_value,
INIT_17 => vector_value,
INIT_18 => vector_value,
INIT_19 => vector_value,
INIT_1A => vector_value,
INIT_1B => vector_value,
INIT_1C => vector_value,
INIT_1D => vector_value,
INIT_1E => vector_value,
INIT_1F => vector_value,
INIT_20 => vector_value,
INIT_21 => vector_value,
INIT_22 => vector_value,
INIT_23 => vector_value,
INIT_24 => vector_value,
INIT_25 => vector_value,
INIT_26 => vector_value,
INIT_27 => vector_value,
INIT_28 => vector_value,
INIT_29 => vector_value,
INIT_2A => vector_value,
INIT_2B => vector_value,
INIT_2C => vector_value,
INIT_2D => vector_value,
INIT_2E => vector_value,
INIT_2F => vector_value,
INIT_30 => vector_value,
INIT_31 => vector_value,
INIT_32 => vector_value,
INIT_33 => vector_value,
INIT_34 => vector_value,
```

```
      INIT_35 => vector_value,
      INIT_36 => vector_value,
      INIT_37 => vector_value,
      INIT_38 => vector_value,
      INIT_39 => vector_value,
      INIT_3A => vector_value,
      INIT_3B => vector_value,
      INIT_3C => vector_value,
      INIT_3D => vector_value,
      INIT_3E => vector_value,
      INIT_3F => vector_value,
      SRVAL=> bit_value,
      WRITE_MODE => user_WRITE_MODE)
  -- synopsys translate_on
   port map (DO => user_DO,
             ADDR => user_ADDR,
             CLK => user_CLK,
             DI => user_DI,
              EN => user_EN,
             SSR => user_SSR,
             WE => user_WE);
```

Verilog Instantiation Template for RAMB16_S1, S2, and S4

```
RAMB16_{S1 | S2 | S4} user_instance_name (.DO(user_DO),
                                          .ADDR(user_ADDR),
                                          .CLK(user_CLK),
                                          .DI(user_DI),
                                          .EN(user_EN),
                                          .SSR(user_SSR),
                                          .WE(user_WE));

    defparam user_instance_name.INIT = bit_value;
    defparam user_instance_name.INIT_00 = 256_bit_hex_value;
    defparam user_instance_name.INIT_01 = 256_bit_hex_value;
    defparam user_instance_name.INIT_02 = 256_bit_hex_value;
    defparam user_instance_name.INIT_03 = 256_bit_hex_value;
    defparam user_instance_name.INIT_04 = 256_bit_hex_value;
    defparam user_instance_name.INIT_05 = 256_bit_hex_value;
    defparam user_instance_name.INIT_06 = 256_bit_hex_value;
    defparam user_instance_name.INIT_07 = 256_bit_hex_value;
    defparam user_instance_name.INIT_08 = 256_bit_hex_value;
    defparam user_instance_name.INIT_09 = 256_bit_hex_value;
    defparam user_instance_name.INIT_0A = 256_bit_hex_value;
    defparam user_instance_name.INIT_0B = 256_bit_hex_value;
    defparam user_instance_name.INIT_0C = 256_bit_hex_value;
    defparam user_instance_name.INIT_0D = 256_bit_hex_value;
    defparam user_instance_name.INIT_0E = 256_bit_hex_value;
    defparam user_instance_name.INIT_0F = 256_bit_hex_value;
    defparam user_instance_name.INIT_10 = 256_bit_hex_value;
    defparam user_instance_name.INIT_11 = 256_bit_hex_value;
    defparam user_instance_name.INIT_12 = 256_bit_hex_value;
    defparam user_instance_name.INIT_13 = 256_bit_hex_value;
    defparam user_instance_name.INIT_14 = 256_bit_hex_value;
```

```
defparam user_instance_name.INIT_15 = 256_bit_hex_value;
defparam user_instance_name.INIT_16 = 256_bit_hex_value;
defparam user_instance_name.INIT_17 = 256_bit_hex_value;
defparam user_instance_name.INIT_18 = 256_bit_hex_value;
defparam user_instance_name.INIT_19 = 256_bit_hex_value;
defparam user_instance_name.INIT_1A = 256_bit_hex_value;
defparam user_instance_name.INIT_1B = 256_bit_hex_value;
defparam user_instance_name.INIT_1C = 256_bit_hex_value;
defparam user_instance_name.INIT_1D = 256_bit_hex_value;
defparam user_instance_name.INIT_1E = 256_bit_hex_value;
defparam user_instance_name.INIT_1F = 256_bit_hex_value;
defparam user_instance_name.INIT_20 = 256_bit_hex_value;
defparam user_instance_name.INIT_21 = 256_bit_hex_value;
defparam user_instance_name.INIT_22 = 256_bit_hex_value;
defparam user_instance_name.INIT_23 = 256_bit_hex_value;
defparam user_instance_name.INIT_24 = 256_bit_hex_value;
defparam user_instance_name.INIT_25 = 256_bit_hex_value;
defparam user_instance_name.INIT_26 = 256_bit_hex_value;
defparam user_instance_name.INIT_27 = 256_bit_hex_value;
defparam user_instance_name.INIT_28 = 256_bit_hex_value;
defparam user_instance_name.INIT_29 = 256_bit_hex_value;
defparam user_instance_name.INIT_2A = 256_bit_hex_value;
defparam user_instance_name.INIT_2B = 256_bit_hex_value;
defparam user_instance_name.INIT_2C = 256_bit_hex_value;
defparam user_instance_name.INIT_2D = 256_bit_hex_value;
defparam user_instance_name.INIT_2E = 256_bit_hex_value;
defparam user_instance_name.INIT_2F = 256_bit_hex_value;
defparam user_instance_name.INIT_30 = 256_bit_hex_value;
defparam user_instance_name.INIT_31 = 256_bit_hex_value;
defparam user_instance_name.INIT_32 = 256_bit_hex_value;
defparam user_instance_name.INIT_33 = 256_bit_hex_value;
defparam user_instance_name.INIT_34 = 256_bit_hex_value;
defparam user_instance_name.INIT_35 = 256_bit_hex_value;
defparam user_instance_name.INIT_36 = 256_bit_hex_value;
defparam user_instance_name.INIT_37 = 256_bit_hex_value;
defparam user_instance_name.INIT_38 = 256_bit_hex_value;
defparam user_instance_name.INIT_39 = 256_bit_hex_value;
defparam user_instance_name.INIT_3A = 256_bit_hex_value;
defparam user_instance_name.INIT_3B = 256_bit_hex_value;
defparam user_instance_name.INIT_3C = 256_bit_hex_value;
defparam user_instance_name.INIT_3D = 256_bit_hex_value;
defparam user_instance_name.INIT_3E = 256_bit_hex_value;
defparam user_instance_name.INIT_3F = 256_bit_hex_value;
defparam user_instance_name.SRVAL = bit_value;
defparam user_instance_name.WRITE_MODE = write_mode;
```

VHDL Instantiation Template for RAMB16_S9, S18 and S36

```
-- Component Declaration for RAMB16_{S9 | S18 | S36}
-- Should be placed after architecture statement but before begin keyword
component RAMB16_{S9 | S18 | S36}
  -- synthesis translate_off
  generic (
      INIT : bit_vector   := X"0";
      INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_08 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_09 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_10 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_11 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_12 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_13 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_14 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_15 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_16 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_17 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_18 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_19 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_20 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_21 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_22 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_23 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_24 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_25 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_26 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_27 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_28 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_29 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_30 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_31 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_32 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_33 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_34 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_35 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_36 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_37 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_38 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_39 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       SRVAL : bit_vector  := X"0";
       WRITE_MODE : string := "WRITE_FIRST";
);
  -- synthesis translate_on

  port (DO     : out STD_LOGIC_VECTOR (0 downto 0);
        DOP    : out STD_LOGIC_VECTOR (1 downto 0);
        ADDR   : in STD_LOGIC_VECTOR (13 downto 0);
        CLK    : in STD_ULOGIC;
        DI     : in STD_LOGIC_VECTOR (0 downto 0);
```

```
        DIP    : in STD_LOGIC_VECTOR (0 downto 0);
        EN     : in STD_ULOGIC;
        SSR    : in STD_ULOGIC;
        WE     : in STD_ULOGIC);

end component;

-- Component Attribute Specification for RAMB16_{S9 | S18 | S36}
-- Should be placed after architecture declaration but before the begin keyword

-- Put attributes, if necessary


-- Component Instantiation for RAMB16_{S9 | S18 | S36}
-- Should be placed in architecture after the begin keyword

RAMB16_{S9 | S18 | S36}_INSTANCE_NAME : RAMB16_S1
  -- synthesis translate_off
  generic map (
       INIT => bit_value,
       INIT_00 => vector_value,
       INIT_01 => vector_value,
       INIT_02 => vector_value,
       INIT_03 => vector_value,
       INIT_04 => vector_value,
       INIT_05 => vector_value,
       INIT_06 => vector_value,
       INIT_07 => vector_value,
       INIT_08 => vector_value,
       INIT_09 => vector_value,
       INIT_0A => vector_value,
       INIT_0B => vector_value,
       INIT_0C => vector_value,
       INIT_0D => vector_value,
       INIT_0E => vector_value,
       INIT_0F => vector_value,
       INIT_10 => vector_value,
       INIT_11 => vector_value,
       INIT_12 => vector_value,
       INIT_13 => vector_value,
       INIT_14 => vector_value,
       INIT_15 => vector_value,
       INIT_16 => vector_value,
       INIT_17 => vector_value,
       INIT_18 => vector_value,
       INIT_19 => vector_value,
       INIT_1A => vector_value,
       INIT_1B => vector_value,
       INIT_1C => vector_value,
       INIT_1D => vector_value,
       INIT_1E => vector_value,
       INIT_1F => vector_value,
       INIT_20 => vector_value,
       INIT_21 => vector_value,
```

```
      INIT_22 => vector_value,
      INIT_23 => vector_value,
      INIT_24 => vector_value,
      INIT_25 => vector_value,
      INIT_26 => vector_value,
      INIT_27 => vector_value,
      INIT_28 => vector_value,
      INIT_29 => vector_value,
      INIT_2A => vector_value,
      INIT_2B => vector_value,
      INIT_2C => vector_value,
      INIT_2D => vector_value,
      INIT_2E => vector_value,
      INIT_2F => vector_value,
      INIT_30 => vector_value,
      INIT_31 => vector_value,
      INIT_32 => vector_value,
      INIT_33 => vector_value,
      INIT_34 => vector_value,
      INIT_35 => vector_value,
      INIT_36 => vector_value,
      INIT_37 => vector_value,
      INIT_38 => vector_value,
      INIT_39 => vector_value,
      INIT_3A => vector_value,
      INIT_3B => vector_value,
      INIT_3C => vector_value,
      INIT_3D => vector_value,
      INIT_3E => vector_value,
      INIT_3F => vector_value,
      INITP_00 => vector_value,
      INITP_01 => vector_value,
      INITP_02 => vector_value,
      INITP_03 => vector_value,
      INITP_04 => vector_value,
      INITP_05 => vector_value,
      INITP_06 => vector_value,
      INITP_07 => vector_value
      SRVAL=> bit_value,
      WRITE_MODE => user_WRITE_MODE)
  -- synopsys translate_on
   port map (DO => user_DO,
             DOP => user_DOP,
             ADDR => user_ADDR,
             CLK => user_CLK,
             DI => user_DI,
             DIP => user_DIP,
              EN => user_EN,
             SSR => user_SSR,
             WE => user_WE);
```

Verilog Instantiation Template for RAMB16_S18 and S36

```
RAMB16_{S9 | S18 | S36} user_instance_name (.DO(user_DO),
                                            .DOP (user_DOP),
                                            .ADDR (user_ADDR),
                                            .CLK (user_CLK),
                                            .DI (user_DI),
                                            .DIP (user_DIP),
                                            .EN (user_EN),
                                            .SSR (user_SSR),
                                            .WE (user_WE));

    defparam user_instance_name.INIT = bit_value;
    defparam user_instance_name.INIT_00 = 256_bit_hex_value;
    defparam user_instance_name.INIT_01 = 256_bit_hex_value;
    defparam user_instance_name.INIT_02 = 256_bit_hex_value;
    defparam user_instance_name.INIT_03 = 256_bit_hex_value;
    defparam user_instance_name.INIT_04 = 256_bit_hex_value;
    defparam user_instance_name.INIT_05 = 256_bit_hex_value;
    defparam user_instance_name.INIT_06 = 256_bit_hex_value;
    defparam user_instance_name.INIT_07 = 256_bit_hex_value;
    defparam user_instance_name.INIT_08 = 256_bit_hex_value;
    defparam user_instance_name.INIT_09 = 256_bit_hex_value;
    defparam user_instance_name.INIT_0A = 256_bit_hex_value;
    defparam user_instance_name.INIT_0B = 256_bit_hex_value;
    defparam user_instance_name.INIT_0C = 256_bit_hex_value;
    defparam user_instance_name.INIT_0D = 256_bit_hex_value;
    defparam user_instance_name.INIT_0E = 256_bit_hex_value;
    defparam user_instance_name.INIT_0F = 256_bit_hex_value;
    defparam user_instance_name.INIT_10 = 256_bit_hex_value;
    defparam user_instance_name.INIT_11 = 256_bit_hex_value;
    defparam user_instance_name.INIT_12 = 256_bit_hex_value;
    defparam user_instance_name.INIT_13 = 256_bit_hex_value;
    defparam user_instance_name.INIT_14 = 256_bit_hex_value;
    defparam user_instance_name.INIT_15 = 256_bit_hex_value;
    defparam user_instance_name.INIT_16 = 256_bit_hex_value;
    defparam user_instance_name.INIT_17 = 256_bit_hex_value;
    defparam user_instance_name.INIT_18 = 256_bit_hex_value;
    defparam user_instance_name.INIT_19 = 256_bit_hex_value;
    defparam user_instance_name.INIT_1A = 256_bit_hex_value;
    defparam user_instance_name.INIT_1B = 256_bit_hex_value;
    defparam user_instance_name.INIT_1C = 256_bit_hex_value;
    defparam user_instance_name.INIT_1D = 256_bit_hex_value;
    defparam user_instance_name.INIT_1E = 256_bit_hex_value;
    defparam user_instance_name.INIT_1F = 256_bit_hex_value;
    defparam user_instance_name.INIT_20 = 256_bit_hex_value;
    defparam user_instance_name.INIT_21 = 256_bit_hex_value;
    defparam user_instance_name.INIT_22 = 256_bit_hex_value;
    defparam user_instance_name.INIT_23 = 256_bit_hex_value;
    defparam user_instance_name.INIT_24 = 256_bit_hex_value;
    defparam user_instance_name.INIT_25 = 256_bit_hex_value;
    defparam user_instance_name.INIT_26 = 256_bit_hex_value;
    defparam user_instance_name.INIT_27 = 256_bit_hex_value;
    defparam user_instance_name.INIT_28 = 256_bit_hex_value;
```

```
defparam user_instance_name.INIT_29 = 256_bit_hex_value;
defparam user_instance_name.INIT_2A = 256_bit_hex_value;
defparam user_instance_name.INIT_2B = 256_bit_hex_value;
defparam user_instance_name.INIT_2C = 256_bit_hex_value;
defparam user_instance_name.INIT_2D = 256_bit_hex_value;
defparam user_instance_name.INIT_2E = 256_bit_hex_value;
defparam user_instance_name.INIT_2F = 256_bit_hex_value;
defparam user_instance_name.INIT_30 = 256_bit_hex_value;
defparam user_instance_name.INIT_31 = 256_bit_hex_value;
defparam user_instance_name.INIT_32 = 256_bit_hex_value;
defparam user_instance_name.INIT_33 = 256_bit_hex_value;
defparam user_instance_name.INIT_34 = 256_bit_hex_value;
defparam user_instance_name.INIT_35 = 256_bit_hex_value;
defparam user_instance_name.INIT_36 = 256_bit_hex_value;
defparam user_instance_name.INIT_37 = 256_bit_hex_value;
defparam user_instance_name.INIT_38 = 256_bit_hex_value;
defparam user_instance_name.INIT_39 = 256_bit_hex_value;
defparam user_instance_name.INIT_3A = 256_bit_hex_value;
defparam user_instance_name.INIT_3B = 256_bit_hex_value;
defparam user_instance_name.INIT_3C = 256_bit_hex_value;
defparam user_instance_name.INIT_3D = 256_bit_hex_value;
defparam user_instance_name.INIT_3E = 256_bit_hex_value;
defparam user_instance_name.INIT_3F = 256_bit_hex_value;
defparam user_instance_name.INITP_00 = 256_bit_hex_value;
defparam user_instance_name.INITP_01 = 256_bit_hex_value;
defparam user_instance_name.INITP_02 = 256_bit_hex_value;
defparam user_instance_name.INITP_03 = 256_bit_hex_value;
defparam user_instance_name.INITP_04 = 256_bit_hex_value;
defparam user_instance_name.INITP_05 = 256_bit_hex_value;
defparam user_instance_name.INITP_06 = 256_bit_hex_value;
defparam user_instance_name.INITP_07 = 256_bit_hex_value;
defparam user_instance_name.SRVAL = bit_value;
defparam user_instance_name.WRITE_MODE = write_mode;
```

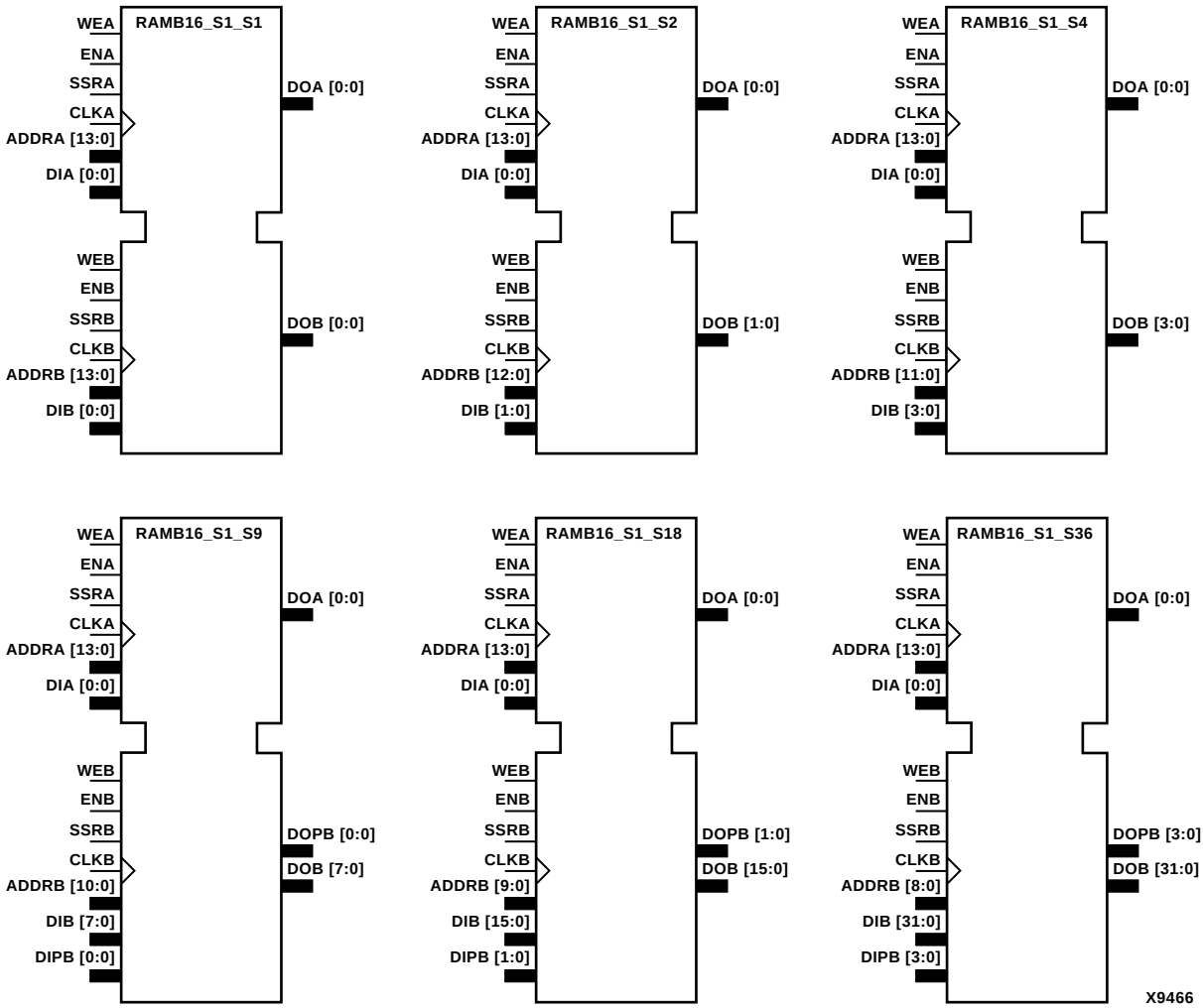
Commonly Used Constraints

INIT, INIT_xx, SRVAL, WRITE_MODE, HU_SET, INITP_xx, SRVAL, WRITE_MODE

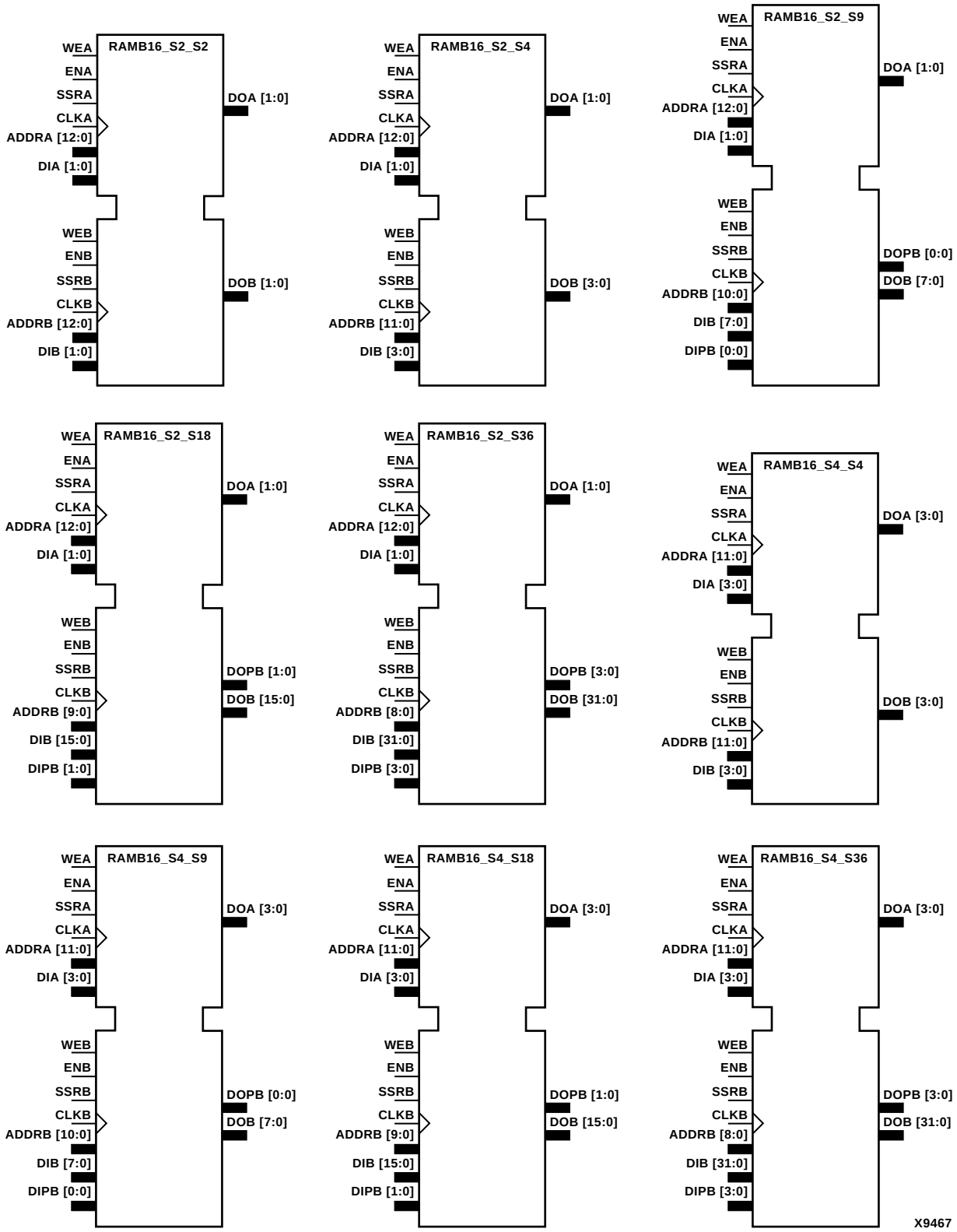
RAMB16_S*m*_S*n*

16384-Bit Data Memory and 2048-Bit Parity Memory, Dual-Port Synchronous Block RAM with Port Width (m or n) Configured to 1, 2, 4, 9, 18, or 36 Bits

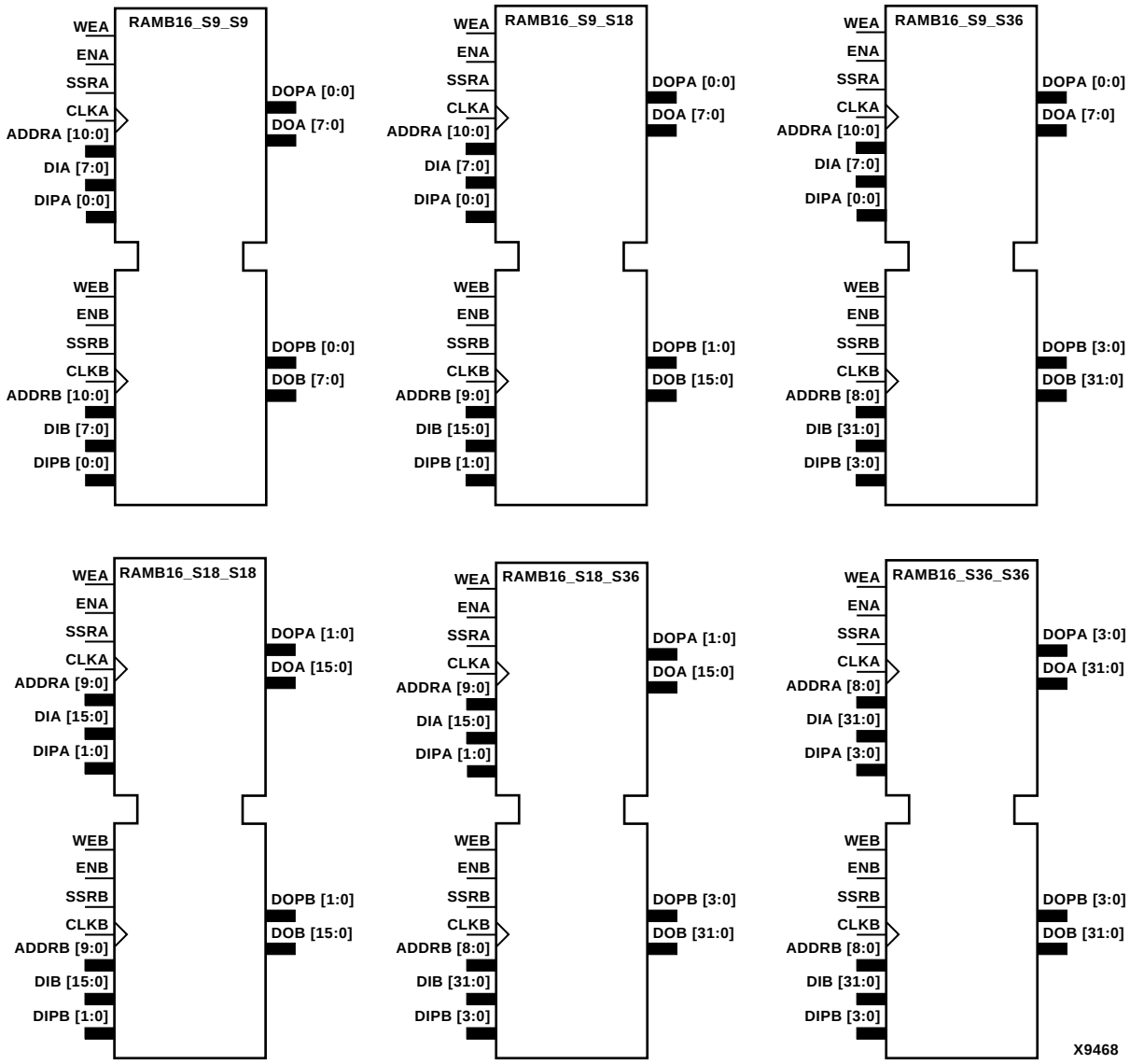
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



RAMB16_S1_S1 through RAMB16_S1_S36 Representations



RAMB16_S2_S2 through RAMB16_S4_S36 Representations



RAMB16_S9_S9 through RAMB16_S36_S36 Representations

The RAMB16_S*m*_S*n* components listed in the following table are dual-ported dedicated random access memory blocks with synchronous write capability. Each block RAM port has 16384 bits of data memory. Ports configured as 9, 18, or 36-bits wide have an additional 2048 bits of parity memory. Each port is independent of the other while accessing the same set of 16384 data memory cells. Each port is independently configured to a specific data width. The possible port and cell configurations are listed in the following table.

Component	Port A					Port B				
	Data Cells[a]	**Parity Cells**[a]	Address Bus	Data Bus	Parity Bus	**Data Cells**[a]	**Parity Cells**[a]	Address Bus	Data Bus	Parity Bus
RAMB16_S1_S1	16384 x 1	-	(13:0)	(0:0)	-	16384 x 1	-	(13:0)	(0:0)	-
RAMB16_S1_S2	16384 x 1	-	(13:0)	(0:0)	-	8192 x 2	-	(12:0)	(1:0)	-
RAMB16_S1_S4	16384 x 1	-	(13:0)	(0:0)	-	4096 x 4	-	(11:0)	(3:0)	-
RAMB16_S1_S9	16384 x 1	-	(13:0)	(0:0)	-	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)
RAMB16_S1_S18	16384 x 1	-	(13:0)	(0:0)	-	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S1_S36	16384 x 1	-	(13:0)	(0:0)	-	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S2_S2	8192 x 2	-	(12:0)	(1:0)	-	8192 x 2	-	(12:0)	(1:0)	-
RAMB16_S2_S4	8192 x 2	-	(12:0)	(1:0)	-	4096 x 4	-	(11:0)	(3:0)	-
RAMB16_S2_S9	8192 x 2	-	(12:0)	(1:0)	-	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)
RAMB16_S2_S18	8192 x 2	-	(12:0)	(1:0)	-	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S2_S36	8192 x 2	-	(12:0)	(1:0)	-	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S4_S4	4096 x 4	-	(11:0)	(3:0)	-	4096 x 4	-	(11:0)	(3:0)	-
RAMB16_S4_S9	4096 x 4	-	(11:0)	(3:0)	-	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)
RAMB16_S4_S18	4096 x 4	-	(11:0)	(3:0)	-	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S4_S36	4096 x 4	-	(11:0)	(3:0)	-	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S9_S9	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)
RAMB16_S9_S18	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S9_S36	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S18_S18	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S18_S36	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S36_S36	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)

[a] Depth x Width

Each port is fully synchronous with independent clock pins. All port A input pins have setup time referenced to the CLKA pin and its data output bus DOA has a clock-to-out time referenced to the CLKA. All port B input pins have setup time referenced to the CLKB pin and its data output bus DOB has a clock-to-out time referenced to the CLKB.

The enable ENA pin controls read, write, and reset for port A. When ENA is Low, no data is written and the outputs (DOA and DOPA) retain the last state. When ENA is High and reset (SSRA) is High, DOA and DOPA are set to SRVAL_A during the Low-to-High clock (CLKA) transition; if write enable (WEA) is High, the memory contents reflect the data at DIA and DIPA. When ENA is High and WEA is Low, the data stored in the RAM address (ADDRA) is read during the Low-to-High clock transition. By default, WRITE_MODE_A=WRITE_FIRST, when ENA and WEA are High, the data on the data inputs (DIA and DIPA) is loaded into the word selected by the write address (ADDRA) during the Low-to-High clock transition and the data outputs (DOA and DOPA) reflect the selected (addressed) word.

The enable ENB pin controls read, write, and reset for port B. When ENB is Low, no data is written and the outputs (DOB and DOPB) retain the last state. When ENB is High and reset (SSRB) is High, DOB and DOPB are set to SRVAL_B during the Low-to-High clock (CLKB) transition; if write enable (WEB) is High, the memory contents reflect the data at DIB and DIPB. When ENB is High and WEB is Low, the data stored in the RAM address (ADDRB) is read during the Low-to-High clock transition. By default, WRITE_MODE_B=WRITE_FIRST, when ENB and WEB are High, the data on the data inputs (DIB and PB) are loaded into the word selected by the write address (ADDRB) during the Low-to-High clock transition and the data outputs (DOB and DOPB) reflect the selected (addressed) word.

The above descriptions assume active High control pins (ENA, WEA, SSRA, CLKA, ENB, WEB, SSRB, and CLKB). However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB16 port is absorbed into the block and does not use a CLB resource.

Port A Truth Table

Inputs								Outputs			
GSR	ENA	SSRA	WEA	CLKA	ADDRA	DIA	DIPA	DOA	DOPA	RAM Contents	
										Data RAM	Parity RAM
1	X	X	X	X	X	X	X	INIT_A	INIT_A	No Chg	No Chg
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
0	1	1	0	↑	X	X	X	SRVAL_A	SRVAL_A	No Chg	No Chg
0	1	1	1	↑	addr	data	pdata	SRVAL_A	SRVAL_A	RAM(addr) =>data	RAM(addr) =>pdata
0	1	0	0	↑	addr	X	X	RAM(addr)	RAM(addr)	No Chg	No Chg

Port A Truth Table

Inputs								Outputs			
GSR	ENA	SSRA	WEA	CLKA	ADDRA	DIA	DIPA	DOA	DOPA	RAM Contents	
0	1	0	1	↑	addr	data	pdata	No Chg[1] RAM (addr)[2] data[3]	No Chg[1] RAM(addr)[2] pdata[3]	RAM(addr) =>data	RAM(addr) =>pdata

GSR=Global Set Reset

INIT_A=Value specified by the INIT_A attribute for output register. Default is all zeros.
SRVAL_A=register value

addr=RAM address
RAM(addr)=RAM contents at address ADDR
data=RAM input data
pdata=RAM parity data

[1]WRITE_MODE_A=NO_CHANGE
[2]WRITE_MODE_A=READ_FIRST
[3]WRITE_MODE_A=WRITE_FIRST

Port B Truth Table

Inputs								Outputs			
GSR	ENB	SSRB	WEB	CLKB	ADDRB	DIB	DIPB	DOB	DOPB	RAM Contents	
										Data RAM	Parity RAM
1	X	X	X	X	X	X	X	INIT_B	INIT_B	No Chg	No Chg
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
0	1	1	0	↑	X	X	X	SRVAL_B	SRVAL_B	No Chg	No Chg
0	1	1	1	↑	addr	data	pdata	SRVAL_B	SRVAL_B	RAM(addr) =>data	RAM(addr) =>pdata
0	1	0	0	↑	addr	X	X	RAM(addr)	RAM(addr)	No Chg	No Chg
0	1	0	1	↑	addr	data	pdata	No Chg[1] RAM (addr)[2] data[3]	No Chg[1] RAM(addr)[2] pdata[3]	RAM(addr) =>data	RAM(addr) =>pdata

GSR=Global Set Reset

INIT_B=Value specified by the INIT_B attribute for output registers. Default is all zeros.
SRVAL_B=register value

addr=RAM address
RAM(addr)=RAM contents at address ADDR
data=RAM input data
pdata=RAM parity data

[1]WRITE_MODE_B=NO_CHANGE
[2]WRITE_MODE_B=READ_FIRST
[3]WRITE_MODE_B=WRITE_FIRST

Address Mapping

Each port accesses the same set of 18432 memory cells using an addressing scheme that is dependent on the width of the port. For all port widths, 16384 memory cells are available for data as shown in the "Port Address Mapping for Data" table . For 9-, 18-, and 36-bit wide ports, 2408 parity memory cells are also available as shown in "Port Address Mapping for Parity" table. The physical RAM location that is addressed for a particular width is determined from the following formula.

```
Start=((ADDR port+1)*(Widthport)) -1
End=(ADDRport)*(Widthport)
```

The following tables shows address mapping for each port width.

Port Address Mapping for Data

Data Width	Port Data Addresses																															
1	16384	<--	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
2	8192	<--	15		14		13		12		11		10		09		08		07		06		05		04		03		02		01		00	
4	4096	<--	07				06				05				04				03				02				01				00	
8	2048	<--	03								02								01								00					
16	1024	<--	01																00													
32	512	<--	00																													

Port Address Mapping for Parity

Parity Width	Port Parity Addresses					
1	2048	<-----	03	02	01	00
2	1024	<-----	01		00	
4	512	<-----	00			

Initializing Memory Contents of a Dual-Port RAMB16

You can use the INIT_xx attributes to specify an initialization value for the memory contents of a RAMB16 during device configuration. The initialization of each RAMB16_Sm_Sn is set by 64 initialization attributes (INIT_00 through INIT_3F) of 64 hex values for a total of 16384 bits.

You can use the INITP_xx attributes to specify an initial value for the parity memory during device configuration or assertion. The initialization of the parity memory for ports configured for 9, 18, or 36 bits is set by 8 initialization attributes (INITP_00 through INITP_07) of 64 hex values for a total of 2048 bits.

If any INIT_xx or INITP_xx attribute is not specified, it is configured as zeros. Partial strings are padded with zeros to the left.

See the *Constraints Guide* for more information on these attributes.

Initializing the Output Register of a Dual-Port RAMB16

In Virtex-II and Virtex-II Pro, each bit in an output register can be initialized at power on (when GSR is high) to either a 0 or 1. In addition, the initial state specified for power on can be different than the state that results from assertion of a set/reset. Four properties control initialization of the output register for a dual-port RAMB16: INIT_A, INIT_B, SRVAL_A, and SRVAL_B. The INIT_A attribute specifies the output register value at power on for port A and the INIT_B attribute specifies the value for port B. You can use the SRVAL_A attribute to define the state resulting from assertion of the SSR (set/reset) input on port A. You can use the SRVAL_B attribute to define the state resulting from assertion of the SSR input on port B.

The INIT_A, INIT_B, SRVAL_A, and SRVAL_B attributes specify the initialization value as a hexadecimal string. The value is dependent upon the port width. For example, for a RAMB16_S1_S4 with port A width equal to 1 and port B width equal to 4, the port A output register contains 1 bit and the port B output register contains 4 bits. Therefore, the INIT_A or SRVAL_A value can only be specified as a 1 or 0. For port B, the output register contains 4 bits. In this case, you can use INIT_B or SRVAL_B to specify a hexadecimal value from 0 through F to initialize the 4 bits of the output register.

For those ports that include parity bits, the parity portion of the output register is specified in the high order bit position of the INIT_A, INIT_B, SRVAL_A, or SRVAL_B value.

The INIT and SRVAL attributes default to zero if they are not set by the user.

See the *Constraints Guide* for more information on these attributes.

Write Mode Selection

The WRITE_MODE_A attribute controls the memory and output contents of port A for a dual-port RAMB16. The WRITE_MODE_B attribute does the same for port B. By default, both WRITE_MODE_A and WRITE_MODE_B are set to WRITE_FIRST. This means that input is read, written to memory, and then passed to output. You can set the write mode for port A and/or port B to READ_FIRST to read the memory contents, pass the memory contents to the outputs, and then write the input to memory. Or, you can set the write mode to NO_CHANGE to have the input written to memory without changing the output. The “Port A and Port B Conflict Resolution” section describes how read/write conflicts are resolved when both port A and port B are attempting to read/write to the same memory cells.

Port A and Port B Conflict Resolution

Virtex-II and Virtex-II Pro block SelectRAM is True Dual-Port RAM that allows both ports to simultaneously access the same memory cell. When one port writes to a given memory cell, the other port must not address that memory cell (for a write or a read) within the clock-to-clock setup window. For a list of specifics of conflict resolution for port and memory cell write operations that have either a clock common to both ports or synchronous clocks on each port, see *Virtex-II Handbook, Chapter 2, Design Considerations, Using BlockSelectRAM Memory, Conflict Resolution.*

The following tables summarize the collision detection behavior of the dual-port RAMB16 based on the WRITE_MODE_A and WRITE_MODE_B settings.

WRITE_MODE_A=NO_CHANGE and WRITE_MODE_B=NO_CHANGE

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	X	No Chg	X	No Chg	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	No Chg	No Chg	No Chg	X	X

WRITE_MODE_A=READ_FIRST and WRITE_MODE_B=READ_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	X	X

WRITE_MODE_A= WRITE_FIRST and WRITE_MODE_B=WRITE_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	DIA	X	DIPA	X	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	X	DIB	X	DIPB	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	X	X	X	X	X	X

WRITE_MODE_A=NO_CHANGE and WRITE_MODE_B=WRITE_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	DIB	DIPB

WRITE_MODE_A=NO_CHANGE and WRITE_MODE_B=WRITE_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	X	DIB	X	DIPB	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	X	X

WRITE_MODE_A=READ_FIRST and WRITE_MODE_B=WRITE_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	X	DIB	X	DIPB	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	X	DIB	X	DIPB	DIA	DIPA

Usage

For HDL, these design elements can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, please refer to the *XST User Guide*.

VHDL Instantiation Template for RAMB16_S1_S1, RAMB16_S1_S2, RAMB16_S1_S4, RAMB16_S2_S2, RAMB16_S2_S4, and RAMB16_S4_S4

```
-- Component Declaration for RAMB16_S1_{S1 | S2 | S4}, RAMB16_S2_{S2 | S4}, and
-- RAMB16_S4_S4 should be placed after architecture statement but before begin keyword

-- For the following component declaration, enter RAMB16_S1_{S1 | S2 | S4},
-- RAMB16_S2_{S2 | S4}, or RAMB16_S4_S4

component RAMB16_Sm_Sn
  -- synthesis translate_off
  generic (
      INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_08 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_09 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_0C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_10 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_11 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_12 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_13 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_14 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_15 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_16 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_17 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_18 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_19 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_20 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_21 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_22 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_23 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_24 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_25 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_26 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_27 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_28 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_29 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_30 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_31 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_32 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_33 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_34 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_35 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_36 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_37 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_38 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_39 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_A : bit_vector   := X"0";
       INIT_B : bit_vector   := X"0";
       INITP_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INITP_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       SRVAL_A : bit_vector   := X"0";
       SRVAL_B : bit_vector   := X"0";
       WRITE_MODE_A : string := "WRITE_FIRST";
       WRITE_MODE_B : string := "WRITE_FIRST";

  );
  -- synthesis translate_on

  port (DOA   : out STD_LOGIC_VECTOR (n downto 0);
        DOB   : out STD_LOGIC_VECTOR (n downto 0);
        ADDRA : in STD_LOGIC_VECTOR (n downto 0);
        ADDRB : in STD_LOGIC_VECTOR (n downto 0);
        CLKA  : in STD_ULOGIC;
        CLKB  : in STD_ULOGIC;
        DIA   : in STD_LOGIC_VECTOR (n downto 0);
        DIB   : in STD_LOGIC_VECTOR (n downto 0);
        ENA   : in STD_ULOGIC;
        ENB   : in STD_ULOGIC;
        SSRA  : in STD_ULOGIC;
        SSRB  : in STD_ULOGIC;
        WEA   : in STD_ULOGIC;
        WEB   : in STD_ULOGIC);

end component;

-- Component Attribute Specification for design element
-- should be placed after architecture declaration
-- but before the begin keyword

-- Put attributes, if necessary


-- Component Instantiation for design element
-- Should be placed in architecture after the begin keyword

RAMB16_Sm_Sn INSTANCE_NAME : RAMB16_Sm_Sn
  -- synthesis translate_off
  generic map (
       INIT_00 => vector_value,
       INIT_01 => vector_value,
```

```
INIT_02 => vector_value,
INIT_03 => vector_value,
INIT_04 => vector_value,
INIT_05 => vector_value,
INIT_06 => vector_value,
INIT_07 => vector_value,
INIT_08 => vector_value,
INIT_09 => vector_value,
INIT_0A => vector_value,
INIT_0B => vector_value,
INIT_0C => vector_value,
INIT_0D => vector_value,
INIT_0E => vector_value,
INIT_0F => vector_value,
INIT_10 => vector_value,
INIT_11 => vector_value,
INIT_12 => vector_value,
INIT_13 => vector_value,
INIT_14 => vector_value,
INIT_15 => vector_value,
INIT_16 => vector_value,
INIT_17 => vector_value,
INIT_18 => vector_value,
INIT_19 => vector_value,
INIT_1A => vector_value,
INIT_1B => vector_value,
INIT_1C => vector_value,
INIT_1D => vector_value,
INIT_1E => vector_value,
INIT_1F => vector_value,
INIT_20 => vector_value,
INIT_21 => vector_value,
INIT_22 => vector_value,
INIT_23 => vector_value,
INIT_24 => vector_value,
INIT_25 => vector_value,
INIT_26 => vector_value,
INIT_27 => vector_value,
INIT_28 => vector_value,
INIT_29 => vector_value,
INIT_2A => vector_value,
INIT_2B => vector_value,
INIT_2C => vector_value,
INIT_2D => vector_value,
INIT_2E => vector_value,
INIT_2F => vector_value,
INIT_30 => vector_value,
INIT_31 => vector_value,
INIT_32 => vector_value,
INIT_33 => vector_value,
INIT_34 => vector_value,
INIT_35 => vector_value,
INIT_36 => vector_value,
INIT_37 => vector_value,
```

```
      INIT_38 => vector_value,
      INIT_39 => vector_value,
      INIT_3A => vector_value,
      INIT_3B => vector_value,
      INIT_3C => vector_value,
      INIT_3D => vector_value,
      INIT_3E => vector_value,
      INIT_3F => vector_value,
      INIT_A => bit_value,
      INIT_B => bit_value,
      INITP_00 => vector_value,
      INITP_01 => vector_value,
      INITP_02 => vector_value,
      INITP_03 => vector_value,
      INITP_04 => vector_value,
      INITP_05 => vector_value,
      INITP_06 => vector_value,
      INITP_07 => vector_value,
      SRVAL_A => bit_value,
      SRVAL_B => bit_value,
      WRITE_MODE_A => string_value,
      WRITE_MODE_B => string_value)
  -- synopsys translate_on
   port map (DOA => user_DOA,
             DOB => user_DOB,
             ADDRA => user_ADDRA,
             ADDRB => user_ADDRB,
             CLKA => user_CLKA,
             CLKB => user_CLKB,
             DIA => user_DIA,
             DIB => user_DIB,
             ENA => user_ENA,
             ENB => user_ENB,
             SSRA => user_SSRA,
             SSRB => user_SSRB,
             WEA => user_WEA,
             WEB => user_WEB);
```

Verilog Instantiation Template for RAMB16_S1_S1, RAMB16_S1_S2, RAMB16_S1_S4, RAMB16_S2_S2, RAMB16_S2_S4, and RAMB16_S4_S4

```
RAMB16_Sm_Sn user_instance_name (.DOA (user_DOA),
                                 .DOB (user_DOB),
                                 .ADDRA (user_ADDRA),
                                 .ADDRB (user_ADDRB),
                                 .CLKA (user_CLKA),
                                 .CLKB (user_CLKB),
                                 .DIA (user_DIA),
                                 .DIB (user_DIB),
                                 .ENA (user_ENA),
                                 .ENB (user_ENB),
                                 .SSRA (user_SSRA),
                                 .SSRB (user_SSRB),
```

```
                                .WEA (user_WEA),
                                .WEB (user_WEB));

    defparam user_instance_name.INIT_00 = 256_bit_hex_value;
    defparam user_instance_name.INIT_01 = 256_bit_hex_value;
    defparam user_instance_name.INIT_02 = 256_bit_hex_value;
    defparam user_instance_name.INIT_03 = 256_bit_hex_value;
    defparam user_instance_name.INIT_04 = 256_bit_hex_value;
    defparam user_instance_name.INIT_05 = 256_bit_hex_value;
    defparam user_instance_name.INIT_06 = 256_bit_hex_value;
    defparam user_instance_name.INIT_07 = 256_bit_hex_value;
    defparam user_instance_name.INIT_08 = 256_bit_hex_value;
    defparam user_instance_name.INIT_09 = 256_bit_hex_value;
    defparam user_instance_name.INIT_0A = 256_bit_hex_value;
    defparam user_instance_name.INIT_0B = 256_bit_hex_value;
    defparam user_instance_name.INIT_0C = 256_bit_hex_value;
    defparam user_instance_name.INIT_0D = 256_bit_hex_value;
    defparam user_instance_name.INIT_0E = 256_bit_hex_value;
    defparam user_instance_name.INIT_0F = 256_bit_hex_value;
    defparam user_instance_name.INIT_10 = 256_bit_hex_value;
    defparam user_instance_name.INIT_11 = 256_bit_hex_value;
    defparam user_instance_name.INIT_12 = 256_bit_hex_value;
    defparam user_instance_name.INIT_13 = 256_bit_hex_value;
    defparam user_instance_name.INIT_14 = 256_bit_hex_value;
    defparam user_instance_name.INIT_15 = 256_bit_hex_value;
    defparam user_instance_name.INIT_16 = 256_bit_hex_value;
    defparam user_instance_name.INIT_17 = 256_bit_hex_value;
    defparam user_instance_name.INIT_18 = 256_bit_hex_value;
    defparam user_instance_name.INIT_19 = 256_bit_hex_value;
    defparam user_instance_name.INIT_1A = 256_bit_hex_value;
    defparam user_instance_name.INIT_1B = 256_bit_hex_value;
    defparam user_instance_name.INIT_1C = 256_bit_hex_value;
    defparam user_instance_name.INIT_1D = 256_bit_hex_value;
    defparam user_instance_name.INIT_1E = 256_bit_hex_value;
    defparam user_instance_name.INIT_1F = 256_bit_hex_value;
    defparam user_instance_name.INIT_20 = 256_bit_hex_value;
    defparam user_instance_name.INIT_21 = 256_bit_hex_value;
    defparam user_instance_name.INIT_22 = 256_bit_hex_value;
    defparam user_instance_name.INIT_23 = 256_bit_hex_value;
    defparam user_instance_name.INIT_24 = 256_bit_hex_value;
    defparam user_instance_name.INIT_25 = 256_bit_hex_value;
    defparam user_instance_name.INIT_26 = 256_bit_hex_value;
    defparam user_instance_name.INIT_27 = 256_bit_hex_value;
    defparam user_instance_name.INIT_28 = 256_bit_hex_value;
    defparam user_instance_name.INIT_29 = 256_bit_hex_value;
    defparam user_instance_name.INIT_2A = 256_bit_hex_value;
    defparam user_instance_name.INIT_2B = 256_bit_hex_value;
    defparam user_instance_name.INIT_2C = 256_bit_hex_value;
    defparam user_instance_name.INIT_2D = 256_bit_hex_value;
    defparam user_instance_name.INIT_2E = 256_bit_hex_value;
    defparam user_instance_name.INIT_2F = 256_bit_hex_value;
    defparam user_instance_name.INIT_30 = 256_bit_hex_value;
    defparam user_instance_name.INIT_31 = 256_bit_hex_value;
    defparam user_instance_name.INIT_32 = 256_bit_hex_value;
```

```
    defparam user_instance_name.INIT_33 = 256_bit_hex_value;
    defparam user_instance_name.INIT_34 = 256_bit_hex_value;
    defparam user_instance_name.INIT_35 = 256_bit_hex_value;
    defparam user_instance_name.INIT_36 = 256_bit_hex_value;
    defparam user_instance_name.INIT_37 = 256_bit_hex_value;
    defparam user_instance_name.INIT_38 = 256_bit_hex_value;
    defparam user_instance_name.INIT_39 = 256_bit_hex_value;
    defparam user_instance_name.INIT_3A = 256_bit_hex_value;
    defparam user_instance_name.INIT_3B = 256_bit_hex_value;
    defparam user_instance_name.INIT_3C = 256_bit_hex_value;
    defparam user_instance_name.INIT_3D = 256_bit_hex_value;
    defparam user_instance_name.INIT_3E = 256_bit_hex_value;
    defparam user_instance_name.INIT_3F = 256_bit_hex_value;
    defparam user_instance_name.INIT_A = bit_value;
    defparam user_instance_name.INIT_B = bit_value;
    defparam user_instance_name.INITP_00 = 256_bit_hex_value;
    defparam user_instance_name.INITP_01 = 256_bit_hex_value;
    defparam user_instance_name.INITP_02 = 256_bit_hex_value;
    defparam user_instance_name.INITP_03 = 256_bit_hex_value;
    defparam user_instance_name.INITP_04 = 256_bit_hex_value;
    defparam user_instance_name.INITP_05 = 256_bit_hex_value;
    defparam user_instance_name.INITP_06 = 256_bit_hex_value;
    defparam user_instance_name.INITP_07 = 256_bit_hex_value;
    defparam user_instance_name.SRVAL_A = bit_value;
    defparam user_instance_name.SRVAL_B = bit_value;
    defparam user_instance_name.WRITE_MODE_A = string_value;
    defparam user_instance_name.WRITE_MODE_B = string_value;
```

VHDL Instantiation Template for RAMB16_S1_S9, RAMB16_S1_S18, RAMB16_S1_S36, RAMB16_S2_S9, RAMB16_S2_S18, RAMB16_S2_S36, RAMB16_S4_S9, RAMB16_S4_S18, and RAMB16_S4_S36

```
-- Component Declaration for these design elements
-- should be placed after architecture statement but before begin keyword

-- For the following component declaration, enter RAMB16_S1_{S9 | S18 | S36},
-- RAMB16_S2_{S9 | S18 | S36}, or RAMB16_S4_{S9 | S18 | S36}

component RAMB16_Sm_Sn
  -- synthesis translate_off
  generic (
      INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_08 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_09 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_10 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_11 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_12 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_13 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_14 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_15 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_16 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_17 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_18 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_19 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_20 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_21 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_22 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_23 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_24 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_25 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_26 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_27 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_28 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_29 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_30 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_31 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_32 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_33 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_34 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_35 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_36 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_37 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_38 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_39 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
```

```
       INIT_3C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_A : bit_vector   := X"0";
       INIT_B : bit_vector   := X"0";
       INITP_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       SRVAL_A : bit_vector   := X"0";
       SRVAL_B : bit_vector   := X"0";
       WRITE_MODE_A : string := "WRITE_FIRST";
       WRITE_MODE_B : string := "WRITE_FIRST";
  );
  -- synthesis translate_on

  port (DOA   : out STD_LOGIC_VECTOR (n downto 0);
        DOB   : out STD_LOGIC_VECTOR (n downto 0);
        DOPB  : out STD_LOGIC_VECTOR (n downto 0);
        ADDRA : in STD_LOGIC_VECTOR (n downto 0);
        ADDRB : in STD_LOGIC_VECTOR (n downto 0);
        CLKA  : in STD_ULOGIC;
        CLKB  : in STD_ULOGIC;
        DIA   : in STD_LOGIC_VECTOR (n downto 0);
        DIB   : in STD_LOGIC_VECTOR (n downto 0);
        DIPB  : in STD_LOGIC_VECTOR (n downto 0);
        ENA   : in STD_ULOGIC;
        ENB   : in STD_ULOGIC;
        SSRA  : in STD_ULOGIC;
        SSRB  : in STD_ULOGIC;
        WEA   : in STD_ULOGIC;
        WEB   : in STD_ULOGIC);

end component;

-- Component Attribute Specification for design element
-- should be placed after architecture declaration
```

```
-- but before the begin keyword

-- Put attributes, if necessary


-- Component Instantiation for design element
-- Should be placed in architecture after the begin keyword

RAMB16_Sm_Sn INSTANCE_NAME : RAMB16_Sm_Sn
  -- synthesis translate_off
  generic map (
       INIT_00 => vector_value,
       INIT_01 => vector_value,
       INIT_02 => vector_value,
       INIT_03 => vector_value,
       INIT_04 => vector_value,
       INIT_05 => vector_value,
       INIT_06 => vector_value,
       INIT_07 => vector_value,
       INIT_08 => vector_value,
       INIT_09 => vector_value,
       INIT_0A => vector_value,
       INIT_0B => vector_value,
       INIT_0C => vector_value,
       INIT_0D => vector_value,
       INIT_0E => vector_value,
       INIT_0F => vector_value,
       INIT_10 => vector_value,
       INIT_11 => vector_value,
       INIT_12 => vector_value,
       INIT_13 => vector_value,
       INIT_14 => vector_value,
       INIT_15 => vector_value,
       INIT_16 => vector_value,
       INIT_17 => vector_value,
       INIT_18 => vector_value,
       INIT_19 => vector_value,
       INIT_1A => vector_value,
       INIT_1B => vector_value,
       INIT_1C => vector_value,
       INIT_1D => vector_value,
       INIT_1E => vector_value,
       INIT_1F => vector_value,
       INIT_20 => vector_value,
       INIT_21 => vector_value,
       INIT_22 => vector_value,
       INIT_23 => vector_value,
       INIT_24 => vector_value,
       INIT_25 => vector_value,
       INIT_26 => vector_value,
       INIT_27 => vector_value,
       INIT_28 => vector_value,
       INIT_29 => vector_value,
       INIT_2A => vector_value,
```

```
      INIT_2B => vector_value,
      INIT_2C => vector_value,
      INIT_2D => vector_value,
      INIT_2E => vector_value,
      INIT_2F => vector_value,
      INIT_30 => vector_value,
      INIT_31 => vector_value,
      INIT_32 => vector_value,
      INIT_33 => vector_value,
      INIT_34 => vector_value,
      INIT_35 => vector_value,
      INIT_36 => vector_value,
      INIT_37 => vector_value,
      INIT_38 => vector_value,
      INIT_39 => vector_value,
      INIT_3A => vector_value,
      INIT_3B => vector_value,
      INIT_3C => vector_value,
      INIT_3D => vector_value,
      INIT_3E => vector_value,
      INIT_3F => vector_value,
      INIT_A => bit_value,
      INIT_B => bit_value,
      INITP_00 => vector_value,
      INITP_01 => vector_value,
      INITP_02 => vector_value,
      INITP_03 => vector_value,
      INITP_04 => vector_value,
      INITP_05 => vector_value,
      INITP_06 => vector_value,
      INITP_07 => vector_value,
      SRVAL_A => bit_value,
      SRVAL_B => bit_value,
      WRITE_MODE_A => string_value,
      WRITE_MODE_B => string_value)
  -- synopsys translate_on
   port map (DOA => user_DOA,
             DOB => user_DOB,
             DOPB => user_DOPB,
             ADDRA => user_ADDRA,
             ADDRB => user_ADDRB,
             CLKA => user_CLKA,
             CLKB => user_CLKB,
             DIA => user_DIA,
             DIB => user_DIB,
             DIPB => user_DIPB,
             ENA => user_ENA,
             ENB => user_ENB,
             SSRA => user_SSRA,
             SSRB => user_SSRB,
             WEA => user_WEA,
             WEB => user_WEB);
```

Verilog Instantiation Template for RAMB16_S1_S9, RAMB16_S1_S18, RAMB16_S1_S36, RAMB16_S2_S9, RAMB16_S2_S18, RAMB16_S2_S36, RAMB16_S4_S9, RAMB16_S4_S18, and RAMB16_S4_S36

```
RAMB16_Sm_Sn user_instance_name (.DOA (user_DOA),
                                 .DOB (user_DOB),
                                 .DOPB (user_DOPB),
                                 .ADDRA (user_ADDRA),
                                 .ADDRB (user_ADDRB),
                                 .CLKA (user_CLKA),
                                 .CLKB (user_CLKB),
                                 .DIA (user_DIA),
                                 .DIB (user_DIB),
                                 .DIPB (user_DIB),
                                 .ENA (user_ENA),
                                 .ENB (user_ENB),
                                 .SSRA (user_SSRA),
                                 .SSRB (user_SSRB),
                                 .WEA (user_WEA),
                                 .WEB (user_WEB));

    defparam user_instance_name.INIT_00 = 256_bit_hex_value;
    defparam user_instance_name.INIT_01 = 256_bit_hex_value;
    defparam user_instance_name.INIT_02 = 256_bit_hex_value;
    defparam user_instance_name.INIT_03 = 256_bit_hex_value;
    defparam user_instance_name.INIT_04 = 256_bit_hex_value;
    defparam user_instance_name.INIT_05 = 256_bit_hex_value;
    defparam user_instance_name.INIT_06 = 256_bit_hex_value;
    defparam user_instance_name.INIT_07 = 256_bit_hex_value;
    defparam user_instance_name.INIT_08 = 256_bit_hex_value;
    defparam user_instance_name.INIT_09 = 256_bit_hex_value;
    defparam user_instance_name.INIT_0A = 256_bit_hex_value;
    defparam user_instance_name.INIT_0B = 256_bit_hex_value;
    defparam user_instance_name.INIT_0C = 256_bit_hex_value;
    defparam user_instance_name.INIT_0D = 256_bit_hex_value;
    defparam user_instance_name.INIT_0E = 256_bit_hex_value;
    defparam user_instance_name.INIT_0F = 256_bit_hex_value;
    defparam user_instance_name.INIT_10 = 256_bit_hex_value;
    defparam user_instance_name.INIT_11 = 256_bit_hex_value;
    defparam user_instance_name.INIT_12 = 256_bit_hex_value;
    defparam user_instance_name.INIT_13 = 256_bit_hex_value;
    defparam user_instance_name.INIT_14 = 256_bit_hex_value;
    defparam user_instance_name.INIT_15 = 256_bit_hex_value;
    defparam user_instance_name.INIT_16 = 256_bit_hex_value;
    defparam user_instance_name.INIT_17 = 256_bit_hex_value;
    defparam user_instance_name.INIT_18 = 256_bit_hex_value;
    defparam user_instance_name.INIT_19 = 256_bit_hex_value;
    defparam user_instance_name.INIT_1A = 256_bit_hex_value;
    defparam user_instance_name.INIT_1B = 256_bit_hex_value;
    defparam user_instance_name.INIT_1C = 256_bit_hex_value;
    defparam user_instance_name.INIT_1D = 256_bit_hex_value;
    defparam user_instance_name.INIT_1E = 256_bit_hex_value;
    defparam user_instance_name.INIT_1F = 256_bit_hex_value;
```

```
defparam user_instance_name.INIT_20 = 256_bit_hex_value;
defparam user_instance_name.INIT_21 = 256_bit_hex_value;
defparam user_instance_name.INIT_22 = 256_bit_hex_value;
defparam user_instance_name.INIT_23 = 256_bit_hex_value;
defparam user_instance_name.INIT_24 = 256_bit_hex_value;
defparam user_instance_name.INIT_25 = 256_bit_hex_value;
defparam user_instance_name.INIT_26 = 256_bit_hex_value;
defparam user_instance_name.INIT_27 = 256_bit_hex_value;
defparam user_instance_name.INIT_28 = 256_bit_hex_value;
defparam user_instance_name.INIT_29 = 256_bit_hex_value;
defparam user_instance_name.INIT_2A = 256_bit_hex_value;
defparam user_instance_name.INIT_2B = 256_bit_hex_value;
defparam user_instance_name.INIT_2C = 256_bit_hex_value;
defparam user_instance_name.INIT_2D = 256_bit_hex_value;
defparam user_instance_name.INIT_2E = 256_bit_hex_value;
defparam user_instance_name.INIT_2F = 256_bit_hex_value;
defparam user_instance_name.INIT_30 = 256_bit_hex_value;
defparam user_instance_name.INIT_31 = 256_bit_hex_value;
defparam user_instance_name.INIT_32 = 256_bit_hex_value;
defparam user_instance_name.INIT_33 = 256_bit_hex_value;
defparam user_instance_name.INIT_34 = 256_bit_hex_value;
defparam user_instance_name.INIT_35 = 256_bit_hex_value;
defparam user_instance_name.INIT_36 = 256_bit_hex_value;
defparam user_instance_name.INIT_37 = 256_bit_hex_value;
defparam user_instance_name.INIT_38 = 256_bit_hex_value;
defparam user_instance_name.INIT_39 = 256_bit_hex_value;
defparam user_instance_name.INIT_3A = 256_bit_hex_value;
defparam user_instance_name.INIT_3B = 256_bit_hex_value;
defparam user_instance_name.INIT_3C = 256_bit_hex_value;
defparam user_instance_name.INIT_3D = 256_bit_hex_value;
defparam user_instance_name.INIT_3E = 256_bit_hex_value;
defparam user_instance_name.INIT_3F = 256_bit_hex_value;
defparam user_instance_name.INIT_A = bit_value;
defparam user_instance_name.INIT_B = bit_value;
defparam user_instance_name.INITP_00 = 256_bit_hex_value;
defparam user_instance_name.INITP_01 = 256_bit_hex_value;
defparam user_instance_name.INITP_02 = 256_bit_hex_value;
defparam user_instance_name.INITP_03 = 256_bit_hex_value;
defparam user_instance_name.INITP_04 = 256_bit_hex_value;
defparam user_instance_name.INITP_05 = 256_bit_hex_value;
defparam user_instance_name.INITP_06 = 256_bit_hex_value;
defparam user_instance_name.INITP_07 = 256_bit_hex_value;
defparam user_instance_name.SRVAL_A = bit_value;
defparam user_instance_name.SRVAL_B = bit_value;
defparam user_instance_name.WRITE_MODE_A = string_value;
defparam user_instance_name.WRITE_MODE_B = string_value;
```

VHDL Instantiation Template RAMB16_S9_S9, RAMB16_S9_S18, RAMB16_S9_S36, RAMB16_S18_S18, RAMB16_S18_S36, and RAMB16_S36_S36

```
-- Component Declaration for these design elements
-- should be placed after architecture statement but before begin keyword

-- For the following component declaration, enter RAMB16_S9_{S9 | S18 | S36},
-- RAMB16_S18_{S18 | S36}, or RAMB16_S36_S36

component RAMB16_Sm_Sn
  -- synthesis translate_off
  generic (
      INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_08 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_09 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_0F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_10 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_11 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_12 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_13 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_14 : bit_vector :=
```

```
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_15 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_16 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_17 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_18 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_19 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_1F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_20 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_21 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_22 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_23 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_24 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_25 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_26 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_27 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_28 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_29 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_2F : bit_vector :=
```

```
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_30 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_31 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_32 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_33 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_34 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_35 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_36 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_37 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_38 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_39 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3A : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3B : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3C : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3D : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3E : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3F : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_A : bit_vector   := X"0";
       INIT_B : bit_vector   := X"0";
       INITP_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_04 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_05 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_06 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_07 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       SRVAL_A : bit_vector   := X"0";
       SRVAL_B : bit_vector   := X"0";
       WRITE_MODE_A : string := "WRITE_FIRST";
```

```
       WRITE_MODE_B : string := "WRITE_FIRST";
);
  -- synthesis translate_on

  port (DOA   : out STD_LOGIC_VECTOR (n downto 0);
        DOB   : out STD_LOGIC_VECTOR (n downto 0);
        DOPA  : out STD_LOGIC_VECTOR (n downto 0);
        DOPB  : out STD_LOGIC_VECTOR (n downto 0);
        ADDRA : in STD_LOGIC_VECTOR (n downto 0);
        ADDRB : in STD_LOGIC_VECTOR (n downto 0);
        CLKA  : in STD_ULOGIC;
        CLKB  : in STD_ULOGIC;
        DIA   : in STD_LOGIC_VECTOR (n downto 0);
        DIB   : in STD_LOGIC_VECTOR (n downto 0);
        DIPA  : in STD_LOGIC_VECTOR (n downto 0);
        DIPB  : in STD_LOGIC_VECTOR (n downto 0);
        ENA   : in STD_ULOGIC;
        ENB   : in STD_ULOGIC;
        SSRA  : in STD_ULOGIC;
        SSRB  : in STD_ULOGIC;
        WEA   : in STD_ULOGIC;
        WEB   : in STD_ULOGIC);

end component;

-- Component Attribute Specification for design element
-- should be placed after architecture declaration
-- but before the begin keyword

-- Put attributes, if necessary


-- Component Instantiation for design element
-- Should be placed in architecture after the begin keyword

RAMB16_Sm_Sn INSTANCE_NAME : RAMB16_Sm_Sn
  -- synthesis translate_off
  generic map (
INIT_00 => vector_value,
       INIT_01 => vector_value,
       INIT_02 => vector_value,
       INIT_03 => vector_value,
       INIT_04 => vector_value,
       INIT_05 => vector_value,
       INIT_06 => vector_value,
       INIT_07 => vector_value,
       INIT_08 => vector_value,
       INIT_09 => vector_value,
       INIT_0A => vector_value,
       INIT_0B => vector_value,
       INIT_0C => vector_value,
       INIT_0D => vector_value,
       INIT_0E => vector_value,
       INIT_0F => vector_value,
```

```
INIT_10 => vector_value,
INIT_11 => vector_value,
INIT_12 => vector_value,
INIT_13 => vector_value,
INIT_14 => vector_value,
INIT_15 => vector_value,
INIT_16 => vector_value,
INIT_17 => vector_value,
INIT_18 => vector_value,
INIT_19 => vector_value,
INIT_1A => vector_value,
INIT_1B => vector_value,
INIT_1C => vector_value,
INIT_1D => vector_value,
INIT_1E => vector_value,
INIT_1F => vector_value,
INIT_20 => vector_value,
INIT_21 => vector_value,
INIT_22 => vector_value,
INIT_23 => vector_value,
INIT_24 => vector_value,
INIT_25 => vector_value,
INIT_26 => vector_value,
INIT_27 => vector_value,
INIT_28 => vector_value,
INIT_29 => vector_value,
INIT_2A => vector_value,
INIT_2B => vector_value,
INIT_2C => vector_value,
INIT_2D => vector_value,
INIT_2E => vector_value,
INIT_2F => vector_value,
INIT_30 => vector_value,
INIT_31 => vector_value,
INIT_32 => vector_value,
INIT_33 => vector_value,
INIT_34 => vector_value,
INIT_35 => vector_value,
INIT_36 => vector_value,
INIT_37 => vector_value,
INIT_38 => vector_value,
INIT_39 => vector_value,
INIT_3A => vector_value,
INIT_3B => vector_value,
INIT_3C => vector_value,
INIT_3D => vector_value,
INIT_3E => vector_value,
INIT_3F => vector_value,
INIT_A => bit_value,
INIT_B => bit_value,
INITP_00 => vector_value,
INITP_01 => vector_value,
INITP_02 => vector_value,
INITP_03 => vector_value,
```

```
      INITP_04 => vector_value,
      INITP_05 => vector_value,
      INITP_06 => vector_value,
      INITP_07 => vector_value,
      SRVAL_A => bit_value,
      SRVAL_B => bit_value,
      WRITE_MODE_A => string_value,
      WRITE_MODE_B => string_value)
  -- synopsys translate_on
   port map (DOA => user_DOA,
             DOB => user_DOB,
             DOPA => user_DOPA,
             DOPB => user_DOPB,
             ADDRA => user_ADDRA,
             ADDRB => user_ADDRB,
             CLKA => user_CLKA,
             CLKB => user_CLKB,
             DIA => user_DIA,
             DIB => user_DIB,
             DIPA => user_DIPA,
             DIPB => user_DIPB,
             ENA => user_ENA,
             ENB => user_ENB,
             SSRA => user_SSRA,
             SSRB => user_SSRB,
             WEA => user_WEA,
             WEB => user_WEB);
```

Verilog Instantiation Template for RAMB16_S9_S9, RAMB16_S9_S18, RAMB16_S9_S36, RAMB16_S18_S18, RAMB16_S18_S36, and RAMB16_S36_S36

```
RAMB16_Sm_Sn user_instance_name (.DOA (user_DOA),
                                 .DOB (user_DOB),
                                 .DOPA (user_DOPA),
                                 .DOPB (user_DOPB),
                                 .ADDRA (user_ADDRA),
                                 .ADDRB (user_ADDRB),
                                 .CLKA (user_CLKA),
                                 .CLKB (user_CLKB),
                                 .DIA (user_DIA),
                                 .DIB (user_DIB),
                                 .DIPA (user_DIPA),
                                 .DIPB (user_DIB),
                                 .ENA (user_ENA),
                                 .ENB (user_ENB),
                                 .SSRA (user_SSRA),
                                 .SSRB (user_SSRB),
                                 .WEA (user_WEA),
                                 .WEB (user_WEB));

    defparam user_instance_name.INIT_00 = 256_bit_hex_value;
    defparam user_instance_name.INIT_01 = 256_bit_hex_value;
    defparam user_instance_name.INIT_02 = 256_bit_hex_value;
```

```
defparam user_instance_name.INIT_03 = 256_bit_hex_value;
defparam user_instance_name.INIT_04 = 256_bit_hex_value;
defparam user_instance_name.INIT_05 = 256_bit_hex_value;
defparam user_instance_name.INIT_06 = 256_bit_hex_value;
defparam user_instance_name.INIT_07 = 256_bit_hex_value;
defparam user_instance_name.INIT_08 = 256_bit_hex_value;
defparam user_instance_name.INIT_09 = 256_bit_hex_value;
defparam user_instance_name.INIT_0A = 256_bit_hex_value;
defparam user_instance_name.INIT_0B = 256_bit_hex_value;
defparam user_instance_name.INIT_0C = 256_bit_hex_value;
defparam user_instance_name.INIT_0D = 256_bit_hex_value;
defparam user_instance_name.INIT_0E = 256_bit_hex_value;
defparam user_instance_name.INIT_0F = 256_bit_hex_value;
defparam user_instance_name.INIT_10 = 256_bit_hex_value;
defparam user_instance_name.INIT_11 = 256_bit_hex_value;
defparam user_instance_name.INIT_12 = 256_bit_hex_value;
defparam user_instance_name.INIT_13 = 256_bit_hex_value;
defparam user_instance_name.INIT_14 = 256_bit_hex_value;
defparam user_instance_name.INIT_15 = 256_bit_hex_value;
defparam user_instance_name.INIT_16 = 256_bit_hex_value;
defparam user_instance_name.INIT_17 = 256_bit_hex_value;
defparam user_instance_name.INIT_18 = 256_bit_hex_value;
defparam user_instance_name.INIT_19 = 256_bit_hex_value;
defparam user_instance_name.INIT_1A = 256_bit_hex_value;
defparam user_instance_name.INIT_1B = 256_bit_hex_value;
defparam user_instance_name.INIT_1C = 256_bit_hex_value;
defparam user_instance_name.INIT_1D = 256_bit_hex_value;
defparam user_instance_name.INIT_1E = 256_bit_hex_value;
defparam user_instance_name.INIT_1F = 256_bit_hex_value;
defparam user_instance_name.INIT_20 = 256_bit_hex_value;
defparam user_instance_name.INIT_21 = 256_bit_hex_value;
defparam user_instance_name.INIT_22 = 256_bit_hex_value;
defparam user_instance_name.INIT_23 = 256_bit_hex_value;
defparam user_instance_name.INIT_24 = 256_bit_hex_value;
defparam user_instance_name.INIT_25 = 256_bit_hex_value;
defparam user_instance_name.INIT_26 = 256_bit_hex_value;
defparam user_instance_name.INIT_27 = 256_bit_hex_value;
defparam user_instance_name.INIT_28 = 256_bit_hex_value;
defparam user_instance_name.INIT_29 = 256_bit_hex_value;
defparam user_instance_name.INIT_2A = 256_bit_hex_value;
defparam user_instance_name.INIT_2B = 256_bit_hex_value;
defparam user_instance_name.INIT_2C = 256_bit_hex_value;
defparam user_instance_name.INIT_2D = 256_bit_hex_value;
defparam user_instance_name.INIT_2E = 256_bit_hex_value;
defparam user_instance_name.INIT_2F = 256_bit_hex_value;
defparam user_instance_name.INIT_30 = 256_bit_hex_value;
defparam user_instance_name.INIT_31 = 256_bit_hex_value;
defparam user_instance_name.INIT_32 = 256_bit_hex_value;
defparam user_instance_name.INIT_33 = 256_bit_hex_value;
defparam user_instance_name.INIT_34 = 256_bit_hex_value;
defparam user_instance_name.INIT_35 = 256_bit_hex_value;
defparam user_instance_name.INIT_36 = 256_bit_hex_value;
defparam user_instance_name.INIT_37 = 256_bit_hex_value;
defparam user_instance_name.INIT_38 = 256_bit_hex_value;
```

```
defparam user_instance_name.INIT_39 = 256_bit_hex_value;
defparam user_instance_name.INIT_3A = 256_bit_hex_value;
defparam user_instance_name.INIT_3B = 256_bit_hex_value;
defparam user_instance_name.INIT_3C = 256_bit_hex_value;
defparam user_instance_name.INIT_3D = 256_bit_hex_value;
defparam user_instance_name.INIT_3E = 256_bit_hex_value;
defparam user_instance_name.INIT_3F = 256_bit_hex_value;
defparam user_instance_name.INIT_A = bit_value;
defparam user_instance_name.INIT_B = bit_value;
defparam user_instance_name.INITP_00 = 256_bit_hex_value;
defparam user_instance_name.INITP_01 = 256_bit_hex_value;
defparam user_instance_name.INITP_02 = 256_bit_hex_value;
defparam user_instance_name.INITP_03 = 256_bit_hex_value;
defparam user_instance_name.INITP_04 = 256_bit_hex_value;
defparam user_instance_name.INITP_05 = 256_bit_hex_value;
defparam user_instance_name.INITP_06 = 256_bit_hex_value;
defparam user_instance_name.INITP_07 = 256_bit_hex_value;
defparam user_instance_name.SRVAL_A = bit_value;
defparam user_instance_name.SRVAL_B = bit_value;
defparam user_instance_name.WRITE_MODE_A = string_value;
defparam user_instance_name.WRITE_MODE_B = string_value;
```

Commonly Used Constraints

INIT, INIT_xx, INIT_A, INIT_B, INTP_xx, SRVAL_A, SRVAL_B, WRITE_MODE_A, WRITE_MODE_B

ROC

Reset On Configuration

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A

The ROC is a component used for VHDL simulation of FPGA designs. This component should not be used for Verilog or schematic entry. The ROC's function is to mimic the function of the internal reset signal during the FPGA configuration process. In order to use ROC, it must be connected to the reset/preset signal for all inferred and instantiated registers in the design. During synthesis and implementation, this reset signal will use the dedicated global set/reset network and will not use local routing resources. During simulation, ROC will emit a one-shot pulse for the amount of time specified by the WIDTH generic (default is 100 ns). This one-shot pulse is intended to reset all registers so that at the beginning of operation, all registers are at a known value as would happen in the real silicon during configuration of the device.

For more information, please consult the *Xilinx Synthesis and Simulation Design Guide.*

Port O will be high at simulation time 0 for the amount of time specified by the WIDTH generic attribute. After that time, it will be 0. This will not affect implementation in any way.

VHDL Instantiation Code

```
component ROC
-- synthesis translate_off
  generic (WIDTH : Time := 100 ns);
-- synthesis translate_on
  port (O : out std_ulogic := '1');
end component;
```

Commonly Used Constraints

For simulation, the WIDTH generic can be modified to change the amount of time the one-shot pulse is applied for.

There are no supported constraints for this component for implementation.

ROCBUF

Reset On Configuration Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A

The ROCBUF is a component used for VHDL simulation of FPGA designs that is similar to the ROC component except that it contains an input for controlling the global set/reset function rather than a one-shot. This component should not be used for Verilog or schematic entry. The ROCBUF's function allows user control of the function of the global set/reset signal as done during the FPGA configuration process. In order to use the ROCBUF, the input should be connected to a top-level port in the design and the output must be connected to the reset/preset signal for all inferred and instantiated registers in the design. During simulation, the input to the ROCBUF can be toggled by the testbench in order to activate the global set/reset signal in the design. This should be done at the beginning of the simulation as is done in the real silicon after configuration to get the design in a known state. The signal may also be pulsed during simulation to simulate a reconfiguration (PROG pin high) of the device. During synthesis and implementation, this reset signal will use the dedicated global set/reset network and will not use local routing resources. The port connected to this component will be optimized out of the design and not use any pin resources. If you want to have the port implemented in the design, a STARTBUF_*architecture* should be used. In order to replace this port during back-end simulation the -gp switch should be used when invoking the NGD2VER or NGD2VHDL netlister. If using the ISE GUI, use the "Bring Out Global Set/Reset Net as a Port" option in the Simulation Model Properties window.

For more information, please consult the *Xilinx Synthesis and Simulation Design Guide.*

The value at port O will always be the value at port I (it is a buffer).

VHDL Instantiation Code

```
component ROCBUF
  port( I : in  std_ulogic;
        O : out std_ulogic);
end component;
```

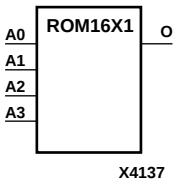
Commonly Used Constraints

None

ROM16X1

16-Deep by 1-Wide ROM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



ROM16X1 is a 16-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 4-bit address (A3 – A0). The ROM is initialized to a known value during configuration with the INIT=value parameter. The value consists of four hexadecimal digits that are written into the ROM from the most-significant digit A=FH to the least-significant digit A=0H. For example, the INIT=10A7 parameter produces the data stream:

```
0001 0000 1010 0111
```

An error occurs if the INIT=*value* is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM16X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for ROM16X1 should be placed
-- after architecture statement but before begin keyword

component ROM16X1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"16");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A3 : in STD_ULOGIC);
end component;

-- Component Attribute specification for ROM16X1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ROM16X1 should be placed
-- in architecture after the begin keyword

ROM16X1_INSTANCE_NAME : ROM16X1
```

```
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3);
```

Verilog Instantiation Template

```
ROM16X1 instance_name (.O (user_O),
                       .A0 (user_A0),
                       .A1 (user_A1),
                       .A2 (user_A2),
                       .A3 (user_A3));

defparam user_instance_name.INIT = hex_value;
```

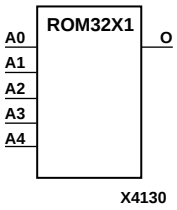
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKN

ROM32X1

32-Deep by 1-Wide ROM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N / A	N / A	N / A



ROM32X1 is a 32-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 5-bit address (A4 – A0). The ROM is initialized to a known value during configuration with the INIT=value parameter. The value consists of eight hexadecimal digits that are written into the ROM from the most-significant digit A=1FH to the least-significant digit A=00H. For example, the INIT=10A78F39 parameter produces the data stream:

```
0001 0000 1010 0111 1000 1111 0011 1001
```

An error occurs if the INIT=*value* is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM32X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for ROM32X1 should be placed
-- after architecture statement but before begin keyword

component ROM32X1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"32");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A3 : in STD_ULOGIC;
            A4 : in STD_ULOGIC);
end component;

-- Component Attribute specification for ROM32X1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ROM32X1 should be placed
-- in architecture after the begin keyword

ROM32X1_INSTANCE_NAME : ROM32X1
```

```
   -- synthesis translate_off
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4   => user_A4);
```

Verilog Instantiation Template

```
ROM32X1 instance_name (.O (user_O),
                       .A0 (user_A0),
                       .A1  (user_A1),
                       .A2  (user_A2),
                       .A3  (user_A3),
                       .A4 (user_A4));

defparam user_instance_name.INIT = hex_value;
```

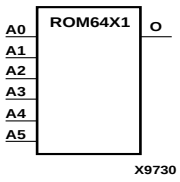
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKN

ROM64X1

64-Deep by 1-Wide ROM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



ROM64X1 is a 64-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 6-bit address (A5 – A0). The ROM is initialized to a known value during configuration with the INIT=value parameter. The value consists of four hexadecimal digits that are written into the ROM from the most-significant digit A=FH to the least-significant digit A=0H.

An error occurs if the INIT=*value* is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM64X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for ROM64X1 should be placed
-- after architecture statement but before begin keyword

component ROM64X1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"64");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A3 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            A5 : in STD_ULOGIC);
end component;

-- Component Attribute specification for ROM64X1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ROM64X1 should be placed
-- in architecture after the begin keyword

ROM64X1_INSTANCE_NAME : ROM64X1
   -- synthesis translate_off
   generic map(INIT => hex_value);
```

```
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 A5     => user_A5);
```

Verilog Instantiation Template

```
ROM64X1 instance_name (.O (user_O),
                       .A0 (user_A0),
                       .A1 (user_A1),
                       .A2 (user_A2),
                       .A3 (user_A3),
                       .A4 (user_A4),
                       .A5(user_A5));

defparam user_instance_name.INIT = hex_value;
```

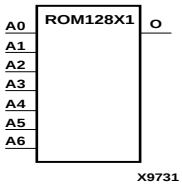
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKN

ROM128X1

128-Deep by 1-Wide ROM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



ROM128X1 is a 128-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 7-bit address (A6 – A0). The ROM is initialized to a known value during configuration with the INIT=value parameter. The value consists of four hexadecimal digits that are written into the ROM from the most-significant digit A=FH to the least-significant digit A=0H.

An error occurs if the INIT=*value* is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM128X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for ROM128X1 should be placed
-- after architecture statement but before begin keyword

component ROM128X1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"128");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A3 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            A5 : in STD_ULOGIC;
            A6 : in STD_ULOGIC);
end component;

-- Component Attribute specification for ROM128X1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ROM128X1 should be placed
-- in architecture after the begin keyword

ROM128X1_INSTANCE_NAME : ROM128X1
   -- synthesis translate_off
```

```
   generic map(INIT => hex_value);
   -- synthesis translate_on
      port map  (O      => user_O,
                 A0     => user_A0,
                 A1     => user_A1,
                 A2     => user_A2,
                 A3     => user_A3,
                 A4     => user_A4,
                 A5     => user_A5,
                 A6     => user_A6);
```

Verilog Instantiation Template

```
ROM128X1 instance_name (.O (user_O),
                        .A0 (user_A0),
                        .A1  (user_A1),
                        .A2  (user_A2),
                        .A3  (user_A3),
                        .A4  (user_A4),
                        .A5 (user_A5),
                        .A6 (user_A6));

defparam user_instance_name.INIT = hex_value;
```

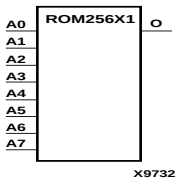
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKN

ROM256X1

256-Deep by 1-Wide ROM

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



ROM256X1 is a 256-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 8-bit address (A7– A0). The ROM is initialized to a known value during configuration with the INIT=value parameter. The value consists of four hexadecimal digits that are written into the ROM from the most-significant digit A=FH to the least-significant digit A=0H.

An error occurs if the INIT=*value* is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM256X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for ROM256X1 should be placed
-- after architecture statement but before begin keyword

component ROM256X1
   -- synthesis translate_off
   generic (INIT: bit_vector := X"256");
   -- synthesis translate_on
      port (O : out STD_ULOGIC;
            A0 : in STD_ULOGIC;
            A1 : in STD_ULOGIC;
            A2 : in STD_ULOGIC;
            A3 : in STD_ULOGIC;
            A4 : in STD_ULOGIC;
            A5 : in STD_ULOGIC;
            A6 : in STD_ULOGIC;
            A7 : in STD_ULOGIC);
end component;

-- Component Attribute specification for ROM256X1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for ROM256X1 should be placed
-- in architecture after the begin keyword

ROM256X1_INSTANCE_NAME : ROM256X1
```

```
-- synthesis translate_off
generic map(INIT => hex_value);
-- synthesis translate_on
   port map  (O      => user_O,
              A0     => user_A0,
              A1     => user_A1,
              A2     => user_A2,
              A3     => user_A3,
              A4     => user_A4,
              A5     => user_A5,
              A6     => user_A6,
              A7     => user_A7);
```

Verilog Instantiation Template

```
ROM256X1 instance_name (.O (user_O),
                        .A0 (user_A0),
                        .A1  (user_A1),
                        .A2  (user_A2),
                        .A3  (user_A3),
                        .A4  (user_A4),
                        .A5 (user_A5),
                        .A6 (user_A6),
                        .A7 (user_A7));

defparam user_instance_name.INIT = hex_value;
```

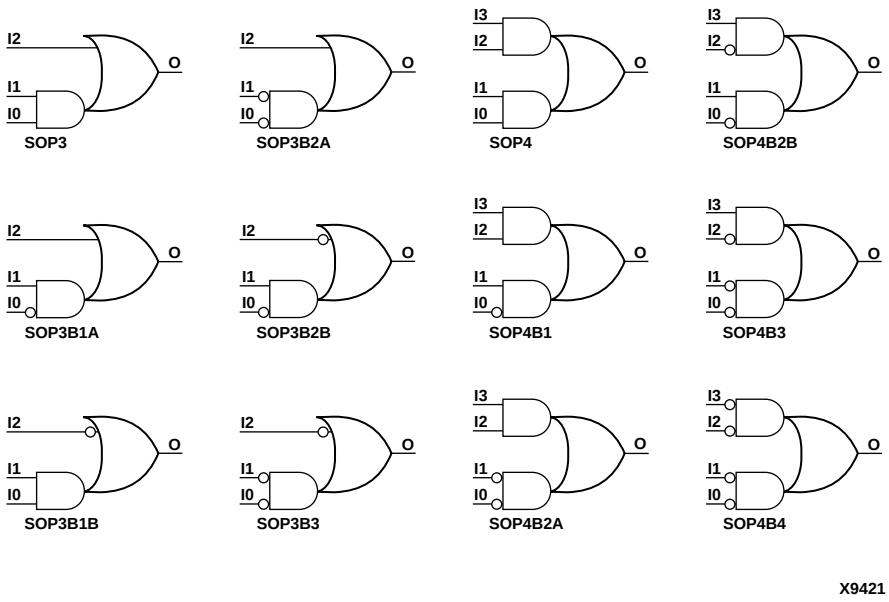
Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKN

SOP3-4

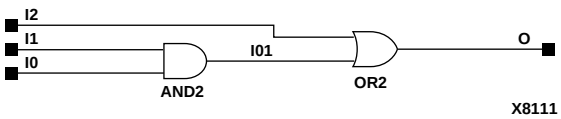
Sum of Products

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
SOP3, SOP3B1A, SOP3B1B, SOP3B2A, SOP3B2B, SOP3B3 SOP4, SOP4B1, SOP4B2A, SOP4B2B, SOP4B3, SOP4B4	Macro	Macro	Macro	Macro	Macro	Macro

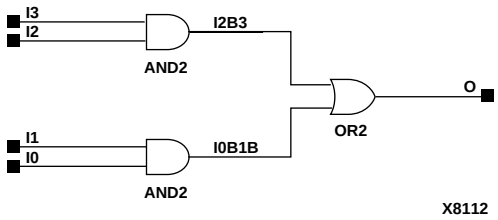


SOP Gate Representations

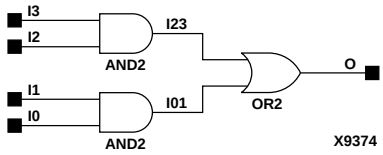
Sum Of Products (SOP) macros provide common logic functions by OR gating the outputs of two AND functions or the output of one AND function with one direct input. Variations of inverting and non-inverting inputs are available.



SOP3 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



SOP4 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E

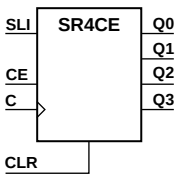


SOP4 Implementation Virtex-II, Virtex-II Pro

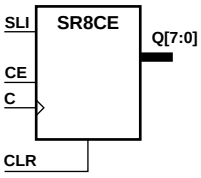
SR4CE, SR8CE, SR16CE

4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear

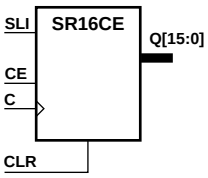
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



X4145



X4151



X4157

SR4CE, SR8CE, and SR16CE are 4-, 8-, and 16-bit shift registers, respectively, with a shift-left serial input (SLI), parallel outputs (Q), and clock enable (CE) and asynchronous clear (CLR) inputs. The CLR input, when High, overrides all other inputs and resets the data outputs (Q) Low. When CE is High and CLR is Low, the data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent Low-to-High clock transitions, when CE is High and CLR is Low, data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth). The register ignores clock transitions when CE is Low.

Registers can be cascaded by connecting the last Q output (Q3 for SR4CE, Q7 for SR8CE, or Q15 for SR16CE) of one stage to the SLI input of the next stage and connecting clock, CE, and CLR in parallel.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

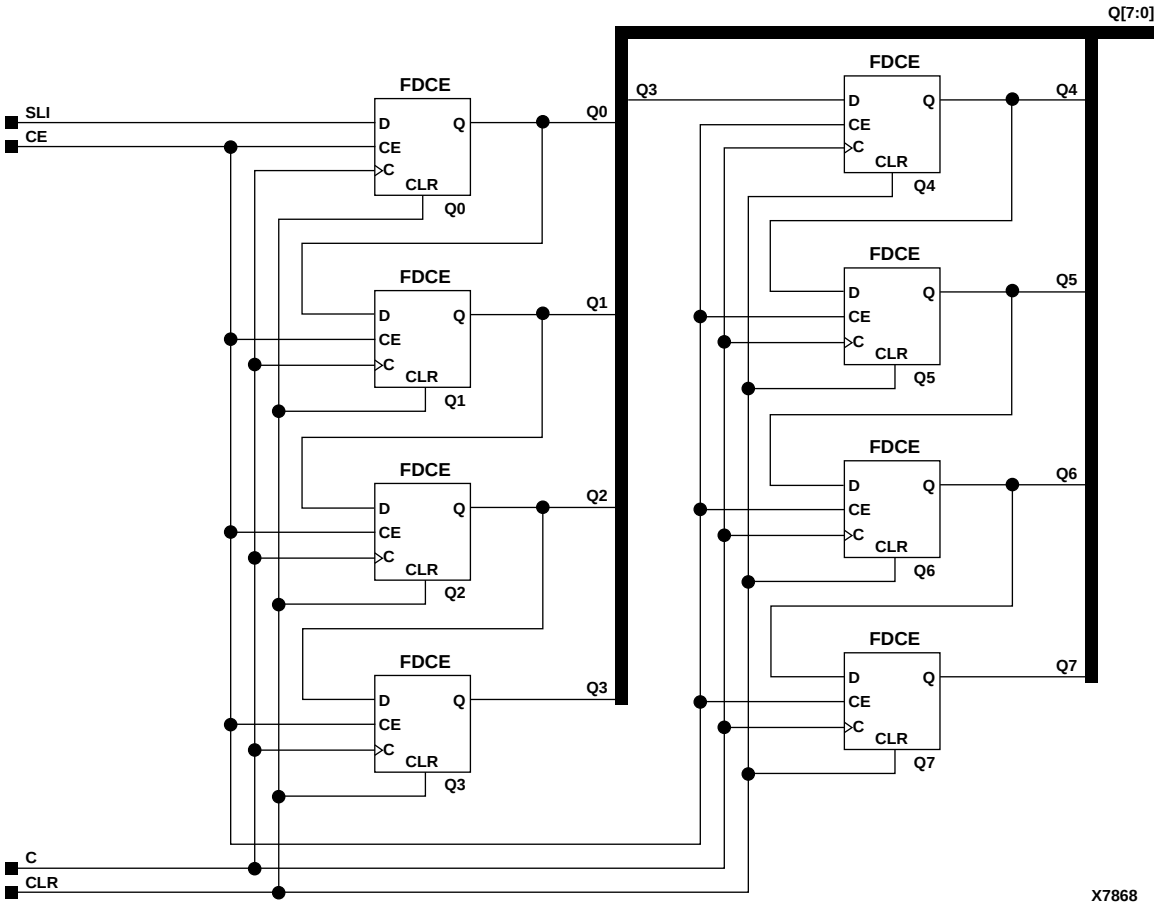
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs	
CLR	CE	SLI	C	Q0	Qz – Q1
1	X	X	X	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	0	↑	0	qn-1

z = 3 for SR4CE; z = 7for SR8CE; z = 15 for SR16CE

qn-1 = state of referenced output one setup time prior to active clock transition



SR8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of sr4ce is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C, CLR)
begin
 if (CLR='1') then
  Q_INT <= (others => '0');
 elsif (C'event and C='1') then
  if (CE='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  end if;
 end if;
end process;

Q <= Q_INT;

end Behavioral;
```

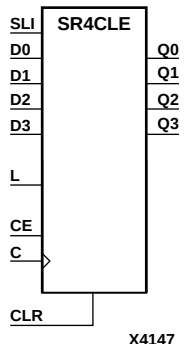
Verilog Inference Code

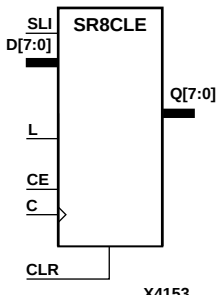
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
    Q <= {Q[WIDTH-2:0],SLI};
end
```

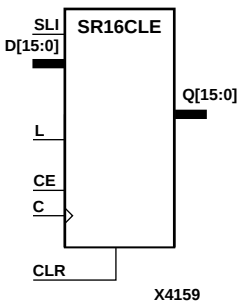
SR4CLE, SR8CLE, SR16CLE

4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







SR4CLE, SR8CLE, and SR16CLE are 4-, 8-, and 16-bit shift registers, respectively, with a shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and asynchronous clear (CLR).The register ignores clock transitions when L and CE are Low. The asynchronous CLR, when High, overrides all other inputs and resets the data outputs (Q) Low. When L is High and CLR is Low, data on the Dn – D0 inputs is loaded into the corresponding Qn – Q0 bits of the register. When CE is High and L and CLR are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and CLR are Low, the data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth).

Registers can be cascaded by connecting the last Q output (Q3 for SR4CLE, Q7 for SR8CLE, or Q15 for SR16CLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and CLR inputs in parallel.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

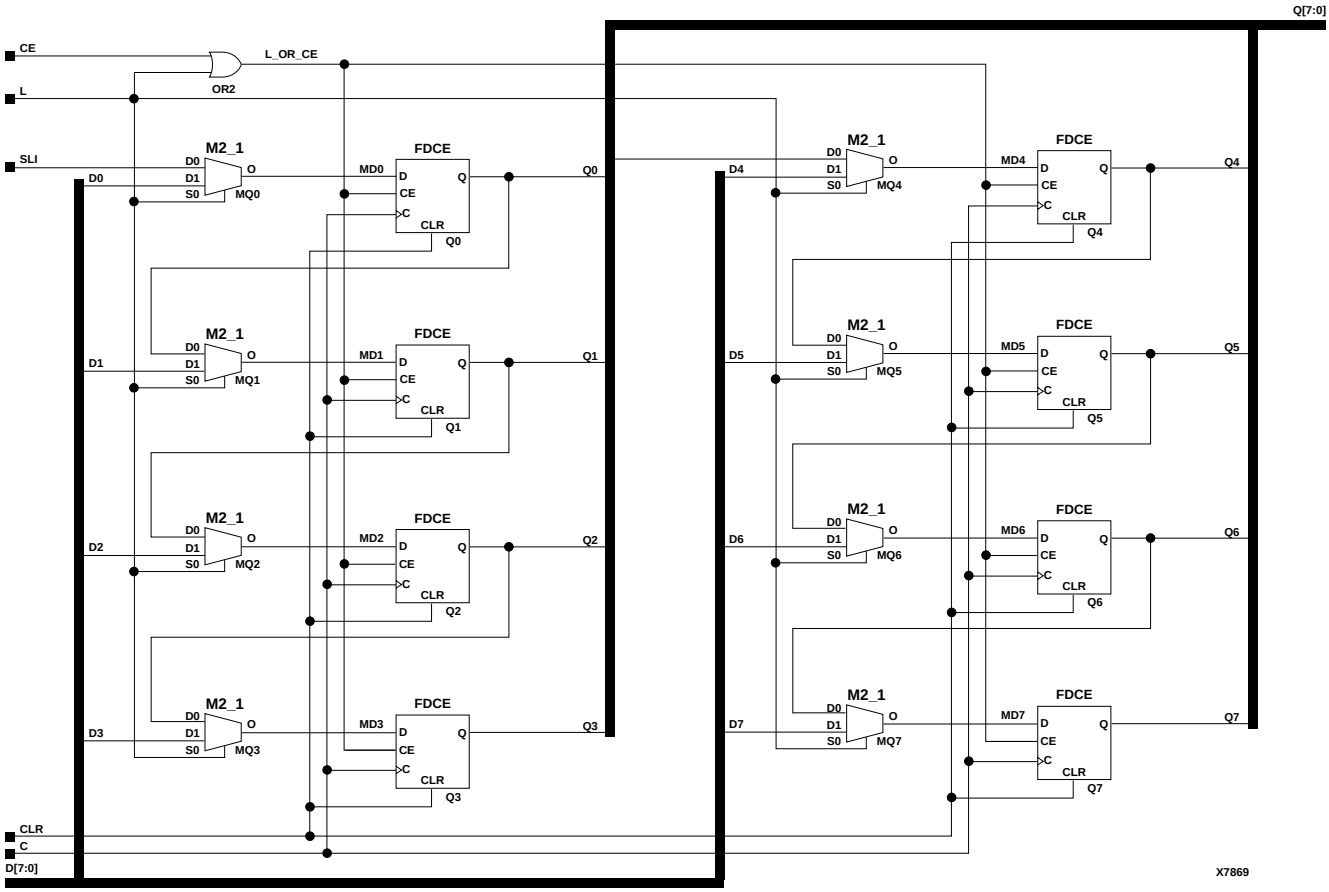
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs						Outputs	
CLR	L	CE	SLI	Dn – D0	C	Q0	Qz – Q1
1	X	X	X	X	X	0	0
0	1	X	X	Dn – D0	↑	d0	dn
0	0	1	SLI	X	↑	SLI	qn-1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SR4CLE; z = 7 for SR8CLE; z = 15 for SR16CLE

dn = state of referenced input one setup time prior to active clock transition

qn-1 = state of referenced output one setup time prior to active clock transition



SR8CLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of sr4cle is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C, CLR)
begin
 if (CLR='1') then
  Q_INT <= (others => '0');
 elsif (C'event and C='1') then
  if (CE='1') then
  if (L='1') then
   Q_INT <= D;
  else
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  end if;
  end if;
```

```
 end if;
end process;

Q <= Q_INT;

end Behavioral;
```

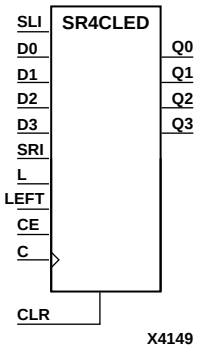
Verilog Inference Code

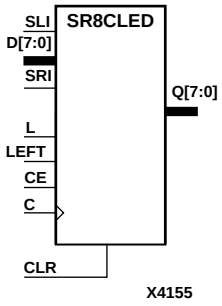
```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
 if (L)
   Q <= D;
 else
   Q <= {Q[WIDTH-2:0],SLI};
end
```

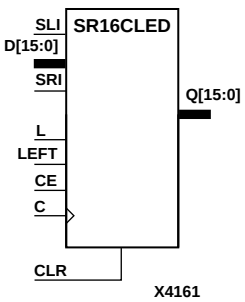
SR4CLED, SR8CLED, SR16CLED

4-, 8-, 16-Bit Shift Registers with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







SR4CLED, SR8CLED, and SR16CLED are 4-, 8-, and 16-bit shift registers, respectively, with shift-left (SLI) and shift-right (SRI) serial inputs, parallel inputs (D), parallel outputs (Q), and four control inputs: clock enable (CE), load enable (L), shift left / right (LEFT), and asynchronous clear (CLR). The register ignores clock transitions when CE and L are Low. The asynchronous clear, when High, overrides all other inputs and resets the data outputs (Qn) Low. When L is High and CLR is Low, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and CLR are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on the SLI is loaded into Q0 during the Low-to-High clock transition and shifted left (to Q1, Q2, and so forth) during subsequent clock transitions. If LEFT is Low, data on the SRI is loaded into the last Q output (Q3 for SR4CLED, Q7 for SR8CLED, or Q15 for SR16CLED) during the Low-to-High clock transition and shifted right (to Q2, Q1,... for SR4CLED; to Q6, Q5,... for SR8CLED; and to Q14, Q13,... for SR16CLED) during subsequent clock transitions. The truth tables for SR4CLED, SR8CLED, and SR16CLED indicate the state of the Q outputs under all input conditions for SR4CLED, SR8CLED, and SR16CLED.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

SR4CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D3– D0	↑	d0	d3	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

qn-1 and qn+1 = state of referenced output one setup time prior to active clock transition

SR8CLED Truth Table

Inputs								Outputs		
CLR	**L**	**CE**	**LEFT**	**SLI**	**SRI**	**D7 – D0**	**C**	**Q0**	**Q7**	**Q6 – Q1**
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D7 – D0	↑	d0	d7	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

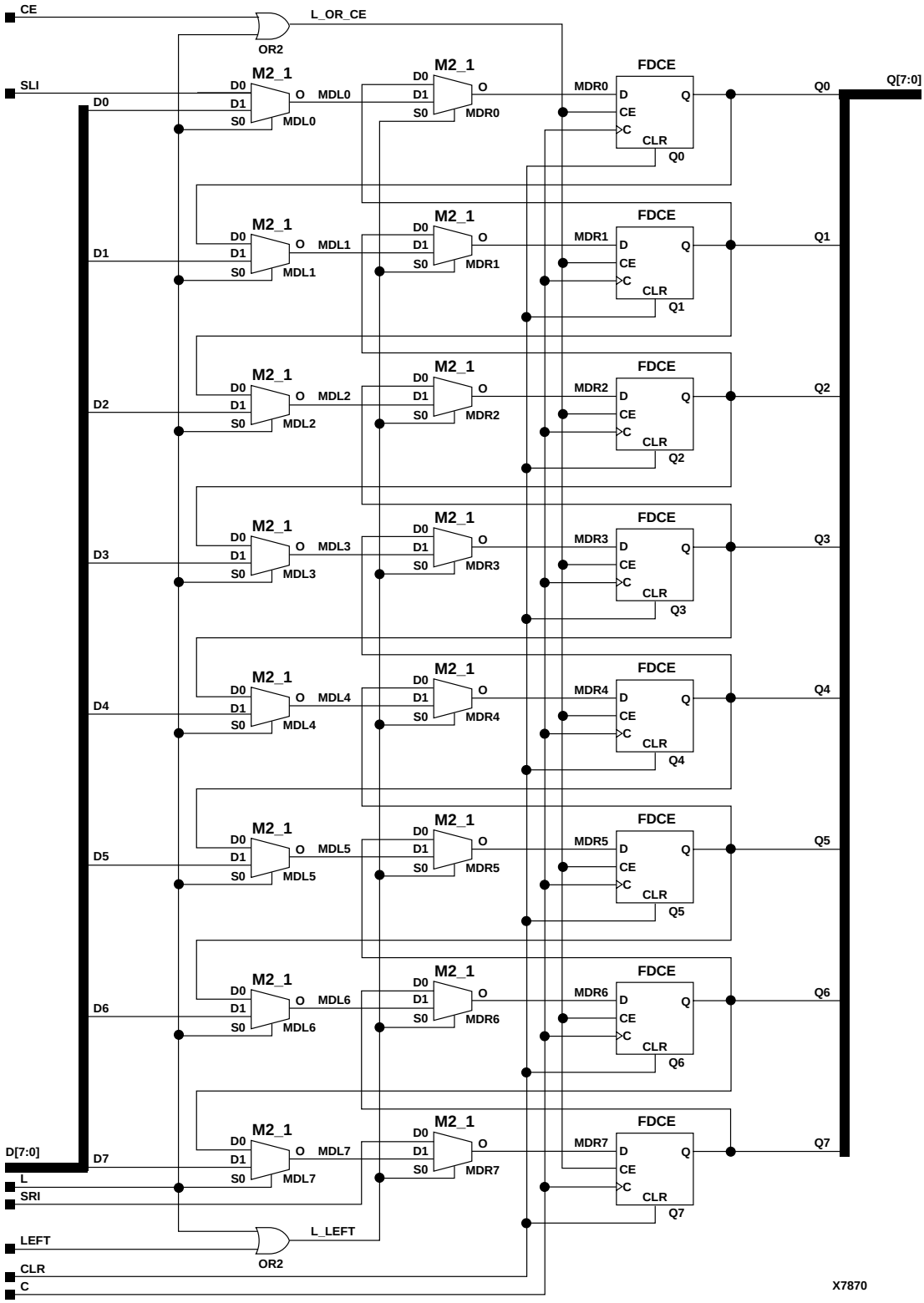
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SR16CLED Truth Table

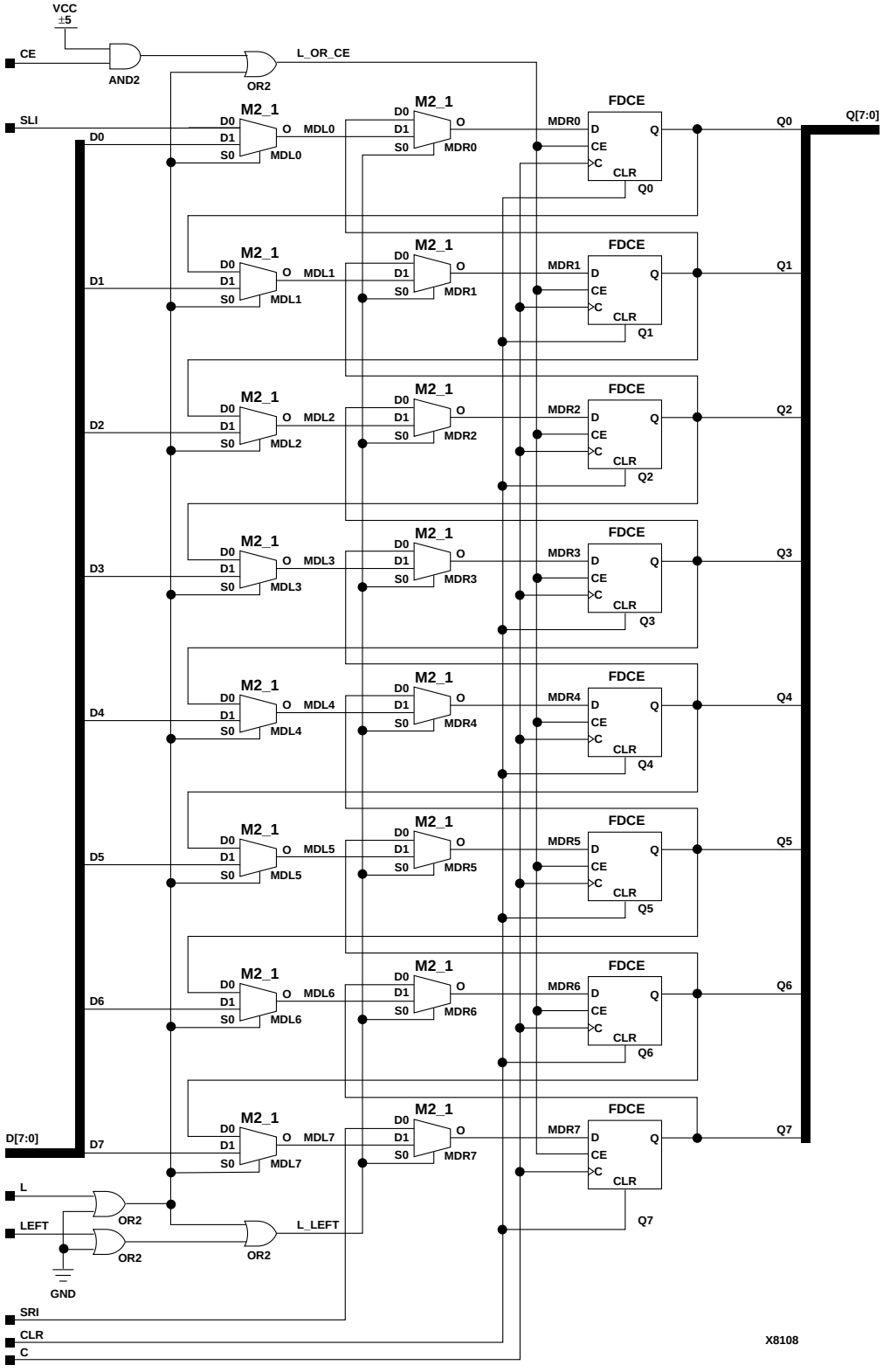
Inputs								Outputs		
CLR	**L**	**CE**	**LEFT**	**SLI**	**SRI**	**D15 – D0**	**C**	**Q0**	**Q15**	**Q14 – Q1**
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D15 – D0	↑	d0	d15	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition



SR8CLED Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro



SR8CLED Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of sr4cled is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C, CLR)
begin
 if (CLR='1') then
  Q_INT <= (others => '0');
 elsif (C'event and C='1') then
  if (CE='1') then
  if (L='1') then
   Q_INT <= D;
  else
  if (LEFT='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  else
   Q_INT <= SRI & Q_INT(WIDTH-1 downto 1);
  end if;
  end if;
 end if;
 end if;
end process;

Q <= Q_INT;

end Behavioral;
```

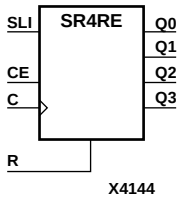
Verilog Inference Code

```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
 if (L)
   Q <= D;
 else if (LEFT)
   Q <= {Q[WIDTH-2:0],SLI};
 else
   Q <= {SRI, Q[WIDTH-1:1]};
end
```

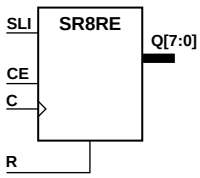
SR4RE, SR8RE, SR16RE

4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset

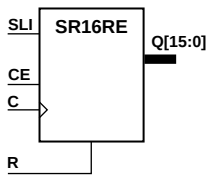
Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro



X4144



X4150



X4156

SR4RE, SR8RE, and SR16RE are 4-, 8-, and 16-bit shift registers, respectively, with shift-left serial input (SLI), parallel outputs (Qn), clock enable (CE), and synchronous reset (R) inputs. The R input, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When CE is High and R is Low, the data on the SLI is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent Low-to-High clock transitions, when CE is High and R is Low, data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth). The register ignores clock transitions when CE is Low.

Registers can be cascaded by connecting the last Q output (Q3 for SR4RE, Q7 for SR8RE, or Q15 for SR16RE) of one stage to the SLI input of the next stage and connecting clock, CE, and R in parallel.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

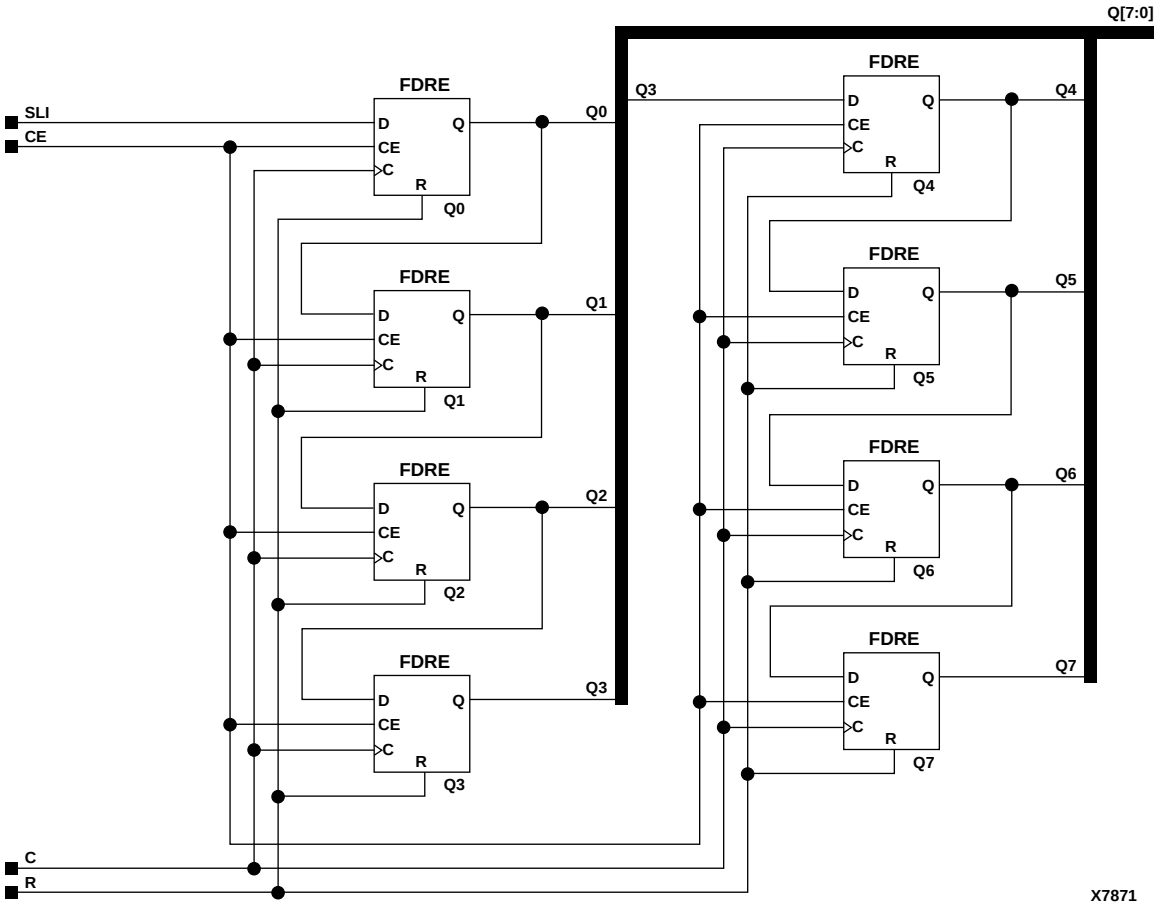
Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs	
R	CE	SLI	C	Q0	Qz – Q1
1	X	X	↑	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	0	↑	0	qn-1

z = 3 for SR4RE; z = 7 for SR8RE; z = 15 for SR16RE

qn-1 = state of referenced output one setup time prior to active clock transition



SR8RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of sr4re is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C)
begin
 if C'event and C='1' then
 if (R='1') then
  Q_INT <= (others => '0');
  elsif (CE='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  end if;
 end if;
end process;

Q <= Q_INT;

end Behavioral;
```

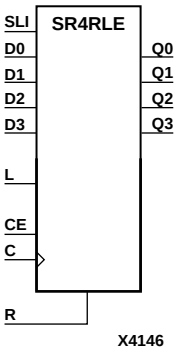
Verilog Inference Code

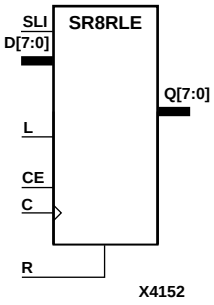
```
always @ (posedge C)
begin
 if (R)
  Q <= 0;
 else if (CE)
    Q <= {Q[WIDTH-2:0],SLI};
end
```

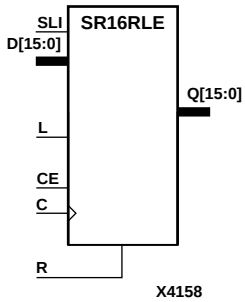
SR4RLE, SR8RLE, SR16RLE

4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







SR4RLE, SR8RLE, and SR16RLE are 4-, 8-, and 16-bit shift registers, respectively, with shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and synchronous reset (R). The register ignores clock transitions when L and CE are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low during the Low-to-High clock transition, data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and R are Low, the data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth).

Registers can be cascaded by connecting the last Q output (Q3 for SR4RLE, Q7 for SR8RLE, or 15 for SR16RLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and R inputs in parallel.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

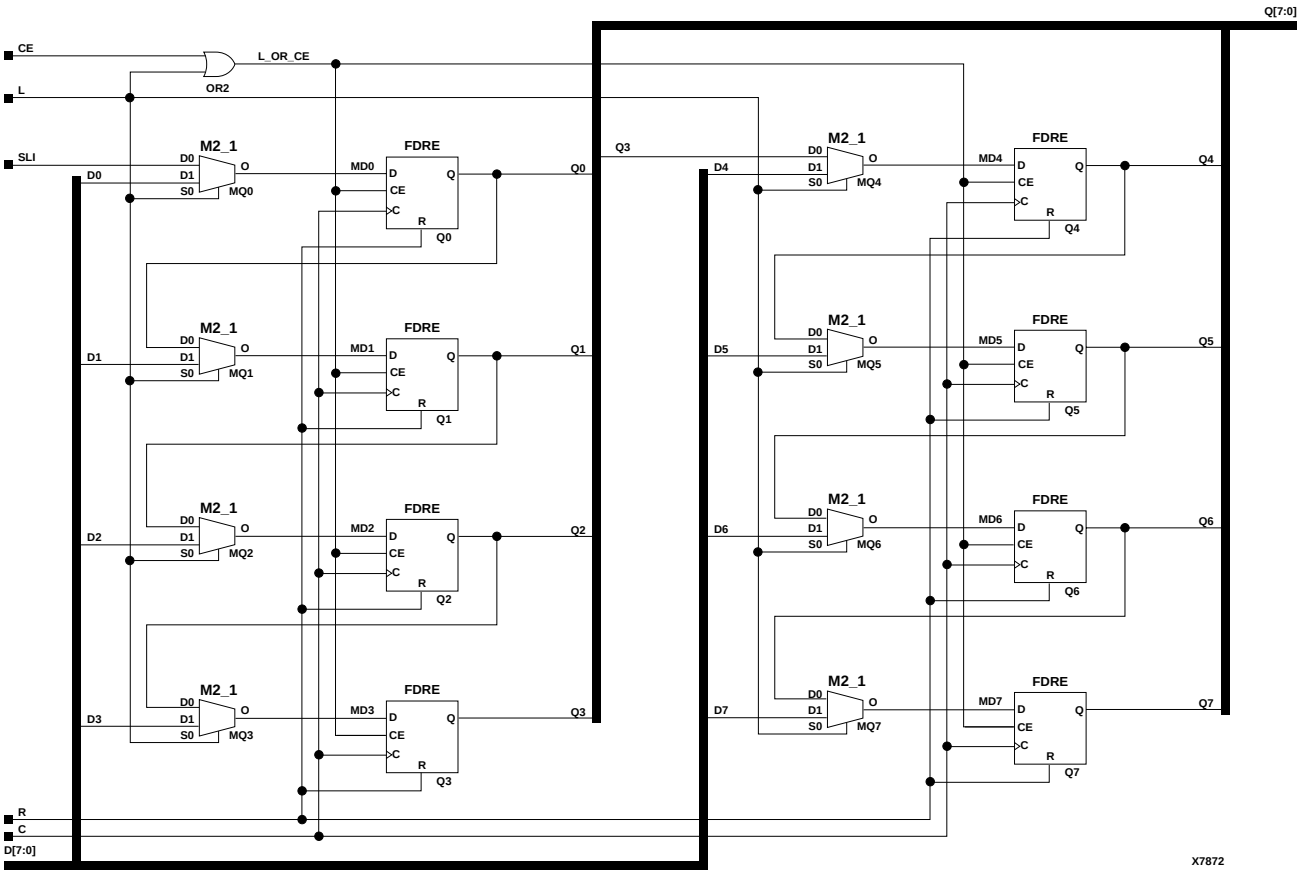
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs						Outputs	
R	L	CE	SLI	Dz – D0	C	Q0	Qz – Q1
1	X	X	X	X	↑	0	0
0	1	X	X	Dz – D0	↑	d0	dn
0	0	1	SLI	X	↑	SLI	qn-1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SR4RLE; z = 7 for SR8RLE; z = 15 for SR16RLE

dn = state of referenced input one setup time prior to active clock transition

qn-1 = state of referenced output one setup time prior to active clock transition



SR8RLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of sr4rle is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C)
begin
 if C'event and C='1' then
 if (R='1') then
  Q_INT <= (others => '0');
  if (CE='1') then
  if (L='1') then
   Q_INT <= D;
  else
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  end if;
  end if;
```

```
  end if;
 end if;
end process;

Q <= Q_INT;

end Behavioral;
```

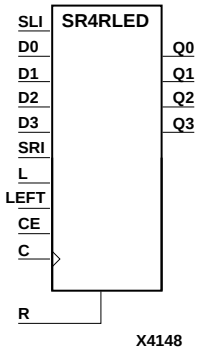
Verilog Inference Code

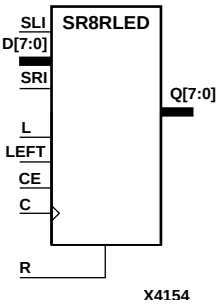
```
always @ (posedge C)
begin
 if (R)
   Q <= 0;
 else if (CE)
 if (L)
   Q <= D;
 else
   Q <= {Q[WIDTH-2:0],SLI};
end
```

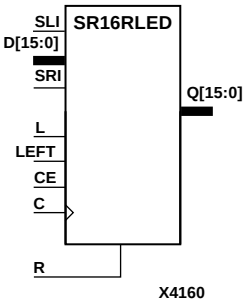
SR4RLED, SR8RLED, SR16RLED

4-, 8-, 16-Bit Shift Registers with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Macro	Macro	Macro	Macro	Macro	Macro







SR4RLED, SR8RLED, and SR16RLED are 4-, 8-, and 16-bit shift registers, respectively, with shift-left (SLI) and shift-right (SRI) serial inputs, parallel inputs (D), parallel outputs (Q) and four control inputs — clock enable (CE), load enable (L), shift left / right (LEFT), and synchronous reset (R). The register ignores clock transitions when CE and L are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low during the Low-to-High clock transition, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on SLI is loaded into Q0 during the Low-to-High clock transition and shifted left (to Q1, Q2, and so forth) during subsequent clock transitions. If LEFT is Low, data on the SRI is loaded into the last Q output (Q3 for SR4RLED, Q7 for SR8RLED, or Q15 for SR16RLED) during the Low-to-High clock transition and shifted right (to Q2, Q1,... for SR4RLED; to Q6, Q5,... for SR8RLED; or to Q14, Q13,... for SR16RLED) during subsequent clock transitions. The truth table indicates the state of the Q outputs under all input conditions.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500 / XV / XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro simulate power-on when global set / reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

SR4RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D3 – D0	↑	d0	d3	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SR8RLED Truth Table

Inputs								Outputs		
R	**L**	**CE**	**LEFT**	**SLI**	**SRI**	**D7– D0**	**C**	**Q0**	**Q7**	**Q6 – Q1**
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D7 – D0	↑	d0	d7	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

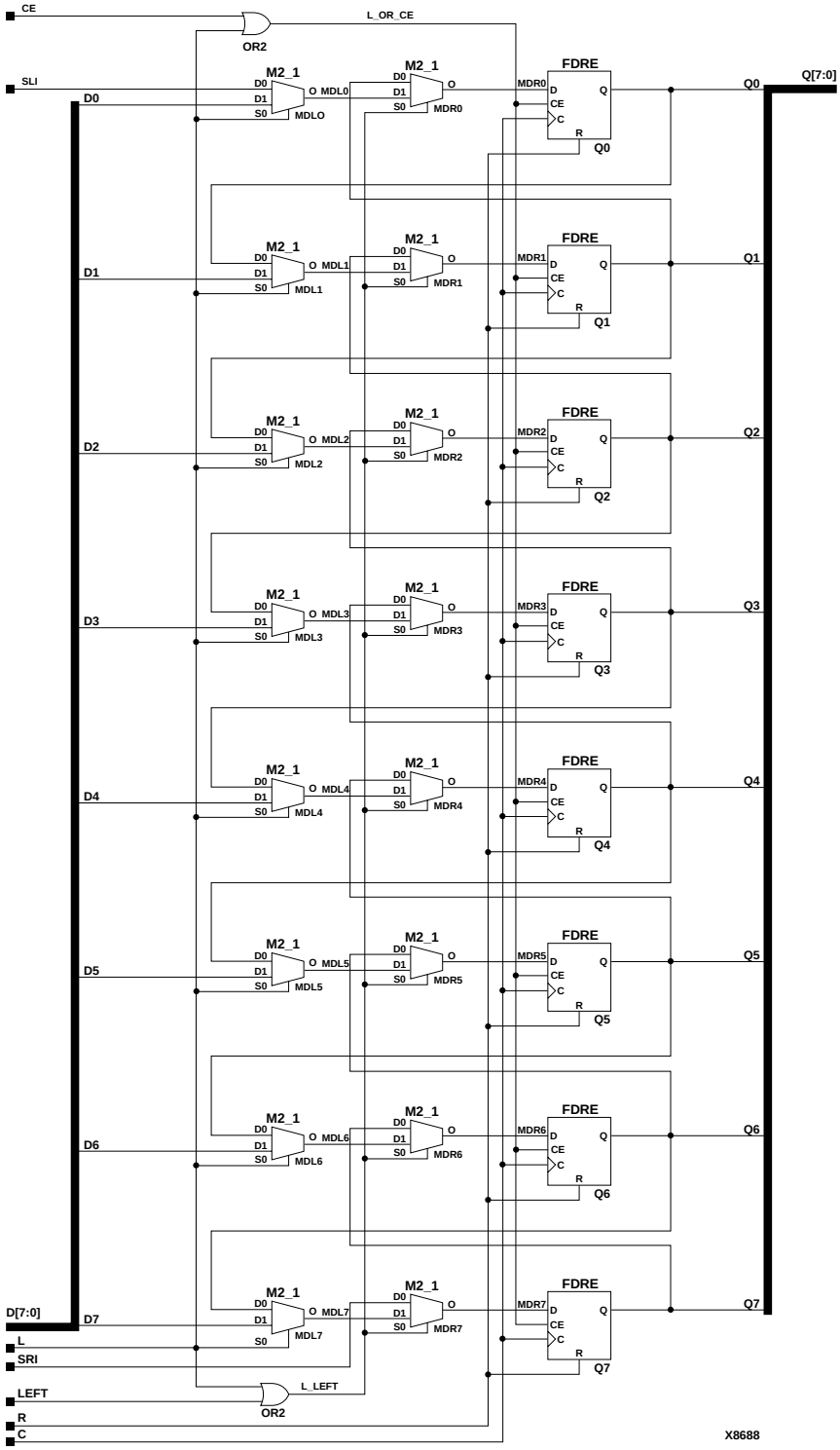
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SR16RLED Truth Table

Inputs								Outputs		
R	**L**	**CE**	**LEFT**	**SLI**	**SRI**	**D15 – D0**	**C**	**Q0**	**Q15**	**Q14 – Q1**
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D15 – D0	↑	d0	d15	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition



SR8RLED Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of sr4rled is

begin

process(C)
begin
        if (C'event and C='1') then
               if (R='1') then
                        Q <= (others => '0');
                elsif (CE='1') then
                        if (L='1') then
                                Q <= D;
                        else
                                if (LEFT='1') then
                                        Q <= Q(WIDTH-2 downto 0) & SLI;
                                else
                                        Q <= SRI & Q(WIDTH-1 downto 1) ;
                                end if;
                        end if;
                end if;
        end if;
end process;

end Behavioral;
```

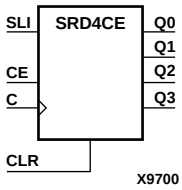
Verilog Inference Code

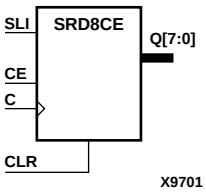
```
always @ (posedge C)
begin
        if (R)
                Q <= 0;
        else if (CE)
                if (L)
                        Q <= D;
                else if (LEFT)
                        Q <= {Q[WIDTH-2:0], SLI};
                else
                        Q <= {SRI, Q[WIDTH-1:1]};
end
```

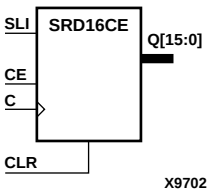
SRD4CE, SRD8CE, SRD16CE

4-, 8-, 16-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







SRD4CE, SRD8CE, and SRD16CE are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with a shift-left serial input (SLI), parallel outputs (Q), clock enable (CE) and asynchronous clear (CLR) inputs. The CLR input, when High, overrides all other inputs and resets the data outputs (Q) Low. When CE is High and CLR is Low, the data on the SLI input is loaded into the first bit of the shift register during the Low-to-High (or High-to-Low) clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and CLR is Low, data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth). The register ignores clock transitions when CE is Low.

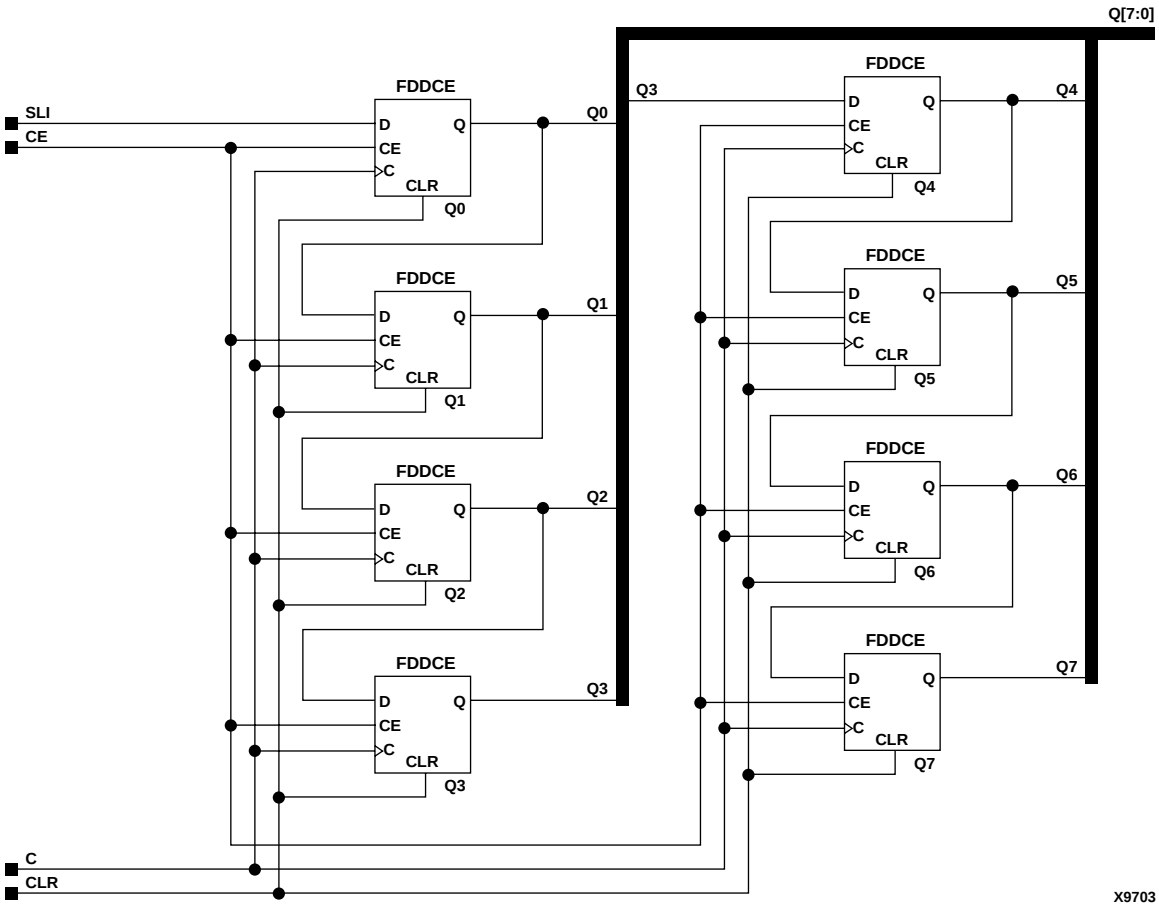
Registers can be cascaded by connecting the last Q output (Q3 for SRD4CE, Q7 for SRD8CE, or Q15 for SRD16CE) of one stage to the SLI input of the next stage and connecting clock, CE, and CLR in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs	
CLR	CE	SLI	C	Q0	Qz – Q1
1	X	X	X	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	1	↓	1	qn-1
0	1	0	↑	0	qn-1
0	1	0	↓	0	qn-1

z = 3 for SRD4CE; z = 7for SRD8CE; z = 15 for SRD16CE

qn-1 = state of referenced output one setup time prior to active clock transition



SRD8CE Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of srd4ce is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C, CLR)
begin
 if (CLR='1') then
  Q_INT <= (others => '0');
 elsif (C'event) then
  if (CE='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  end if;
 end if;
end process;

Q <= Q_INT;

end Behavioral;
```

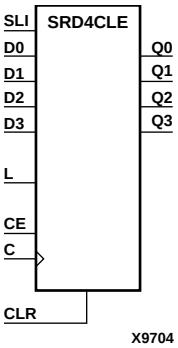
Verilog Inference Code

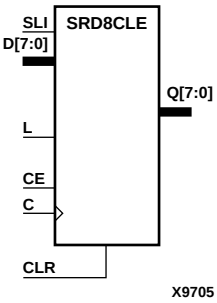
```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
    Q <= {Q[WIDTH-2:0],SLI};
end
```

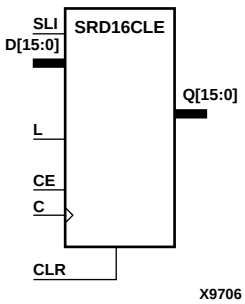
SRD4CLE, SRD8CLE, SRD16CLE

4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







SRD4CLE, SRD8CLE, and SRD16CLE are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with a shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and asynchronous clear (CLR). The register ignores clock transitions when L and CE are Low. The asynchronous CLR, when High, overrides all other inputs and resets the data outputs (Q) Low. When L is High and CLR is Low, data on the Dn – D0 inputs is loaded into the corresponding Qn – Q0 bits of the register. When CE is High and L and CLR are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High (or High-to-Low) clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and CLR are Low, the data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth).

Registers can be cascaded by connecting the last Q output (Q3 for SRD4CLE, Q7 for SRD8CLE, or Q15 for SRD16CLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and CLR inputs in parallel.

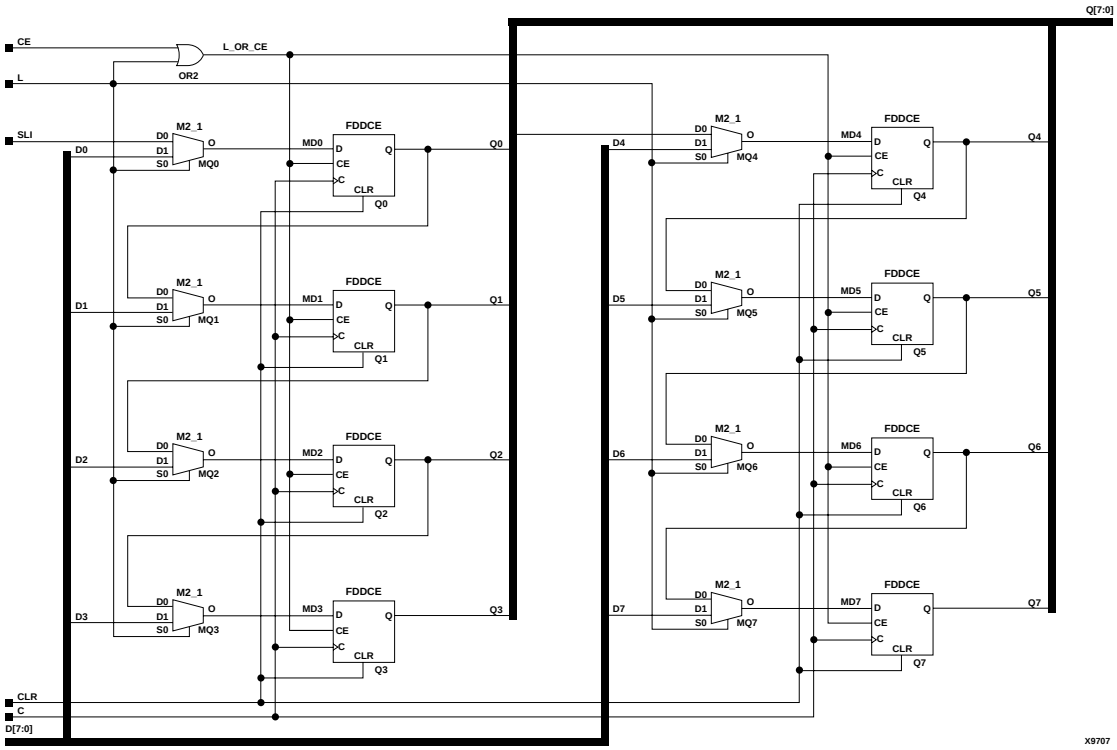
The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs	
CLR	L	CE	SLI	Dn – D0	C	Q0	Qz – Q1
1	X	X	X	X	X	0	0
0	1	X	X	Dn – D0	↑	d0	dn
0	1	X	X	Dn – D0	↓	d0	dn
0	0	1	SLI	X	↑	SLI	qn-1
0	0	1	SLI	X	↓	SLI	qn-1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SRD4CLE; z = 7 for SRD8CLE; z = 15 for SRD16CLE

dn = state of referenced input one setup time prior to active clock transition

qn-1 = state of referenced output one setup time prior to active clock transition



SRD8CLE Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of srd4cle is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C, CLR)
begin
 if (CLR='1') then
  Q_INT <= (others => '0');
 elsif (C'event) then
  if (CE='1') then
  if (L='1') then
   Q_INT <= D;
  else
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  end if;
  end if;
 end if;
end process;

Q <= Q_INT;
end Behavioral;
```

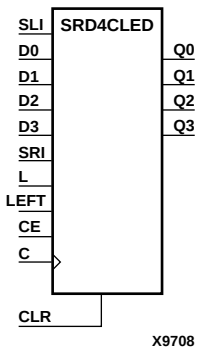
Verilog Inference Code

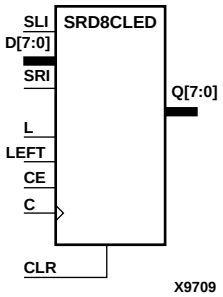
```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
   if (L)
      Q <= D;
 else
   Q <= {Q[WIDTH-2:0],SLI};
end
```

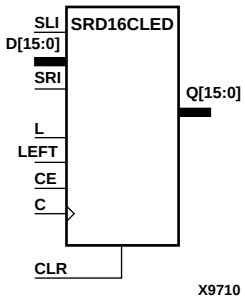
SRD4CLED, SRD8CLED, SRD16CLED

4-, 8-, 16-Bit Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







SRD4CLED, SRD8CLED, and SRD16CLED are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with shift-left (SLI) and shift-right (SRI) serial inputs, parallel inputs (D), parallel outputs (Q), and four control inputs: clock enable (CE), load enable (L), shift left / right (LEFT), and asynchronous clear (CLR). The register ignores clock transitions when CE and L are Low. The asynchronous clear, when High, overrides all other inputs and resets the data outputs (Qn) Low. When L is High and CLR is Low, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and CLR are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on the SLI is loaded into Q0 during the Low-to-High or High-to-Low clock transition and shifted left (to Q1, Q2, and so forth) during subsequent clock transitions. If LEFT is Low, data on the SRI is loaded into the last Q output (Q3 for SRD4CLED, Q7 for SRD8CLED, or Q15 for SRD16CLED) during the Low-to-High or High-to-Low clock transition and shifted right (to Q2, Q1,... for SRD4CLED; to Q6, Q5,... for SRD8CLED; and to Q14, Q13,... for SRD16CLED) during subsequent clock transitions. The truth tables for SRD4CLED, SRD8CLED, and SRD16CLED indicate the state of the Q outputs under all input conditions for SRD4CLED, SRD8CLED, and SRD16CLED.

The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

SRD4CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D3– D0	↑	d0	d3	dn
0	1	X	X	X	X	D3– D0	↓	d0	d3	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q2	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1
0	0	1	0	X	SRI	X	↓	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

qn-1 and qn+1 = state of referenced output one setup time prior to active clock transition

SRD8CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D7 – D0	C	Q0	Q7	Q6 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D7 – D0	↑	d0	d7	dn
0	1	X	X	X	X	D7 – D0	↓	d0	d7	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q6	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1
0	0	1	0	X	SRI	X	↓	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

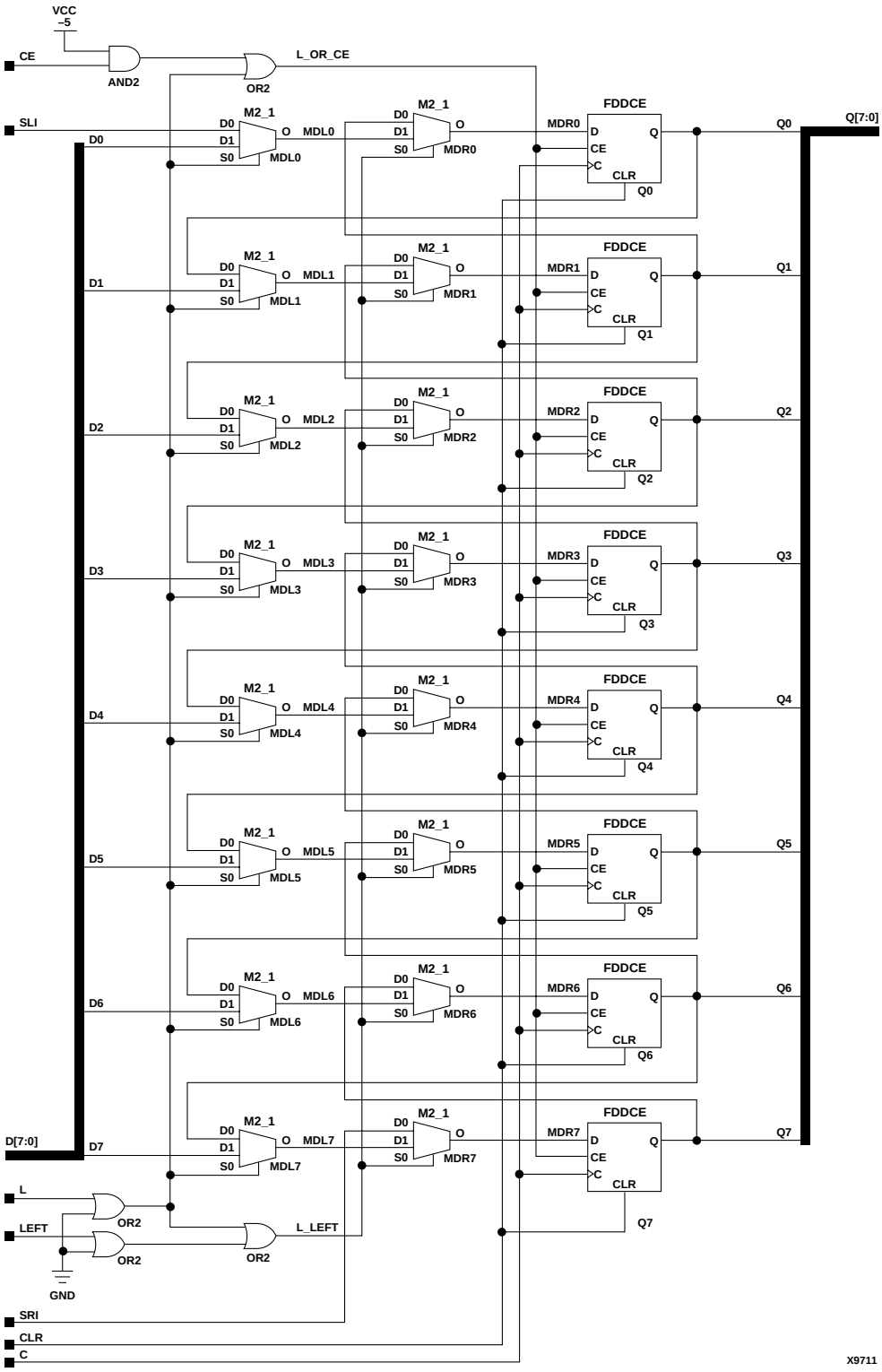
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SRD16CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D15 – D0	C	Q0	Q15	Q14 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D15 – D0	↑	d0	d15	dn
0	1	X	X	X	X	D15 – D0	↓	d0	d15	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q14	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1
0	0	1	0	X	SRI	X	↓	q1	SRI	qn+1

dn = state of referenced input one setup time prior to active clock transition

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition



SRD8CLED Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of srd4cled is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C, CLR)
begin
 if (CLR='1') then
  Q_INT <= (others => '0');
 elsif (C'event) then
  if (CE='1') then
  if (L='1') then
   Q_INT <= D;
  else
  if (LEFT='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  else
   Q_INT <= SRI & Q_INT(WIDTH-1 downto 1);
  end if;
  end if;
 end if;
 end if;
end process;

Q <= Q_INT;

end Behavioral;
```

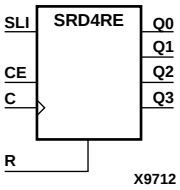
Verilog Inference Code

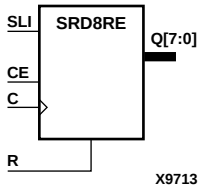
```
always @ (posedge C or negedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (CE)
 if (L)
   Q <= D;
 else if (LEFT)
   Q <= {Q[WIDTH-2:0],SLI};
 else
   Q <= {SRI, Q[WIDTH-1:1]};

end
```

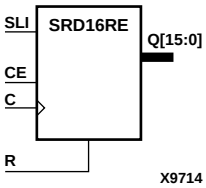
SRD4RE, SRD8RE, SRD16RE

4-, 8-, 16-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







SRD4RE, SRD8RE, and SRD16RE are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with shift-left serial input (SLI), parallel outputs (Qn), clock enable (CE), and synchronous reset (R) inputs. The R input, when High, overrides all other inputs during the Low-to-High or High-to-Low clock (C) transition and resets the data outputs (Q) Low. When CE is High and R is Low, the data on the SLI is loaded into the first bit of the shift register during the Low-to-High clock or High-to-Low (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and R is Low, data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth). The register ignores clock transitions when CE is Low.

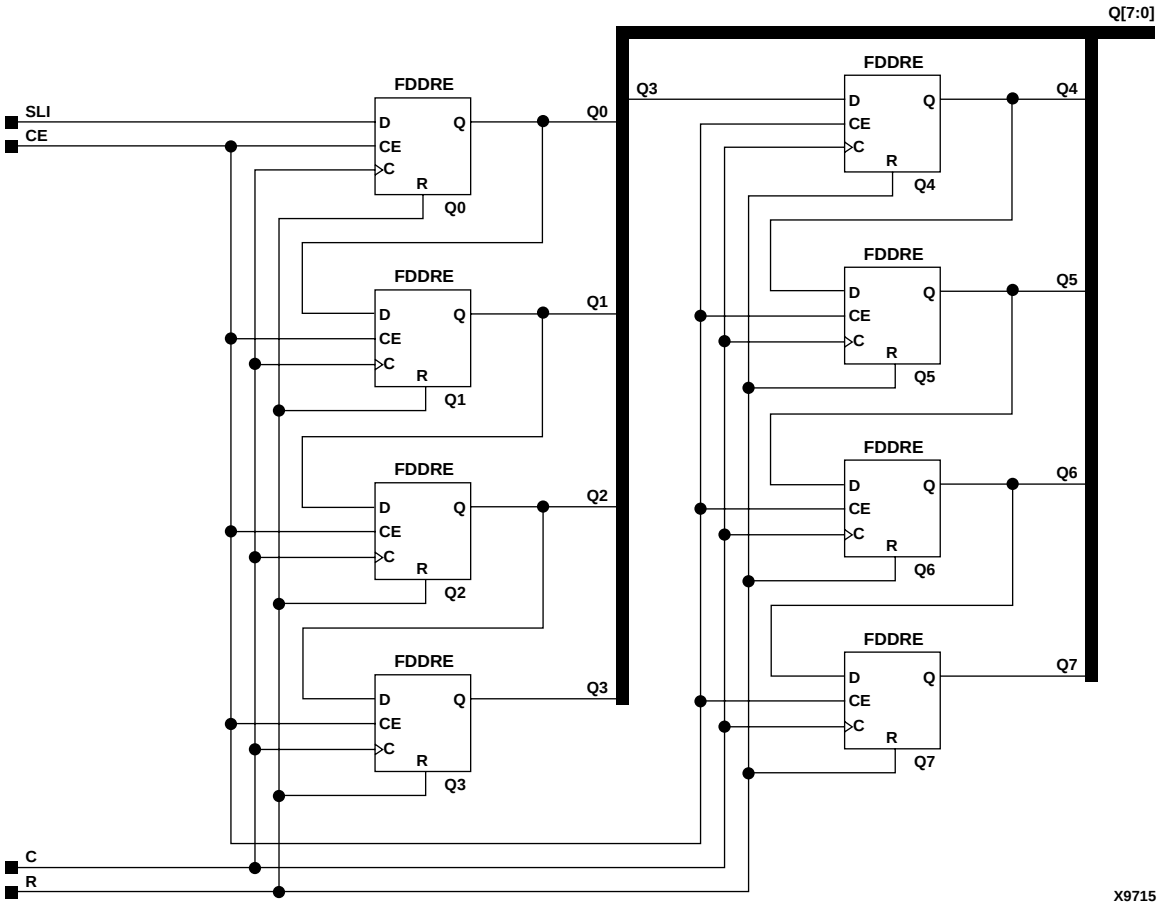
Registers can be cascaded by connecting the last Q output (Q3 for SRD4RE, Q7 for SRD8RE, or Q15 for SRD16RE) of one stage to the SLI input of the next stage and connecting clock, CE, and R in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs	
R	CE	SLI	C	Q0	Qz – Q1
1	X	X	↑	0	0
1	X	X	↓	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	1	↓	1	qn-1
0	1	0	↑	0	qn-1
0	1	0	↓	0	qn-1

z = 3 for SRD4RE; z = 7 for SRD8RE; z = 15 for SRD16RE

qn-1 = state of referenced output one setup time prior to active clock transition



SRD8RE Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of srd4re is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C)
begin
 if C'event then
 if (R='1') then
  Q_INT <= (others => '0');
  elsif (CE='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  end if;
 end if;
end process;

Q <= Q_INT;
end Behavioral;
```

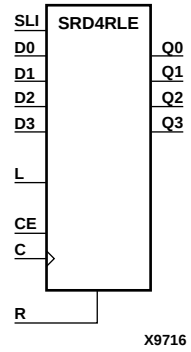
Verilog Inference Code

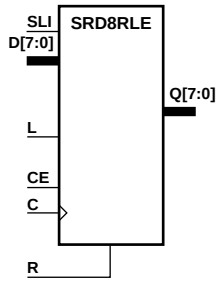
```
always @ (posedge C or negedge C)
begin
 if (R)
   Q <= 0;
 else if (CE)
   Q <= {Q[WIDTH-2:0],SLI};
end
```

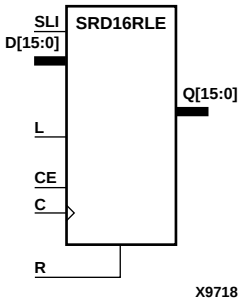
SRD4RLE, SRD8RLE, SRD16RLE

4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







SRD4RLE, SRD8RLE, and SRD16RLE are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and synchronous reset (R). The register ignores clock transitions when L and CE are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High or High-to-Low clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low, data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High or High-to-Low clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and R are Low, the data is shifted to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, and so forth).

Registers can be cascaded by connecting the last Q output (Q3 for SRD4RLE, Q7 for SRD8RLE, or 15 for SRD16RLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and R inputs in parallel.

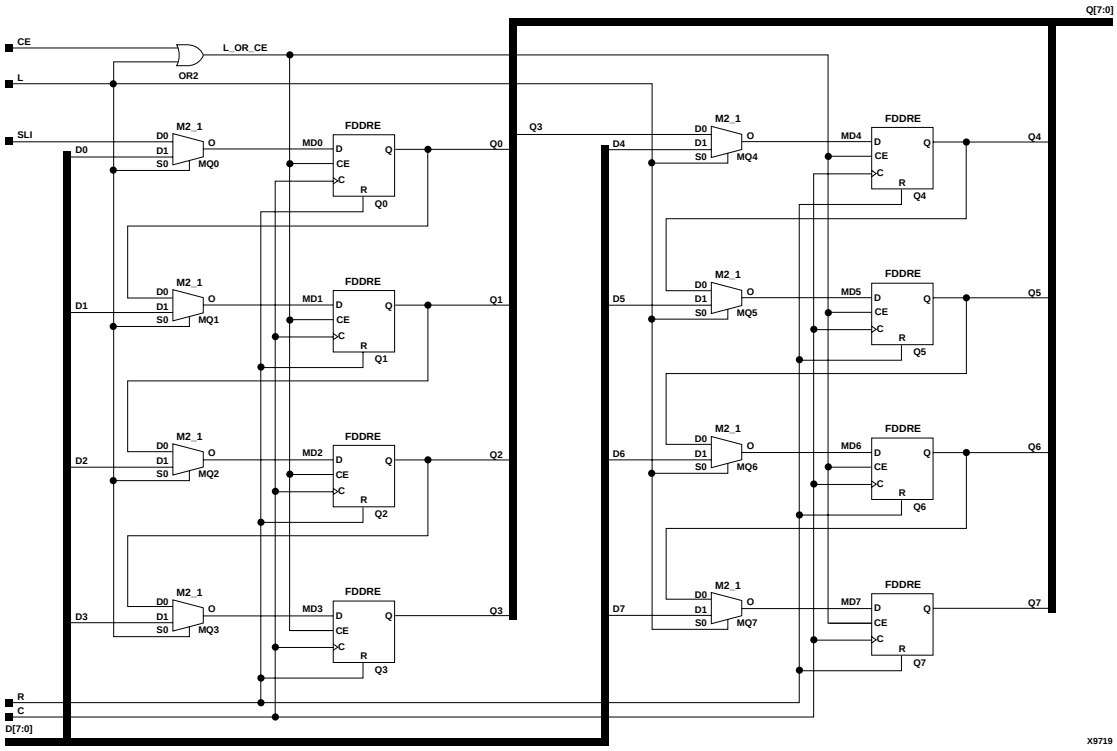
The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs	
R	L	CE	SLI	Dz – D0	C	Q0	Qz – Q1
1	X	X	X	X	↑	0	0
1	X	X	X	X	↓	0	0
0	1	X	X	Dz – D0	↑	d0	dn
0	1	X	X	Dz – D0	↓	d0	dn
0	0	1	SLI	X	↑	SLI	qn-1
0	0	1	SLI	X	↓	SLI	qn-1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SRD4RLE; z = 7 for SRD8RLE; z = 15 for SRD16RLE

dn = state of referenced input one setup time prior to active clock transition

qn-1 = state of referenced output one setup time prior to active clock transition



SRD8RLE Implementation CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of srd4rle is
signal Q_INT: std_logic_vector(WIDTH-1 downto 0);
begin

process(C)
begin
 if (C'event) then
 if (R='1') then
  Q_INT <= (others => '0');
  if (CE='1') then
  if (L='1') then
   Q_INT <= D;
  else
   Q_INT <= Q_INT(WIDTH-2 downto 0) & SLI;
  end if;
  end if;
  end if;
 end if;
end process;

Q <= Q_INT;

end Behavioral;
```

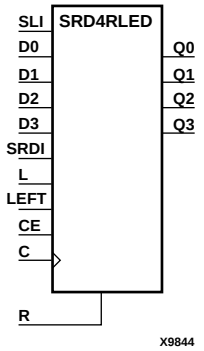
Verilog Inference Code

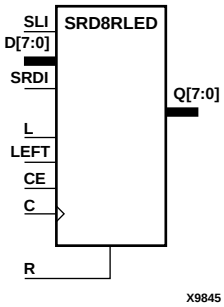
```
always @ (posedge C or negedge C)
begin
 if (R)
  Q <= 0;
 else if (CE)
   if (L)
      Q <= D;
 else
   Q <= {Q[WIDTH-2:0],SLI};
end
```

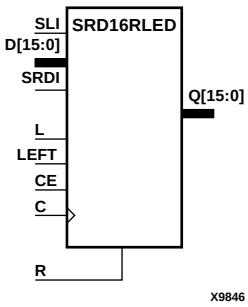
SRD4RLED, SRD8RLED, SRD16RLED

4-, 8-, 16-Bit Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N / A	N / A	N / A	N / A	N / A	Macro







SRD4RLED, SRD8RLED, and SRD16RLED are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with shift-left (SLI) and shift-right (SRDI) serial inputs, parallel inputs (D), parallel outputs (Q), and four control inputs — clock enable (CE), load enable (L), shift left / right (LEFT), and synchronous reset (R). The register ignores clock transitions when CE and L are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High or High-to-Low clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on SLI is loaded into Q0 during the Low-to-High or High-to-Low clock transition and shifted left (to Q1, Q2, and so forth) during subsequent clock transitions. If LEFT is Low, data on the SRDI is loaded into the last Q output (Q3 for SRD4RLED, Q7 for SRD8RLED, or Q15 for SRD16RLED) during the Low-to-High or High-to-Low clock transition and shifted right (to Q2, Q1,... for SRD4RLED; to Q6, Q5,... for SRD8RLED; or to Q14, Q13,... for SRD16RLED) during subsequent clock transitions. The truth table indicates the state of the Q outputs under all input conditions.

The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

SRD4RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRDI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	↑	0	0	0
1	X	X	X	X	X	X	↓	0	0	0
0	1	X	X	X	X	D3 – D0	↑	d0	d3	dn
0	1	X	X	X	X	D3 – D0	↓	d0	d3	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q2	qn-1
0	0	1	0	X	SRDI	X	↑	q1	SRDI	qn+1
0	0	1	0	X	SRDI	X	↓	q1	SRDI	qn+1

dn = state of referenced input one setup time prior to active clock transition

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SRD8RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRDI	D7– D0	C	Q0	Q7	Q6 – Q1
1	X	X	X	X	X	X	↑	0	0	0
1	X	X	X	X	X	X	↓	0	0	0
0	1	X	X	X	X	D7 – D0	↑	d0	d7	dn
0	1	X	X	X	X	D7 – D0	↓	d0	d7	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q6	qn-1
0	0	1	0	X	SRDI	X	↑	q1	SRDI	qn+1
0	0	1	0	X	SRDI	X	↓	q1	SRDI	qn+1

dn = state of referenced input one setup time prior to active clock transition

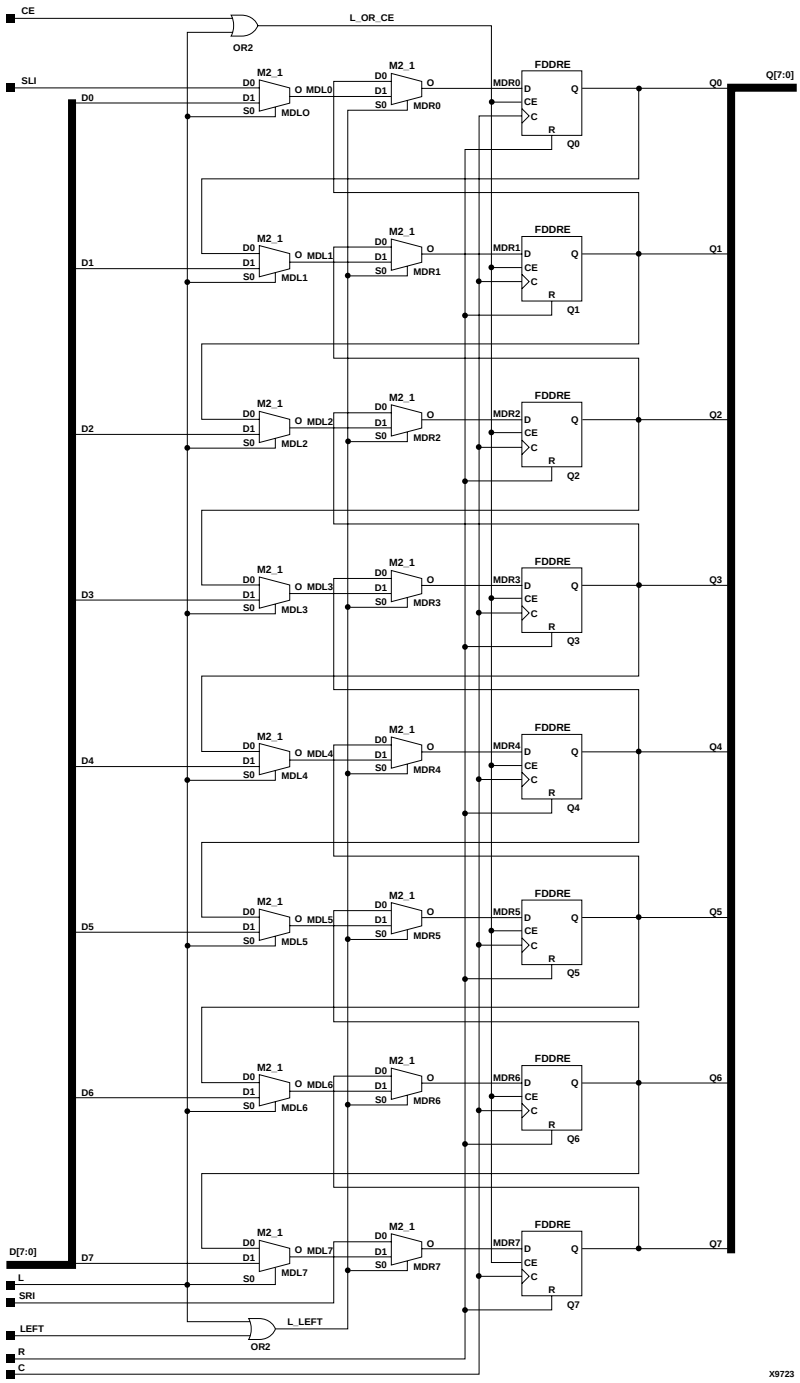
qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SRD16RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRDI	D15 – D0	C	Q0	Q15	Q14 – Q1
1	X	X	X	X	X	X	↑	0	0	0
1	X	X	X	X	X	X	↓	0	0	0
0	1	X	X	X	X	D15 – D0	↑	d0	d15	dn
0	1	X	X	X	X	D15 – D0	↓	d0	d15	dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q14	qn-1
0	0	1	0	X	SRDI	X	↑	q1	SRDI	qn+1
0	0	1	0	X	SRDI	X	↓	q1	SRDI	qn+1

dn = state of referenced input one setup time prior to active clock transition

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

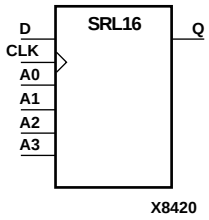


SRD8RLED Implementation CoolRunner-II

SRL16

16-Bit Shift Register Look-Up-Table (LUT)

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



SRL16 is a shift register look up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. During subsequent Low-to-High clock transitions data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

Static Length Mode

To get a fixed length shift register, drive the A3 through A0 inputs with static values. The length of the shift register can vary from 1 bit to 16 bits as determined from the following formula:

```
Length = (8*A3) +(4*A2) + (2*A1) + A0 +1
```

If A3, A2, A1, and A0 are all zeros (0000), the shift register is one bit long. If they are all ones (1111), it is 16 bits long.

Dynamic Length Mode

The length of the shift register can be changed dynamically by changing the values driving the A3 through A0 inputs. For example, if A2, A1, and A0 are all ones (111) and A3 toggles between a one (1) and a zero (0), the length of the shift register changes from 16 bits to 8 bits.

Internally, the length of the shift register is always 16 bits and the input lines A3 through A0 select which of the 16 bits reach the output.

Inputs			Output
Am	CLK	D	Q
Am	X	X	Q(Am)
Am	↑	D	Q(Am-1)

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of srl16 is

signal Q_INT: std_logic_vector(WIDTH-1 downto 0);

begin

process(C)
begin
 if (C'event and C='1') then
  Q_INT <= Q_INT(WIDTH-2 downto 0) & D;
 end if;
end process;

Q <= Q_INT(WIDTH-1);

end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C)
begin
 Q_INT <= {Q_INT[WIDTH-2:0],D};
end

always @(Q_INT)
begin
 Q <= Q_INT[WIDTH-1];
end
```

VHDL Instantiation Template

```
-- Component Declaration for SRL16 should be placed
-- after architecture statement but before begin keyword

component SRL16
   -- synthesis translate_off
   generic (
        INIT  : bit_value:= X"0001");
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         A0   : in STD_ULOGIC;
         A1   : in STD_ULOGIC;
         A2   : in STD_ULOGIC;
         A3   : in STD_ULOGIC;
         CLK  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for SRL16
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Enter attributes in this section


-- Component Instantiation for SRL16 should be placed
-- in architecture after the begin keyword

SRL16_INSTANCE_NAME : SRL16
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 A0 => user_A0,
                 A1 => user_A1,
                 A2 => user_A2,
                 A3 => user_A3,
                 CLK => user_CLK,
                 D => user_D);
```

Verilog Instantiation Template

```
SRL16 SRL16_instance_name(.Q (user_Q),
                          .A0 (user_A0),
                          .A1 (user_A1),
                          .A2 (user_A2),
                          .A3 (user_A3),
                          .CLK (user_CLK),
                          .D (user_D));

defparam SRL16_instance_name.INIT = hex_value;
```

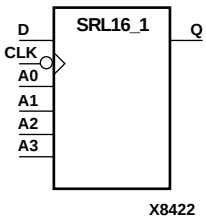
Commonly Used Constraints

BEL, U_SET, INIT

SRL16_1

16-Bit Shift Register Look-Up-Table (LUT) with Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



SRL16_1 is a shift register look up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted. See "Static Length Mode" and "Dynamic Length Mode" in the "SRL16" section.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

Inputs			Output
Am	CLK	D	Q
Am	X	X	Q(Am)
Am	↓	D	Q(Am-1)

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of srl16_1 is

signal Q_INT: std_logic_vector(WIDTH-1 downto 0);

begin

process(C)
begin
 if (C'event and C='0') then
  Q_INT <= Q_INT(WIDTH-2 downto 0) & D;
 end if;
end process;
```

```
Q <= Q_INT(WIDTH-1);

end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C)
begin
 Q_INT <= {Q_INT[WIDTH-2:0],D};
end

always @ (Q_INT)
begin
 Q <= Q_INT[WIDTH-1];
end
```

VHDL Instantiation Template

```
-- Component Declaration for SRL16_1 should be placed
-- after architecture statement but before begin keyword

component SRL16_1
   -- synthesis translate_off
   generic (
        INIT  : bit_value:= X"0001");
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         A0   : in STD_ULOGIC;
         A1   : in STD_ULOGIC;
         A2   : in STD_ULOGIC;
         A3   : in STD_ULOGIC;
         CLK  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for SRL16_1
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes in this section


-- Component Instantiation for SRL16_1 should be placed
-- in architecture after the begin keyword

SRL16_1_INSTANCE_NAME : SRL16_1
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 A0 => user_A0,
                 A1 => user_A1,
```

```
                    A2 => user_A2,
                    A3 => user_A3,
                    CLK => user_CLK,
                    D => user_D);
```

Verilog Instantiation Template

```
SRL16_1 SRL16_1_instance_name(.Q (user_Q),
                              .A0 (user_A0),
                              .A1 (user_A1),
                              .A2 (user_A2),
                              .A3 (user_A3),
                              .CLK (user_CLK),
                              .D (user_D));

defparam SRL16_1_instance_name.INIT = hex_value;
```

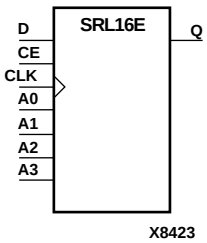
Commonly Used Constraints

BEL, U_SET, INIT

SRL16E

16-Bit Shift Register Look-Up-Table (LUT) with Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



SRL16E is a shift register look up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or dynamically adjusted. See "Static Length Mode" and "Dynamic Length Mode" in the "SRL16" section.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

When CE is High, the data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. During subsequent Low-to-High clock transitions, when CE is High, data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

When CE is Low, the register ignores clock transitions.

Inputs				Output
Am	CE	CLK	D	Q
Am	0	X	X	Q(Am)
Am	1	↑	D	Q(Am-1)

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of srl16e is

signal Q_INT: std_logic_vector(WIDTH-1 downto 0);

begin

process(C)
begin
 if (C'event and C='1') then
  if (CE='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & D;
  end if;
```

```
 end if;
end process;

Q <= Q_INT(WIDTH-1);

end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C)
begin
 if (CE)
  Q_INT <= {Q_INT[WIDTH-2:0],D};
end

always @(Q_INT)
begin
 Q <= Q_INT[WIDTH-1];
end
```

VHDL Instantiation Template

```
-- Component Declaration for SRL16E should be placed
-- after architecture statement but before begin keyword

component SRL16E
   -- synthesis translate_off
   generic (
        INIT  : bit_value:= X"0001");
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         A0   : in STD_ULOGIC;
         A1   : in STD_ULOGIC;
         A2   : in STD_ULOGIC;
         A3   : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLK  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for SRL16E
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes in this section


-- Component Instantiation for SRL16E should be placed
-- in architecture after the begin keyword

SRL16E_INSTANCE_NAME : SRL16E
   -- synthesis translate_off
   generic map(
```

```
      INIT => hex_value);
-- synthesis translate_on
   port map  (Q => user_Q,
              A0 => user_A0,
              A1 => user_A1,
              A2 => user_A2,
              A3 => user_A3,
              CE => user_CE,
              CLK => user_CLK,
              D => user_D);
```

Verilog Instantiation Template

```
SRL16E SRL16E_instance_name(.Q (user_Q),
                            .A0 (user_A0),
                            .A1 (user_A1),
                            .A2 (user_A2),
                            .A3 (user_A3),
                            .CE (user_CE),
                            .CLK (user_CLK),
                            .D (user_D));

defparam SRL16E_instance_name.INIT = hex_value;
```

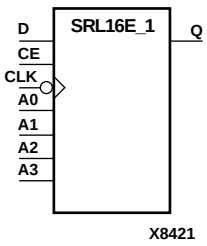
Commonly Used Constraints

BEL, U_SET, INIT

SRL16E_1

16-Bit Shift Register Look-Up-Table (LUT) with Negative-Edge Clock and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



SRL16E_1 is a shift register look up table (LUT) with clock enable (CE). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or dynamically adjusted. See "Static Length Mode" and "Dynamic Length Mode" in the "SRL16" section.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

When CE is High, the data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions, when CE is High, data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

When CE is Low, the register ignores clock transitions.

Inputs				Output
Am	CE	CLK	D	Q
Am	0	X	X	Q(Am)
Am	1	↓	D	Q(Am-1)

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of srl16e_1 is

signal Q_INT: std_logic_vector(WIDTH-1 downto 0);

begin

process(C)
begin
 if (C'event and C='0') then
  if (CE='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & D;
```

```
  end if;
 end if;
end process;

Q <= Q_INT(WIDTH-1);

end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C)
begin
 if (CE)
  Q_INT <= {Q_INT[WIDTH-2:0],D};
end

always @(Q_INT)
begin
 Q <= Q_INT[WIDTH-1];
end
```

VHDL Instantiation Template

```
-- Component Declaration for SRL16E_1 should be placed
-- after architecture statement but before begin keyword

component SRL16E_1
   -- synthesis translate_off
   generic (
        INIT  : bit_value:= X"0001");
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         A0   : in STD_ULOGIC;
         A1   : in STD_ULOGIC;
         A2   : in STD_ULOGIC;
         A3   : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLK  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for SRL16E_1
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes in this section


-- Component Instantiation for SRL16E_1 should be placed
-- in architecture after the begin keyword

SRL16E_1_INSTANCE_NAME : SRL16E_1
   -- synthesis translate_off
```

```
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 A0 => user_A0,
                 A1 => user_A1,
                 A2 => user_A2,
                 A3 => user_A3,
                 CE => user_CE,
                 CLK => user_CLK,
                 D => user_D);
```

Verilog Instantiation Template

```
SRL16E_1 SRL16E_1_instance_name (.Q (user_Q),
                                 .A0 (user_A0),
                                 .A1 (user_A1),
                                 .A2 (user_A2),
                                 .A3 (user_A3),
                                 .CE (user_CE),
                                 .CLK (user_CLK),
                                 .D (user_D));

defparam SRL16E_1_instance_name.INIT = hex_value;
```

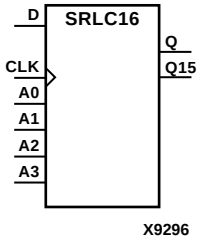
Commonly Used Constraints

BEL, U_SET, INIT

SRLC16

16-Bit Shift Register Look-Up-Table (LUT) with Carry

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



SRLC16 is a shift register look up table (LUT) with Carry. The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length, or it may be dynamically adjusted.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. During subsequent Low-to-High clock transitions data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

The Q15 output is available for the user to cascade multiple shift register LUTs to create larger shift registers.

For information about the static length mode, see "Static Length Mode" in the SRL16 section.

For information about the dynamic length mode, see "Dynamic Length Mode" in the SRL16 section.

Inputs			Output
Am	CLK	D	Q
Am	X	X	Q(Am)
Am	↑	D	Q(Am-1)

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of srlc16 is

signal Q_INT: std_logic_vector(WIDTH-1 downto 0);

begin

process(C)
begin
 if (C'event and C='1') then
  Q_INT <= Q_INT(WIDTH-2 downto 0) & D;
 end if;
end process;

Q <= Q_INT(WIDTH-1);

end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C)
begin
 Q_INT <= {Q_INT[WIDTH-2:0],D};
end

always @(Q_INT)
begin
 Q <= Q_INT[WIDTH-1];
end
```

VHDL Instantiation Template

```
-- Component Declaration for SRLC16 should be placed
-- after architecture statement but before begin keyword

component SRLC16
   -- synthesis translate_off
   generic (
        INIT  : bit_value:= X"0001");
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         Q15  : out STD_ULOGIC;
         A0   : in STD_ULOGIC;
         A1   : in STD_ULOGIC;
         A2   : in STD_ULOGIC;
         A3   : in STD_ULOGIC;
         CLK  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for SRLC16
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Enter attributes in this section


-- Component Instantiation for SRLC16 should be placed
-- in architecture after the begin keyword

SRLC16_INSTANCE_NAME : SRLC16
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 Q15 => user_Q15,
                 A0 => user_A0,
                 A1 => user_A1,
                 A2 => user_A2,
                 A3 => user_A3,
                 CLK => user_CLK,
                 D => user_D);
```

Verilog Instantiation Template

```
SRLC16 SRLC16_instance_name (.Q (user_Q),
                             .Q15 (user_Q15),
                             .A0 (user_A0),
                             .A1 (user_A1),
                             .A2 (user_A2),
                             .A3 (user_A3),
                             .CLK (user_CLK),
                             .D (user_D));

defparam SRLC16_instance_name.INIT = hex_value;
```

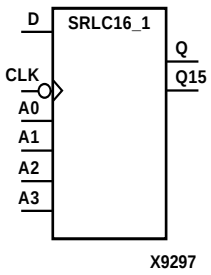
Commonly Used Constraints

INIT

SRLC16_1

16-Bit Shift Register Look-Up-Table (LUT) with Carry and Negative-Edge Clock

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



SRLC16_1 is a shift register look up table (LUT) with carry and a negative-edge clock. The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted. See "Static Length Mode" and "Dynamic Length Mode" in the "SRL16" section.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

The Q15 output is available for the user to cascade multiple shift register LUTs to create larger shift registers.

Inputs			Output	
Am	CLK	D	Q	Q15
Am	X	X	Q(Am)	No Chg
Am	↓	D	Q(Am-1)	Q14

m= 0, 1, 2, 3

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Inference Code

```
architecture Behavioral of srlc16_1 is

signal Q_INT: std_logic_vector(WIDTH-1 downto 0);

begin

process(C)
begin
 if (C'event and C='0') then
  Q_INT <= Q_INT(WIDTH-2 downto 0) & D;
 end if;
```

```
end process;

Q <= Q_INT(WIDTH-1);

end Behavioral;
```

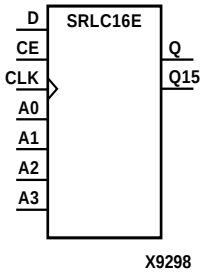
Verilog Inference Code

```
always @ (negedge C)
begin
 Q_INT <= {Q_INT[WIDTH-2:0],D};
end

always @(Q_INT)
begin
 Q <= Q_INT[WIDTH-1];
end
```

SRLC16E

16-Bit Shift Register Look-Up-Table (LUT) with Carry and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



SRLC16E is a shift register look up table (LUT) with carry and clock enable. The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. When CE is High, during subsequent Low-to-High clock transitions, data is shifted to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

The Q15 output is available for the user to cascade multiple shift register LUTs to create larger shift registers.

For information about the static length mode, see "Static Length Mode" in the SRL16 section.

For information about the dynamic length mode, see "Dynamic Length Mode" in the SRL16 section.

Inputs				Output	
Am	CLK	CE	D	Q	Q15
Am	X	0	X	Q(Am)	Q(15)
Am	X	1	X	Q(Am)	Q(15)
Am	↑	1	D	Q(Am-1)	Q15

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of srlc16e is

signal Q_INT: std_logic_vector(WIDTH-1 downto 0);

begin

process(C)
begin
 if (C'event and C='1') then
  if (CE='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & D;
  end if;
 end if;
end process;

Q <= Q_INT(WIDTH-1);

end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C)
begin
 if (CE)
  Q_INT <= {Q_INT[WIDTH-2:0],D};
end

always @ (Q_INT)
begin
 Q <= Q_INT[WIDTH-1];
end
```

VHDL Instantiation Template

```
-- Component Declaration for SRLC16E should be placed
-- after architecture statement but before begin keyword

component SRLC16E
   -- synthesis translate_off
   generic (
        INIT  : bit_value:= X"0001");
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         Q15  : out STD_ULOGIC;
         A0   : in STD_ULOGIC;
         A1   : in STD_ULOGIC;
         A2   : in STD_ULOGIC;
         A3   : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLK  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for SRLC16E
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes in this section


-- Component Instantiation for SRLC16E should be placed
-- in architecture after the begin keyword

SRLC16E_INSTANCE_NAME : SRLC16E
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 Q15 => user_Q15,
                 A0 => user_A0,
                 A1 => user_A1,
                 A2 => user_A2,
                 A3 => user_A3,
                 CE => user_CE,
                 CLK => user_CLK,
                 D => user_D);
```

Verilog Instantiation Template

```
SRLC16E SRLC16E_instance_name (.Q (user_Q),
                               .Q15 (user_Q15),
                               .A0 (user_A0),
                               .A1 (user_A1),
                               .A2 (user_A2),
                               .A3 (user_A3),
                               .CE (user_CE),
                               .CLK (user_CLK),
                               .D (user_D));

defparam SRLC16E_instance_name.INIT = hex_value;
```

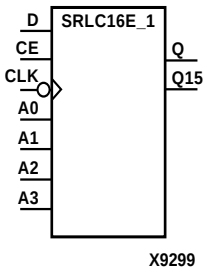
Commonly Used Constraints

INIT

SRLC16E_1

16-Bit Shift Register Look-Up-Table (LUT) with Carry, Negative-Edge Clock, and Clock Enable

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



SRLC16E_1 is a shift register look up table (LUT) with carry, clock enable, and negative-edge clock. The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted. See "SRLC16" and "Dynamic Length Mode" in the "SRL16" section.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

When CE is High, the data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions data is shifted to the next highest bit position as new data is loaded when CE is HIgh. The data appears on the Q output when the shift register length determined by the address inputs is reached.

The Q15 output is available for the user to cascade multiple shift register LUTs to create larger shift registers.

Inputs				Output	
Am	CE	CLK	D	Q	Q15
Am	0	X	X	Q(Am)	No Chg
Am	1	X	X	Q(Am)	No Chg
Am	1	↓	D	Q(Am-1)	Q14

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of srlc16e_1 is

signal Q_INT: std_logic_vector(WIDTH-1 downto 0);

begin

process(C)
begin
 if (C'event and C='0') then
```

```
  if (CE='1') then
   Q_INT <= Q_INT(WIDTH-2 downto 0) & D;
  end if;
 end if;
end process;

Q <= Q_INT(WIDTH-1);

end Behavioral;
```

Verilog Inference Code

```
always @ (negedge C)
begin
 if (CE)
  Q_INT <= {Q_INT[WIDTH-2:0],D};
end

always @(Q_INT)
begin
 Q <= Q_INT[WIDTH-1];
end
```

VHDL Instantiation Template

```
-- Component Declaration for SRLC16E_1 should be placed
-- after architecture statement but before begin keyword

component SRLC16E_1
   -- synthesis translate_off
   generic (
        INIT  : bit_value:= X"0001");
   -- synthesis translate_on
   port (Q    : out STD_ULOGIC;
         Q15  : out STD_ULOGIC;
         A0   : in STD_ULOGIC;
         A1   : in STD_ULOGIC;
         A2   : in STD_ULOGIC;
         A3   : in STD_ULOGIC;
         CE   : in STD_ULOGIC;
         CLK  : in STD_ULOGIC;
         D    : in STD_ULOGIC);
end component;

-- Component Attribute specification for SRLC16E_1
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes in this section


-- Component Instantiation for SRLC16E_1 should be placed
-- in architecture after the begin keyword
```

```
SRLC16E_1_INSTANCE_NAME : SRLC16E_1
   -- synthesis translate_off
   generic map(
        INIT => hex_value);
   -- synthesis translate_on
      port map  (Q => user_Q,
                 Q15 => user_Q15,
                 A0 => user_A0,
                 A1 => user_A1,
                 A2 => user_A2,
                 A3 => user_A3,
                 CE => user_CE,
                 CLK => user_CLK,
                 D => user_D);
```

Verilog Instantiation Template

```
SRLC16E_1 SRLC16E_1_instance_name (.Q (user_Q),
                                   .Q15 (user_Q15),
                                   .A0 (user_A0),
                                   .A1 (user_A1),
                                   .A2 (user_A2),
                                   .A3 (user_A3),
                                   .CE (user_CE),
                                   .CLK (user_CLK),
                                   .D (user_D));

defparam SRLC16E_1_instance_name.INIT = hex_value;
```

Commonly Used Constraints

INIT

STARTBUF_*architecture*

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A

STARTBUF_*architecture* is used for VHDL simulation of FPGA designs that require the use of the STARTUP block. The difference between the STARTBUF_*architecture* and the STARTUP block is that the STARTBUF_*architecture* contains output ports which may be connected to all register set/resets in the design (GSROUT) or to all I/O three-state controls (GTSOUT) so that these functions may be functionally simulated. This design element should not be used for Verilog or schematic entry. In order to use the STARTBUF_*architecture*, the desired input(s) should be connected to a top-level port in the design and the corresponding output(s) must be connected to either the three-state control signal for all inferred and instantiated output buffers in the design (GTSOUT) or all inferred or instantiated register set/resets in the design.

During simulation, the inputs to the STARTBUF_*architecture* can be toggled by the testbench in order to activate the global three-state or global set/reset signal in the design. This should be done at the beginning of the simulation to simulate the behavior of the registers and I/O during configuration. It may also be applied during simulation to simulate a reconfiguration (PROG pin high) of the device. During synthesis and implementation, this component will be treated as a STARTUP block. The connected input ports to this component should remain in the design and be connected to the correct corresponding global resource.

For more information, please consult the *Xilinx Synthesis and Simulation Design Guide.*

Truth table of inputs and outputs:

The value at port GSROUT will be always the be value at port GSRIN. The value at port GTSOUT will be always be the value at port GTSIN. CLKIN has no effect on simulation.

VHDL Instantiation Code

Following are three examples:

```
component STARTBUF_SPARTAN2
  port (GSROUT    : out std_ulogic;
        GTSOUT    : out std_ulogic;
        CLKIN     : in std_ulogic;
        GSRIN     : in std_ulogic;
        GTSIN     : in std_ulogic);
end component;

component STARTBUF_VIRTEX
  port (GTSOUT    : out std_ulogic;
        GSROUT    : out std_ulogic;
        CLKIN     : in std_ulogic;
        GSRIN     : in std_ulogic;
        GTSIN     : in std_ulogic);
end component;

component STARTBUF_VIRTEX2
```

```
  port (GSROUT    : out std_ulogic;
        GTSOUT    : out std_ulogic;
        CLKIN     : in std_ulogic;
        GSRIN     : in std_ulogic;
        GTSIN     : in std_ulogic);
end component;
```

Commonly Used Constraints

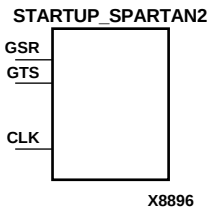
None

STARTUP_SPARTAN2

Spartan-II User Interface to Global Clock, Reset, and 3-State Controls

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive*	N/A	N/A	N/A	N/A	N/A

* Supported for Spartan-II but not for Spartan-IIE which is supported by STARTUP_VIRTEX.



The STARTUP_SPARTAN2 primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAM (RAMB4) output register in the device, depending on the initialization state (S or R) of the component.

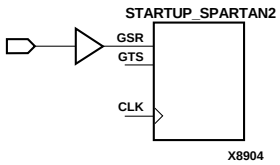
Note Block RAMB4 content, LUT RAMs, delay locked loop elements (CLKDLL, CLKDLLHF, BUFGDLL), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1) are not set/reset.

Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

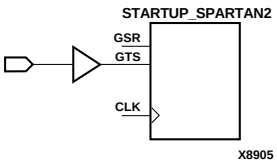
Note GTS= Global 3-State

Including the STARTUP_SPARTAN2 symbol in a design is optional. You must include the symbol under the following conditions.

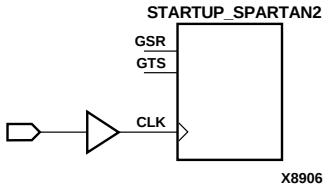
- To exert external control over global set/reset, connect the GSR pin to a top level port and an IBUF, as shown here.



- To exert external control over global 3-state, connect the GTS pin to a top level port and IBUF, as shown here.



- To synchronize startup to a user clock, connect the user clock signal to the CLK input, as shown here. Furthermore, “user clock” must be selected in the BitGen program.



You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

Usage

For HDL, this design element typically is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for STARTUP_SPARTAN2 should be placed
-- after architecture statement but before begin keyword

component STARTUP_SPARTAN2
   port (CLK  : in STD_ULOGIC;
         GSR  : in STD_ULOGIC;
         GTS  : in STD_ULOGIC);
end component;

-- Component Attribute specification for STARTUP_SPARTAN2
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes in this section


-- Component Instantiation for STARTUP_SPARTAN2 should be
   placed
-- in architecture after the begin keyword

STARTUP_SPARTAN2_INSTANCE_NAME : STARTUP_SPARTAN2
      port map  (CLK => user_CLK,
                 GSR => user_GSR,
                 GTS => user_GTS);
```

Verilog Instantiation Template

```
STARTUP_SPARTAN2 STARTUP_SPARTAN2_instance_name (.CLK (user_CLK),
                                                 .GSR (user_GSR),
                                                 .GTS (user_GTS));
```

Commonly Used Constraints

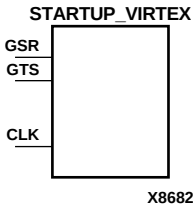
None

STARTUP_VIRTEX

Virtex and Virtex-E User Interface to Global Clock, Reset, and 3-State Controls

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive*	Primitive	N/A	N/A	N/A	N/A

* Supported for Spartan-IIE but not for Spartan-II which is supported by STARTUP_SPARTAN2.



The STARTUP_VIRTEX primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAM (RAMB4) output register in the device, depending on the initialization state (S or R) of the component. For Virtex-II and Virtex-II Pro, see "STARTUP_VIRTEX2".

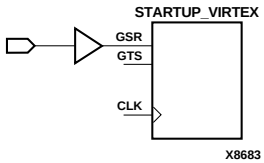
Note Block RAMB4 content, LUT RAMs, delay locked loop elements (CLKDLL, CLKDLLHF, BUFGDLL), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1) are not set/reset.

Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

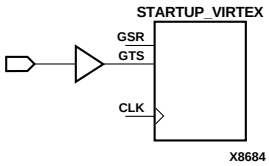
Note GTS= Global 3-State

Including the STARTUP_VIRTEX symbol in a design is optional. You must include the symbol under the following conditions.

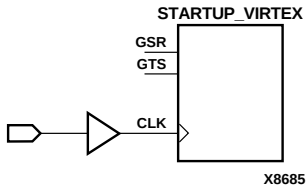
- To exert external control over global set/reset, connect the GSR pin to a top level port and an IBUF, as shown here.



- To exert external control over global 3-state, connect the GTS pin to a top level port and IBUF, as shown here.



- To synchronize startup to a user clock, connect the user clock signal to the CLK input, as shown here. Furthermore, "user clock" must be selected in the BitGen program.



You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

Usage

For HDL, this design element typically is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for STARTUP_VIRTEX should be placed
-- after architecture statement but before begin keyword

component STARTUP_VIRTEX
   port (CLK  : in STD_ULOGIC;
         GSR  : in STD_ULOGIC;
         GTS  : in STD_ULOGIC);
end component;

-- Component Attribute specification for STARTUP_VIRTEX
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes in this section


-- Component Instantiation for STARTUP_VIRTEX should be placed
-- in architecture after the begin keyword

STARTUP_VIRTEX_INSTANCE_NAME : STARTUP_VIRTEX
      port map  (CLK => user_CLK,
                 GSR => user_GSR,
                 GTS => user_GTS);
```

Verilog Instantiation Template

```
STARTUP_VIRTEX STARTUP_VIRTEX_instance_name (.CLK (user_CLK),
                                             .GSR (user_GSR),
                                             .GTS (user_GTS));
```

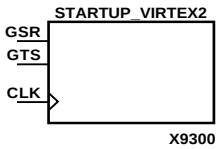
Commonly Used Constraints

None

STARTUP_VIRTEX2

Virtex-II and Virtex-II Pro User Interface to Global Clock, Reset, and 3-State Controls

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	N/A	Primitive	N/A	N/A	N/A



The STARTUP_VIRTEX2 primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAMB16 output register in the device, depending on the initialization state (INIT=1 or 0) of the component. For Virtex and Virtex-E, see "STARTUP_VIRTEX".

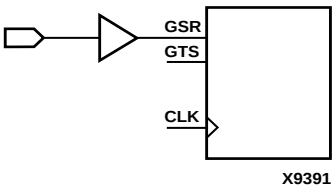
Note Block RAM content, LUT RAMs, the Digital Clock Manager (DCM), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1, SRLC16, SRLC16_1, SRLC16E, and SRLC16E_1) are not set/reset.

Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

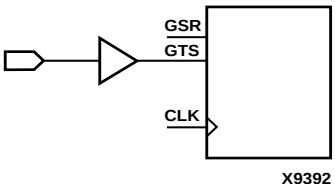
Note GTS= Global 3-State

Including the STARTUP_VIRTEX2 symbol in a design is optional. You must include the symbol under the following conditions.

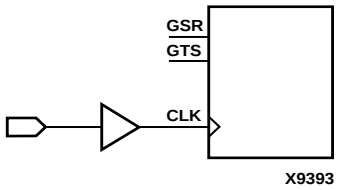
- To exert external control over global set/reset, connect the GSR pin to a top level port and an IBUF, as shown here.



- To exert external control over global 3-state, connect the GTS pin to a top level port and IBUF, as shown here.



- To synchronize startup to a user clock, connect the user clock signal to the CLK input, as shown here. Furthermore, "user clock" must be selected in the BitGen program.



You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

Usage

For HDL, this design element typically is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for STARTUP_VIRTEX2 should be placed
-- after architecture statement but before begin keyword

component STARTUP_VIRTEX2
   port (CLK  : in STD_ULOGIC;
         GSR  : in STD_ULOGIC;
         GTS  : in STD_ULOGIC);
end component;

-- Component Attribute specification for STARTUP_VIRTEX2
-- should be placed after architecture declaration but
-- before the begin keyword


-- Enter attributes in this section


-- Component Instantiation for STARTUP_VIRTEX2 should be placed
-- in architecture after the begin keyword

STARTUP_VIRTEX2_INSTANCE_NAME : STARTUP_VIRTEX2
      port map  (CLK => user_CLK,
                 GSR => user_GSR,
                 GTS => user_GTS);
```

Verilog Instantiation Template

```
STARTUP_VIRTEX2 STARTUP_VIRTEX2_instance_name (.CLK (user_CLK),
                                               .GSR (user_GSR),
                                               .GTS (user_GTS));
```

Commonly Used Constraints

None

TOC

Three-State On Configuration

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A

The TOC is a component used for VHDL simulation of FPGA designs. This component should not be used for Verilog or schematic entry. The TOC's function is to mimic the function of the internal three-state signal during the FPGA configuration process. In order to use TOC, it must be connected to the three-state signal for all inferred and instantiated output buffers in the design. During synthesis and implementation, this three-state signal will use the dedicated global three-state network and will not use local routing resources. During simulation, TOC will emit a one-shot pulse for the amount of time specified by the WIDTH generic (default is 100 ns). This one-shot pulse is intended to three-state all outputs so that at the beginning of operation, all outputs are not being driven as would happen in the real silicon during configuration of the device.

For more information, please consult the *Xilinx Synthesis and Simulation Design Guide.*

Port O will be high at simulation time 0 for the amount of time specified by the WIDTH generic attribute. After that time, it will be 0. This will not affect implementation in any way.

VHDL Instantiation Code

```
component TOC
-- synthesis translate_off
  generic (WIDTH : Time := 100 ns);
-- synthesis translate_on
  port (O : out std_ulogic := '0');
end component;
```

Commonly Used Constraints

For simulation, the WIDTH generic can be modified to change the amount of time the one-shot pulse is applied for.

There are no supported constraints for this component for implementation.

TOCBUF

Three-State On Configuration Buffer

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A

The TOCBUF is a component used for VHDL simulation of FPGA designs. It is similar to the TOC component except that it contains an input for controlling the global I/O three-state function rather than a one-shot. This component should not be used for Verilog or schematic entry. The TOCBUF's function allows user control of the function of the global I/O three-state signal as done during the FPGA configuration process. In order to use the TOCBUF, the input should be connected to a top-level port in the design and the output must be connected to the three-state control signal for all inferred and instantiated output buffers in the design.

During simulation, the input to the TOCBUF can be toggled by the testbench in order to activate the global three-state signal in the design. This should be done at the beginning of the simulation to simulate the behavior of the I/O during configuration. It may also be applied during simulation to simulate a reconfiguration (PROG pin high) of the device. During synthesis and implementation, this three-state signal uses the dedicated global three-state network and does not use local routing resources. The port connected to this component is optimized out of the design and does not use any pin resources. If you want to have the port implemented in the design, a STARTBUF_*architecture* should be used. In order to replace this port during back-end simulation, the -tp switch should be used when invoking the NGD2VER or NGD2VHDL netlister. If using the ISE GUI, use the Bring Out Global Three-state Net as a Port option in the Simulation Model Properties window.

For more information, please consult the *Xilinx Synthesis and Simulation Design Guide.*

The value at port O will be always be the value at port I (it is a buffer).

VHDL Instantiation Code

```
component TOCBUF
  port (I : in  std_ulogic;
        O : out std_ulogic);
end component;
```

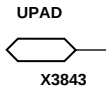
Commonly Used Constraints

None

UPAD

Connects the I/O Node of an IOB to the Internal PLD Circuit

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A

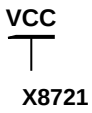


A UPAD allows the use of any unbonded IOBs in a device. It is used the same way as an IOPAD except that the signal output is not visible on any external device pins.

VCC

VCC-Connection Signal Tag

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	Primitive	Primitive	Primitive



X8721

The VCC signal tag or parameter forces a net or input function to a logic High level. A net tied to VCC cannot have any other source.

When the placement and routing software encounters a net or input function tied to VCC, it removes any logic that is disabled by the VCC signal. The VCC signal is only implemented when the disabled logic cannot be removed.

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Inference Code

```
vcc_signal <= '1';
```

Verilog Inference Code

```
assign vcc_signal = 1;
```

VHDL Instantiation Template

```
-- Component Declaration for VCC should be placed
-- after architecture statement but before begin keyword

component VCC
   port (P  : out STD_ULOGIC);
end component;

-- Component Attribute specification for VCC
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here


-- Component Instantiation for VCC should be placed
-- in architecture after the begin keyword

VCC_INSTANCE_NAME : VCC
      port map  (P => user_P);
```

Verilog Instantiation Template

```
VCC instance_name (.P (user_P));
```

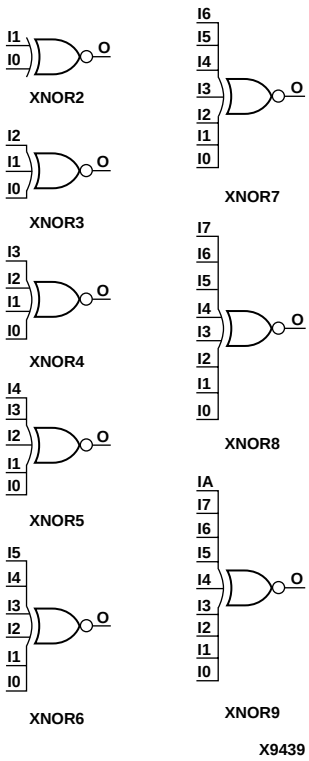
Commonly Used Constraints

None

XNOR2-9

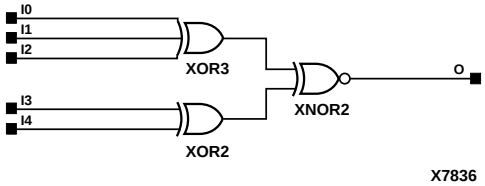
2- to 9-Input XNOR Gates with Non-Inverted Inputs

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
XNOR2, XNOR3, XNOR4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR5	Primitive	Primitive	Primitive	Macro	Macro	Macro
XNOR6, XNOR7, XNOR8, XNOR9	Macro	Macro	Macro	Macro	Macro	Macro

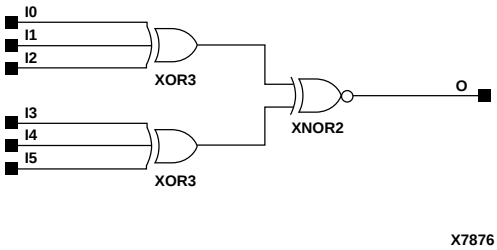


XNOR Gate Representations

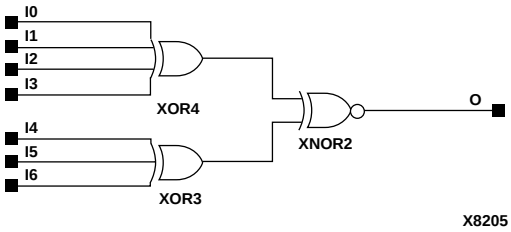
XNOR functions of up to nine inputs are available. All inputs are non-inverting. Because each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.



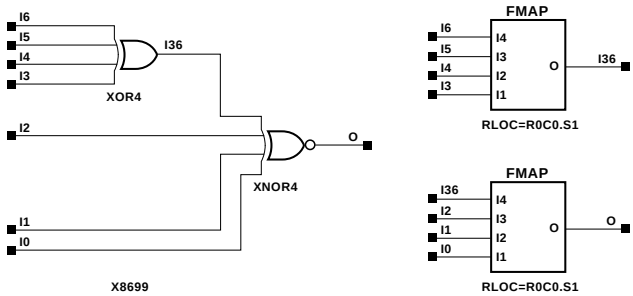
XNOR5 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



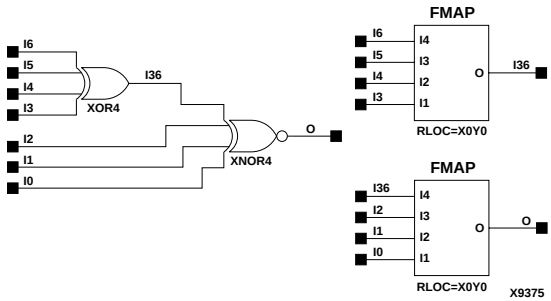
XNOR6 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



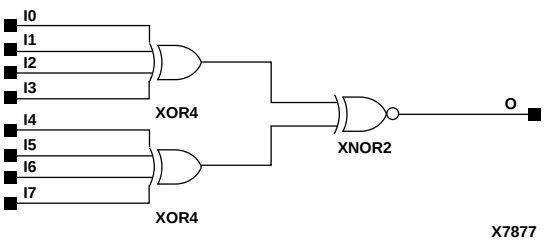
XNOR7 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



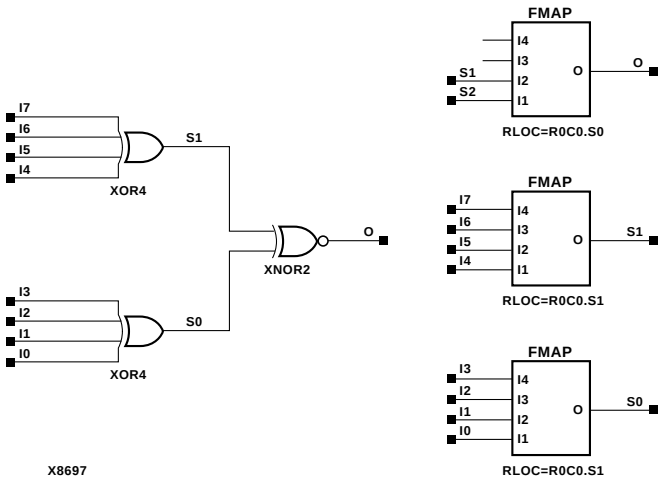
XNOR7 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



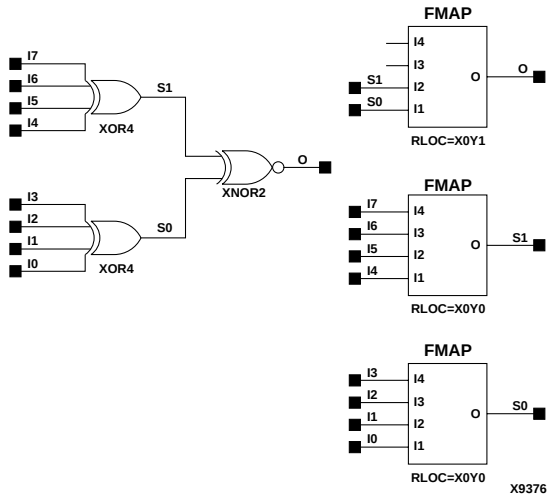
XNOR7 Implementation Virtex-II, Virtex-II Pro



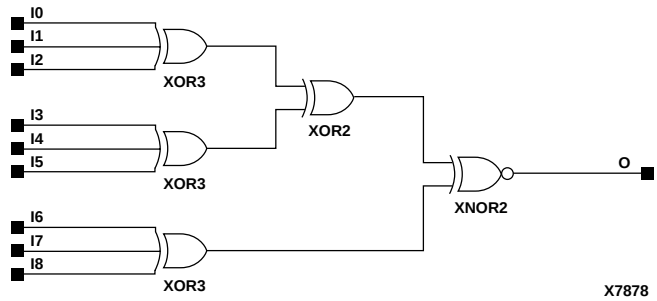
XNOR8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



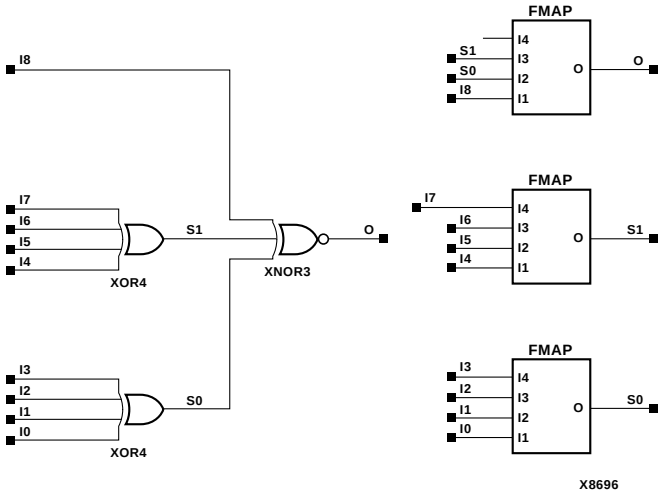
XNOR8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



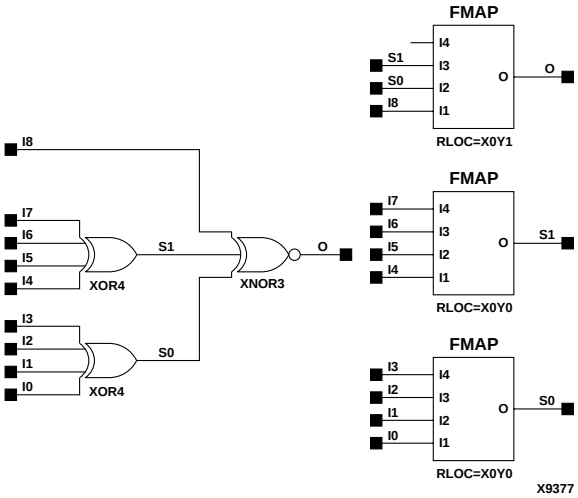
XNOR8 Implementation Virtex-II, Virtex-II Pro



XNOR9 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



XNOR9 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



XNOR9 Implementation Virtex-II, Virtex-II Pro

Usage

For HDL, these design elements can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of xnor2 is

begin

process (I0, I1)
begin
 O <= I0 xnor I1;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (I0 or I1)
begin
 O <= I0~^I1;
end
```

VHDL Instantiation Template for XNOR5

Following is the VHDL code for XNOR5. To instantiate XNOR2, remove I2, I3, and I4. To instantiate XNOR3, remove I3 and I4, For XNOR4, remove I4. XNOR2B1, and XNOR2B2 have the same code as XNOR2. XNOR3B1, 3B2, and 3B3 have the same code as XNOR3 and so forth.

```
-- Component Declaration for XNOR5 should be placed
-- after architecture statement but before begin keyword

component XNOR5
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         I2 : in STD_ULOGIC;
         I3 : in STD_ULOGIC;
         I4 : in STD_ULOGIC);
end component;

-- Component Attribute specification for XNOR5
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for XNOR5 should be placed
-- in architecture after the begin keyword

XNOR5_INSTANCE_NAME : XNOR5
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3,
                 I4 => user_I4);
```

Verilog Instantiation Template for XNOR5

```
XNOR5 XNOR5_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .I2 (user_I2),
                           .I3 (user_I3),
                           .I4 (user_I4));
```

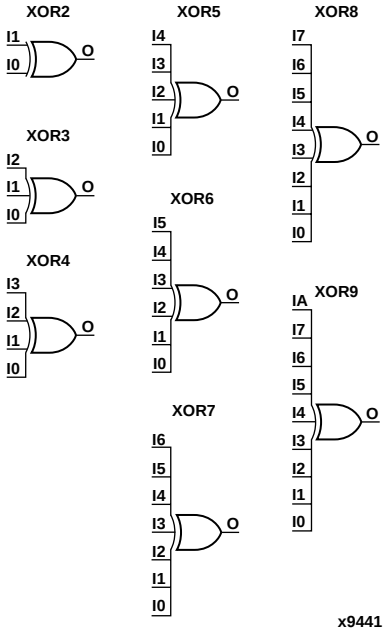
Commonly Used Constraints

None

XOR2-9

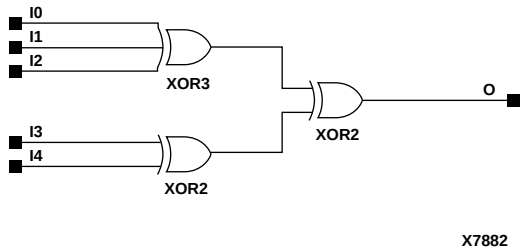
2- to 9-Input XOR Gates with Non-Inverted Inputs

Element	Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
XOR2, XOR3, XOR4	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XOR5	Primitive	Primitive	Primitive	Macro	Macro	Macro
XOR6, XOR7, XOR8, XOR9	Macro	Macro	Macro	Macro	Macro	Macro

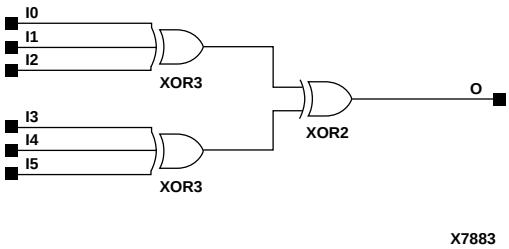


XOR Gate Representations

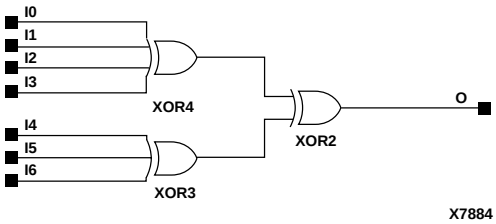
XOR functions of up to nine inputs are available. All inputs are non-inverting. Because each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.



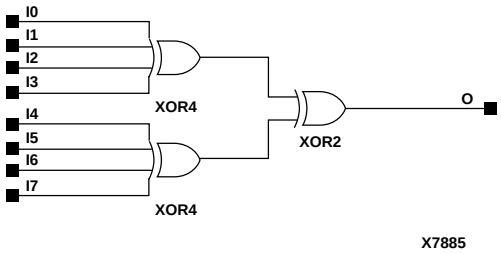
XOR5 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



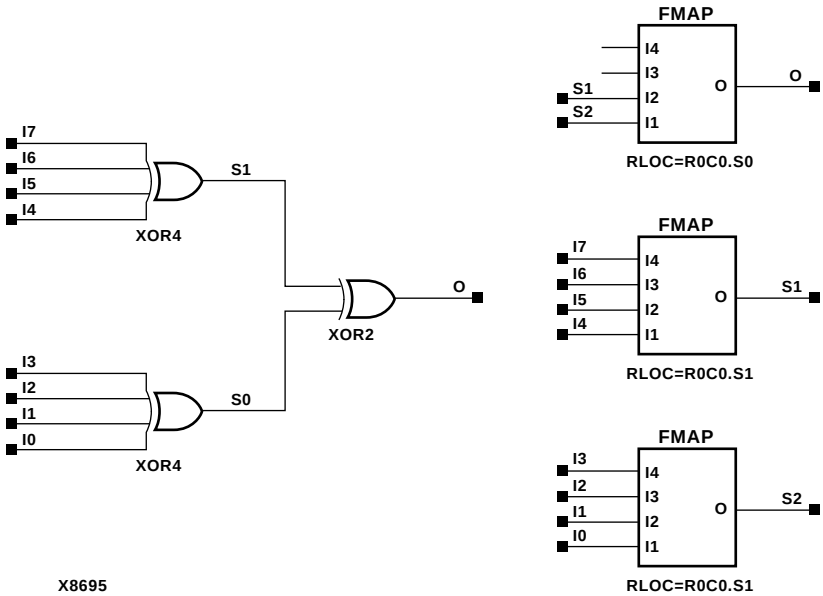
XOR6 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



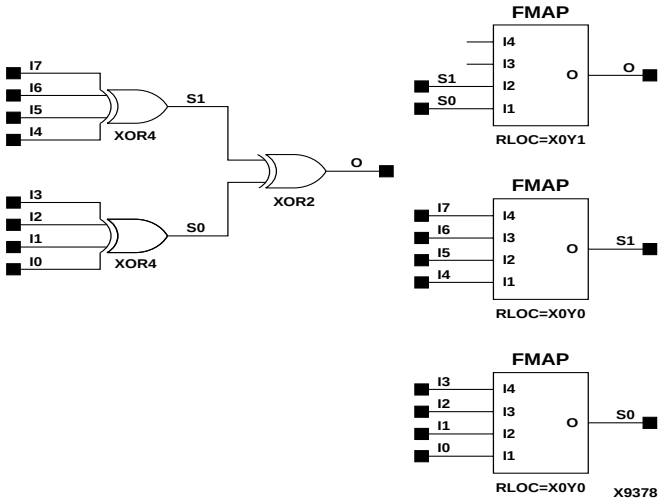
XOR7 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



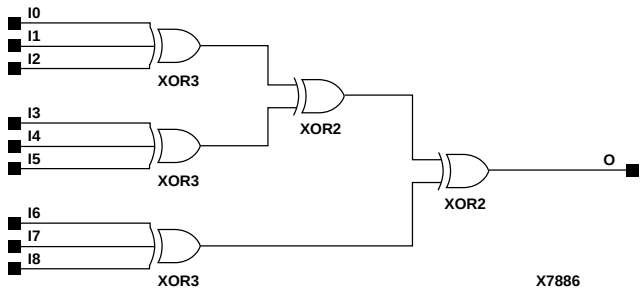
XOR8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



XOR8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



XOR8 Implementation Virtex-II, Virtex-II Pro



XOR9 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements can be inferred or instantiated.

VHDL Inference Code

```
architecture Behavioral of xor2 is

begin

process (I0, I1)
begin
 O <= I0 xor I1;
end process;

end Behavioral;
```

Verilog Inference Code

```
always @ (I0 or I1)
begin
 O <= I0^I1;
end
```

VHDL Instantiation Template for XOR5

Following is the VHDL code for XOR5. To instantiate XOR2, remove I2, I3, and I4. To instantiate XOR3, remove I3 and I4, For XOR4, remove I4. XOR2B1, and XOR2B2 have the same code as XOR2. XOR3B1, 3B2, and 3B3 have the same code as XOR3 and so forth.

```
-- Component Declaration for XOR5 should be placed
-- after architecture statement but before begin keyword

component XOR5
   port (O  : out STD_ULOGIC;
         I0 : in STD_ULOGIC;
         I1 : in STD_ULOGIC;
         I2 : in STD_ULOGIC;
         I3 : in STD_ULOGIC;
         I4 : in STD_ULOGIC);
end component;

-- Component Attribute specification for XOR5
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for XOR5 should be placed
-- in architecture after the begin keyword

XOR5_INSTANCE_NAME : XOR5
      port map  (O  => user_O,
                 I0 => user_I0,
                 I1 => user_I1,
                 I2 => user_I2,
                 I3 => user_I3,
                 I4 => user_I4);
```

Verilog Instantiation Template for XOR5

```
XOR5 XOR5_instance_name(.O (user_O),
                        .I0 (user_I0),
                        .I1 (user_I1),
                        .I2 (user_I2),
                        .I3 (user_I3),
                        .I4 (user_I4));
```

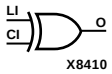
Commonly Used Constraints

None

XORCY

XOR for Carry Logic with General Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



XORCY is a special XOR with general O output used for generating faster and smaller arithmetic functions.

Its O output is a general interconnect. See also "XORCY_D" and "XORCY_L".

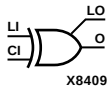
Commonly Used Constraints

BEL

XORCY_D

XOR for Carry Logic with Dual Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



XORCY_D is a special XOR used for generating faster and smaller arithmetic functions.

XORCY_D has two functionally identical outputs, O and LO. The O output is a general interconnect. The LO output is used to connect to another output within the same CLB slice.

See also "XORCY" and "XORCY_L."

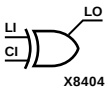
Commonly Used Constraints

BEL

XORCY_L

XOR for Carry Logic with Local Output

Spartan-II, Spartan-IIE	Virtex, Virtex-E	Virtex-II, Virtex-II Pro	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
Primitive	Primitive	Primitive	N/A	N/A	N/A



XORCY_L is a special XOR with local LO output used for generating faster and smaller arithmetic functions. The LO output is used to connect to another output within the same CLB slice.

See also "XORCY" and "XORCY_D."

Commonly Used Constraints

BEL