Script zum Softprocessor-Praktikum Teil 1: Einführung

14. November 2014

Zusammenfassung

Es ist ein minimales Rechnersystem aus einem Softprozessor, einem kleinen Daten- und Befehlsspeicher, einer Schaltereingabe, einer Leuchtdiodenausgabe, einer seriellen Schittstelle für die Kommunikation mit dem Arbeitsplatzrechner und einer Debug-Schnittstelle aus IP-Cores zu konfigurieren und in C zu programmieren.

1 Einführung

Ein einfaches Rechnersystem besteht aus Prozessor, Daten- und Befehlsspeicher sowie Schnittstellenschaltungen zur Peripherie. Hinzu können Einheiten kommen, die:

- den Rechner von bestimmten Aufgaben entlasten (Timer, DMA)
- die Softwareentwicklung vereinfachen (MMU)
- zur Fehlererkennung und Fehlerisolation (Watchdog, Hardware-Debugger, Trace-Schnittstellen)

Je nach erforderlicher Rechenleistung unterscheiden sich die Prozessoren

- in der Bitbreite für Adressen, Befehle und Daten
- in der Taktfrequenz
- im Befehlssatz und
- in der Pipeline-Tiefe.

Schnelle Prozessoren haben Cache-Speicher für kleine, häufig genutzte Adressbereiche des Daten und/oder Befehlsspeichers, damit der Prozessor nicht bei jedem Speicherzugriff mehrere Takte warten muss. Moderne Betriebssysteme benötigen eine Speicherverwaltungseinheit mit virtueller Adressierung (memory management unit, MMU), die jedem Prozess üblicherweise einen eigenen von anderen Prozessen isolierten 32-Bit-Adressraum zur Verfügung stellt und weitere Schutzmechanismen, die eine Beeinträchtigung der Abarbeitung der einzelnen Prozesse durch Fehlfunktionen in anderen Prozessen verhindern.

Ein wichtiger Kostenaspekt für Rechnersysteme ist der Aufwand für die Inbetriebnahme und Fehlersuche in der Hard- und Software (typisch weit über 50% des Entwicklungsaufwands). Die Minimalunterstützung für den Modultest eines eingebetteten Systems ist ein Programmloader und eine serielle Schnittstelle für Testausgaben. Zusätzlich sehr nützlich sind Steuermöglichkeiten für einen Schrittbetrieb, Unterbrechungspunkte, Schreib- und Lesemöglichkeiten

<allg. Bild>

Abbildung 1: Einfaches Rechnersystem

programminterner Daten im angehaltenen Zustand und eine serielle Eingabe. Zur Kontrolle von Zeitabläufen ist es sehr hilfreich, wenn außer den Ausgabesignalverläufen während des Tests auch die Zeitverläufe interner Signale zur Kontrolle aufgezeichnet werden können.

Die EDK/SDK-Entwicklungsumgebung ist ein sehr flexibles System für kombinierte Hard- und Software-Entwurf mit programmierbaren Logikschaltkreisen. Die Hardware wird aus vorentworfenen konfigurierbaren Makros für Prozessor, Speicherschnittstellen, Speicher, EA-Einheiten, ... und optional auch eigenen Schaltungsentwürfen zusammengesetzt. Dazu ist manuell oder mit unterschiedlichen graphischen Werkzeugen eine Hardware-Beschreibung (mhs-Datei) zu erstellen. Daraus wird eine Netzliste, daraus wieder eine Bitdatei für die Programmierung der Hardware-Funktion, eine Beschreibung, wohin die Programme zu laden sind (...-Datei) und eine xml-Datei für die Beschreibung der Programmschnittstellen zur Hardware generiert. Die Software-Entwicklungsumgebung (SDK) importiert im ersten Schritt die drei Datein aus dem Hardware-Entwurf. Im zweiten Schritt wird ein Board-Support-Package angelegt, in dem jedem Prozessor[1] angepasste Treiberprogramme und Standardfunktionen zugeordnet werden. Die hardware-spezifischen Adressen und Konstanten stehen z.B. in dem Header xparameter.h. Der nächste Schritt ist die Entwicklung und der Test der Applikationsprogramme.

Der Softwaretest beginnt mit dem Download der Bitdatei zur Konfiguration der Hardware. Der Douwnload des Programms in den Befehlsspeicher erfolgt davon unabhängig und kann mehrfach wiederholt werden. Für Rechnersysteme mit Debug-Modul können die Programme mit oder ohne Debugger gestartet werden. Im Debug-Modus steht eine Schrittbetrieb auf können mit oder Danach wird das Programm in den Programmspeicher geladen und gestartet oder der Debugger mitgestartet. Die Nutzung des Debuggers setzt ein Prozessor mit Debug-Schnittstelle und Testausgaben eine mit dem PC, auf dem die Software entwickelt wird, verbundene serielle Schnittstelle voraus. Programmierung, Debuggen und serielle Kommunikation mit dem Testobjekt erfolgen über den JTAG-Bus. Bei einem in die Hardware integrierten Logikanalysator (ILA) lassen sich zusätzlich zu Testzwecken interne und Ausgabesignalverläufe des in den FPGA programmierten Rechnersystems mitschreiben und anzeigen.

1.1 Der Microblaze und seine Konfiguartionsmöglichkeiten

Der Microblaze ist ein speziell für die Implementierung in FPGAs entwickelter konfigurierbarer Softprozessor. Abb. # zeigt das Blockschaltbild nach ##.

In der Minimalversion ist er ein RISC-Prozessor mit Befehlszähler, Befehlspuffer, 32-Universalregistern, einem Rechenwerk mit arithmetischen, logischen und 1-Bit-Verschiebeoperationen, einer dreistufigen Verarbeitungs-Pipeline, einem ILMB (instruction local memory bus) zum Anschluss eines Blockspeichers als Befehlsspeicher und einem DLMB (data local memory bus) zum Anschluss eines Blockspeichers (oder des zweiten Ports des Befehlsspeichers) als Datenspeicher. Der Anschluss peripherer Einheiten erfolgt wahlweise über den DPLB oder den M_AXI_DP Bus. Die Schnittstellen über dem AXI-Busse in Abb. # sind für den Anschluss eines Speicherkontrollers für einen größeren externen Hauptspeicher und die datenseitigen Schnittstellen unter dem DLMB für FIFO-gepufferte schnelle Datenverbindungen. Außer der Erweiterung um Cache-Speicher gibt es die Möglichkeit, die Pipeline-Tiefe und damit die maximale Taktfrequenz zu erhöhen, die in Hardware ausführbaren Befehle um Multiplikations-, Blockschiebe- und Divisions-

[1] Das Rechnersystem kann auch mehrere Prozessoren enthalten.

<Entwurfsfluss: mhs-Datei prüfen, Netzliste generieren, Bitdatei generierien, Export an die Softwareentwicklungsumg>

Abbildung 2: Entwurfsfluss

befehle sowie um ein Gleitkommarechenwerk zu erweitern. Des weiteren kann eine Speicherverwaltungseinheit (MMU, memory management unit) mit Übersetzungspuffern von physikalischen in virtuelle Speicheradressen, ein Cache für die Sprungzielvorhersage, eine Debug-Einheit und ein Trace-Anschluss zum Mitschreiben ausgewählter Prozessorsignale ergänzt werden.

Aufgabe:

1.1.1 Befehlssatz

Der Microblaze ist ein RISC-Prozessor mit einem pipeline-fähigen Befehlssatz von knapp 140 Befehlen. Diese sind im wesentlichen unterteilt in

- Verarbeitungsbefehle,
- Lade- und Speicherbefehle sowie
- Sprung- und Verzeigungsbefehle.

Verarbeitungsbefehle verknüpfen entweder die Inhalte zweiter Quellregister oder eines Quellregisters und einer Konstanten und schreiben das Ergebnis in das Zeilregister. Einige Beispiele:

Assembler	0-5	6-10	11-15	16-20	21-31	Funktion
Add Rd,Ra,Rb	000000	Rd	Ra	Rb	00000000000	Rd:=Rb+Ra
Addi Rd, Ra, Imm	001000	Rd	Ra	Imm		Rs:=Ra+Imm
And Rd,Ra,Rb	100001	Rd	Ra	Rb	00000000000	Rd:=Rb and Ra
Cmp Rd,Ra,Rb	00010	Rd	Ra	Rb	00000000000	if (Rb>=Ra) Rd:=1 else Rd:=0
Mul Rd,Ra,Rb	01000	Rd	Ra	Rb	00000000001	Rd:=Rb*Ra
Mulh Rd,Ra,Rb	01000	Rd	Ra	Rb	00000000011	Rd:=(Rb*Ra)>>32 (sighned)

(Imm – imidiate, 16-Bit-Konstante)

Lade- und Speicherbefehle berechnen die Adresse entweder als Summe aus zwei Registerinhalten (Ra+Rb) oder eines Registerinhaltes und einer Konstanten (Ra+Imm). Das Register Rd ist entweder Quelle oder Ziel des Datentransfers. Es gibt die Möglichkeit Worte, Halbworte und einzelne Bytes zu übertragen und die Ergebnisse mit null oder vorzeichenerweitert zu speichern:

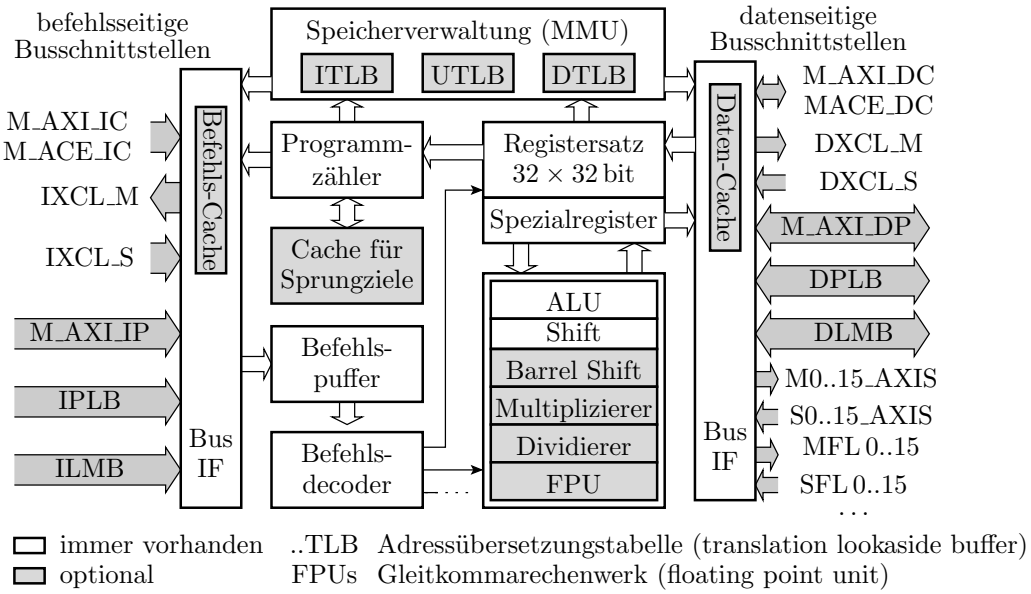


Abbildung 3: Entwurfsfluss

Assembler	0-5	6-10	11-15	16-20	21-31	Funktion
LW Rd,Ra,Rb	110010	Rd	Ra	Rb	00000000000	Adr=Ra+Rb; Rd:=*Adr
LWI Rd, Ra, Imm	111010	Rd	Ra	Imm		Adr=Ra+Imm; Rd:=*Adr
LBU Rd,Ra,Rb	110000	Rd	Ra	Rb	00000000000	Adr=Ra+Rb; Rd[24:31]:=*Adr[0:

Sprung- und Verzweigungsbefehle sind Operationen mit dem Befehlszähle (PC) als Zuweisungsziel. Die Sprungzieladresse kann eine im Befehl codierte Konstante, ein Registerinhalt, die Summe aus der aktuellen Befehlsadresse und einer Konstanten oder eines Registerinhalts sein. Zusätzlich kann der Wert eines Registers als Sprungbedingung ausgewertet oder die Folgeadresse als Rückkehradresse in einem Register gespeichert werden. Beispiele:

Assembler	0-5	6-10	11-15	16-20	21-31	Funktion
BRA Rb	100110	00000	00000	Rb	00000000000	PC:=RB
BR Rb	000000	00000	00000	Rb		PC:=PC+RB
BRLD Rd,Rb	000000	Rd	10100	Rb	00000000000	PC:=PC+RB; Rd:=PC
RTSD	001101	10000	Ra	Imm		PC:=Ra+Imm
BEQ Ra, Rb	100111	00000	Ra	Rb	00000000000	if (Ra=0) PC:=PC+Rb
BEQI Ra, Imm	101111	00000	Ra	Imm		if (Ra=0) PC:=PC+Imm

Die hardware-nahe Software-Entwicklung erfolgt heute überwiegend in C. Dabei ist es möglich, zur Programmoptimierung z.B. Schleifenkörper in Assembler zu beschreiben [http://www.xilinx.co

```
int main() {
while(1)
for (i=128;i>0;i--){
 microblaze_nbwrite_datafsl(i, 0);
 j++;
}
//dieselbe Funktion mit Assemblerbefehlen
asm("lwi r3, r0, 292"); // load word immediate
asm("addik r3, r3, 1"); // immediate add w/ keep carry
asm("swi r3, r0, 292"); // store word immediate } }
```

Im nachfolgenden Beispiel werden die Werte von a und b in die Quellregister des Additionsbefehls übergeben und die Summe aus dem Zielregister anschließend in die Variable a kopiert:

```
int a,b,c = 0;
asm ("add %0,%1,%2": "=r" (a): "r" (b), "r" (c)) ;
```

Aufgabe 1.1: B

efehlsbeschreibungen

Welche Funktion haben die nachfolgenden Assemblerbefehle?

1.1.2 Disassemblieren und Trace

Maschinenprogrammen benötigt man heute hauptsächlich noch für den hardware-nahen Test und die Fehlersuche. Die Testprogramme werden zwar auch hier in der Regel in einer Hochsprache geschrieben, aber die Abarbeitung ist in Maschinenschritten zu untersuchen. Das Dissasemblierten eines ladbaren Programms HalloWelt.elf kann in unserem System z.B. auf der Kommandozeile (XMD-Console) erfolgen mit:

```
mb-objdump -d HalloWelt.elf > HalloWelt.asm
```

Beispiel sei ein einfaches C-Programm, dass in einer Schleife in jedem Durchlauf den Wert der Variablen »dat« um eins erhöht an die Adresse 0x40000000 ausgibt:

```
void main(){
    u8 dat;
    for (dat=0;dat!=0xFF;dat++){
        XIo_Out8(0x40000000, dat);
    }
}
```

Für das unoptimierte (mit Compilerdirektive »O0« übersetzte Programm stehen ab der Startadresse von main() folgende Befehlsfolge im Befehlsspeicher::

```
Adresse   Befehlswort  disassemblierter Befehl
0x5d8: 3021ffd8  addik  r1, r1, -40
0x5dc: f9e10000  swi r15, r1, 0
0x5e0: fa610024  swi r19, r1, 36
0x5e4: 12610000  addk   r19, r1, r0
0x5e8: f013001c  sbi r0, r19, 28
0x5ec: b8000020  bri 32  // Sprungziel 0x60c
0x5f0: b0004000  imm 16384
0x5f4: 30600000  addik  r3, r0, 0
0x5f8: e093001c  lbui   r4, r19, 28
0x5fc: f0830000  sbi r4, r3, 0
0x600: e073001c  lbui   r3, r19, 28
0x604: 30630001  addik  r3, r3, 1
0x608: f073001c  sbi r3, r19, 28
0x60c: e073001c  lbui   r3, r19, 28
0x610: a86300ff  xori   r3, r3, 255
0x614: bc23ffdc  bnei   r3, -36  // Sprungziel 0x5f0
```

Um ein von einem einem Compiler erzeugten Maschinenprogramme zu verstehen, ist es hilfrich zu wissen, wie der Compiler die Register vergibt. Die Regeln hierfür sind zu finden in [MicroBlaze Processor Reference Guide, S. 98]. Für die im Besipielprogramm genutzten Register gilt:

r0: hat immer den Wert null

r1: Stackpointer

r3, r4: temporäre Register, die bei Unterprogrammaufrufen für Rückgabewerte genutzt werden

r15: Rückkehradresse bei Unterprogrammaufrufen

r19: bei Nutzung in Unterprogrammen auf dem Stack zu sicherndes Register.

Mit diesem Wissen und der Befehlsbeschreibung [..., S.#] erschließt sich folgende Funktionsweise für die Befehle vor der Schleife:

Adesse	Befehl	Funktion
0x5d8	addik r1, r1, -40	Verringerung des Stackpointers (SP) um 40 Byte
0x5dc	swi r15, r1, 0	Speichern der Rückkehradresse auf der Stackadresse SP+0
0x5e0	swi r19, r1, 36	Sichern des Register r19 auf der Stackadresse SP+36
0x5e4	addk r19, r1, r0	Initialisieren von r19 als Frame-Pointer (FP) für lokale Variablen mit der
0x5e8	sbi r0, r19, 28	Initialisiere die lokale Variable »dat« auf der Stackadresse FP+28 mit d
0x5ec	bri 32	Springe zum Schleifeneintrittspunkt Adresse 0x60c

Der Schleifenkörper umfasst folgende Befehlsfolge:

Adesse	Befehl	Funktion
XIo_Out(0x40000000, dat);		
0x5f0	imm 16384	Speichere Direktwerts 0x4000 für den Folgebefehl
0x5f4	addik r3, r0, 0	Lade r3 mit 0x40000000
0x5f8	lbui r4, r19, 28	lade »dat« (Adresse FP+28) in r4
0x5fc	sbi r4, r3, 0	Speicher den Wert von r4 auf Adresse 0x40000000
dat++;		
0x600	lbui r3, r19, 28	Lade nochmal »dat« in r3
0x604	addik r3, r3, 1	Erhöhe r3 (data) um eins
0x608	sbi r3, r19, 28	Speichere r3 in »dat«
Springe zurück, wenn dat≠0xFF ist		
0x60c	lbui r3, r19, 28	Lade nochmal »dat« in r3
0x610	xori r3, r3, 255	Invertiere r3 (data)
0x604	bnei r3, -36	wenn r3≠0 Sprung zu Adresse 0x5f0

Offensichtlich lässt sich die Aufgabe auch mit weniger Befehlern lösen, was der Compiler auch einer Erhöhung der Compileroptimierung auf »O1« oder höher tut. Die hier verwendetet Compileroptimierung »O0« dient hauptsächlich zum debuggen, weil bei ihr jede Operation in der Hochsprache vom Operanden aus dem Speicher Laden bis zur Ergebnisspeicherung komplett abgearbeitet wird, bevor die Abarbeitung der nächsten Hochsprachenoperation beginnt., bei der die Operation der Hochsprache nacheinander durch Maschinenbefehlsfolgen nachgebildet werden, wird nur für das Debuggen im Schrittbetrieb oder mit Unterbrechungspunkten gebraucht.

Zur Untersucheung der schrittweisen Pragrammabarbeitung in der Hardware. Wird ein Trace aufgezeichnet. Das ist eine Mitschrift der interessierenden Signalverläufe. Dazu ist ein Logikanalysator an die aufzuzeichnenden Signal anzuschließen, ein Abtasttakt festzulegen und ein Triggerbedingung einzustellen. Die Triggerbedingung legt den Aufzeichnungsbeginn und der Abtatstakt die Aufzeichungszeitpunkte fest. Unserm System hat integrierte Logikanalysatoren (ILA) als Bausteine. Diese können zu Debug-Zwecken mit in das System integriert, an alle zugänglichen internen Signale angeschlossen und über die Programmierschnittstelle mit dem PC gesteuert werden. Abb. # zeigt einen mit einem solchen integrierten Logikanalysator aufgezeichneten Trace für das Programm in Abb. #. Gezeichnet sind die Signalverläufe an den Trace-Ausgängen des Prozessors

- Trace_Valid_Instruction (wenn eins, sind der Befehlszähler- (PC) der Instructionswert gültig,
- Trace_MEM_PipeRun (wenn eins, arbeite die Speicher-Pipeline)
- Trace_Instruction (Befehlswort am Trace-Ausgang),
- Trace_PC (Befehlsadresse des angezeigten Befehlsworts).

und die Ausgabewerte für die Leds GPIO_IO_O. In Abb. # a ist gut zu erkennen, das der Ausgabewert zyklisch hochzählt. Während die Ausgabe sich ändert, wird der Befehl auf der Adresse 0x5F8 am Trace-Port ausgegeben. Das ist einer vor dem Ausgabebefehl. Am Trace-Ausgang wird offenbar immer der zuletzt abgeschlossene Befehl, und keiner von denen, die gerade bearbeitet werden, angezeigt. An dem Signal »Trace_Valid_Instruction« ist zu erkennen, dass im Mittel nur etwa in der Hälfte der Takte neue Befehle fertig gestellt werden und an »Trace_MEM_PipeRun«, dass die Speicher-Piperline bei der Ausgabe des Zählwertes an die Led's, mehrere Wartetakte einfügt. Abb. #b zeigt einen vergrößereten Ausschnitt, in dem

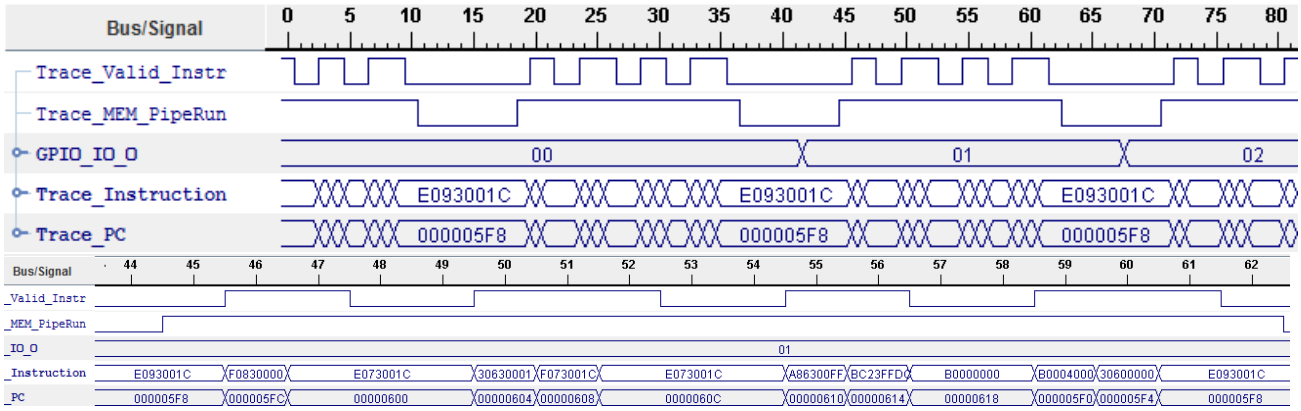


Abbildung 4: Beispiel für einen Trace

auch die Befehls- und Befehlszählerwerte der ohne oder nur mit einem zusätzlichen Wartetakt abgeschlossenen Befehle zu erkennen sind.

Aufgabe 1.2: T

race erzeugen

Erzeugen Sie mit den Hardware-Datein ### und dem

1.2 Busse und periphere Bausteine

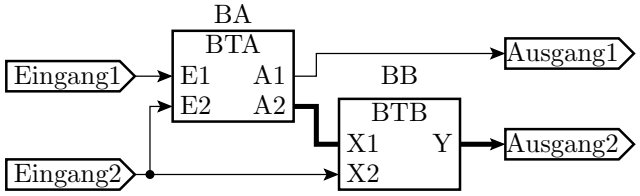
Außer dem Prozessor und integrierten Logikanalysatoren stellt Entwicklungssystem eine Vielzahl weitere vorentworfener Rechnerbausteine als Soft-Cores bereit:

- Resetschaltung, Taktversorgung, Interrupt-Controller,
- unterschiedliche Busssysteme und Busbrücken,
- parallele Schnittstellen,
- Controller für langsame seriele Schnittstellen wie UART, SPI und IIC,
- schnelle Schnittstellen wie CAN, Ethernet und USB,
- DMA-Controller, Timer, Watchdog,
- Schnittstellen-Controller für externe Speicher (SRAM, Flash, DRAM, ...),
- Gleitkommarechenwerke bis hin zu Spezialrechenwerken für die Signal- und Bildverarbeitung.

Die Cores haben unterschiedliche Konfigurationsparameter und werden über vordefinierte Busse, Bitleitungen und Bitvektorleitungen miteinander verbunden. Darüber hinaus können auch eigene Bausteine, z.B. zur Ansteuerung von Sensoren und Motoren entworfen und eingebunden werden.

1.3 Beschreibung des Rechnersystem

Die Beschreibung der zu entwerfenden Rechnersysteme in der mhs-Datei ist eine bausteinorientierten Netzliste. Abb. # zeigt eine Beispielschaltung aus zwei Bausteinen und ihre Beschreibung. Die externen Anschlüsse werden als Ports vereinbart. Den Ports werden die angeschlossenen Leitungen, die Signalflussrichtung und bei Takt und Rücksetzsignalen auch der Signaltyp und weitere Attribute zugeordnet. Jeder Bausteintyp hat Parameter und Ports. Diesen werden Werte bzw. Leitungen zugeordnet. Der Instanzname ist dabei ein Bausteinparameter. Beitere Parameter sind Bitbreiten, Vorhandensein bestimmtzer Merkmale, ... Der Typ der Verbindungen (Bit, Bitvector oder komplexer Bus) wird durch den Typ der Anschlüsse festgelegt.




```
PORT Eingang1 = ltg1, DIR = I; SIGIS = RST, RST_POLARITY = 1 # Rücksetzeingang
PORT Eingang2 = ltg1, DIR = I                                # normaler Eingang
PORT Ausgang1 = ltg1, DIR = O                                # Ausgabebit
PORT Ausgang2 = ltg1, DIR = O, VEC =[0:7]                    # Ausgabebyte

BEGIN BTA                    # Baustein vom Typ BTA
 PARAMETER INSTANCE = B1     # mit dem Instanznamen B1
 PARAMETER ...
 PORT E1 =
 PORT E2 = ...
 PORT A1 = ..
 PORT A2 =
END

BEGIN BTB                    # Baustein vom Typ BTB
 PARAMETER INSTANCE = B2     # mit dem Instanznamen B2
 PARAMETER ...
 PORT X1 =
 PORT X2 = ...
 PORT Y1 = ..
 PORT Y2 =
END
```

Außer PORT für einzelnen Bits oder Bitvektoren gibt es auch die Anschlussart BUS_INTERFACE für komplexe Busstrukturen. Eine Busstruktur kann zusätzlich eine instanzartige Beschreibung haben zum Anschluss z.B. Takt- und Rückssetzsignalen und zur Zuordnung von Parametern. Den externen Anschlüssen sind in der ucf-Datei Schaltkreis-Pins zuzuordnen.

Ein minimales Rechnersystem muss zusätzlich zum Prozesser, mindestens einen Daten -und Befehlsspeicher, eine Initialisierung- (Reset-) Schaltung und eine Ausgabe z.B. auf Leuchtdioden haben. Die einfachste Befehls- und Datenspeicher ist ein interener Blockspeicher, der über je einen lokalen Speicherbus (LMB) und Speicher-Controller für Befehle und Daten an den Prozessor anzuschließen ist. Als Takt genügt der 100 MHz Eingabetakt auf der Baugruppe (Abb. #). Für den Test und die Inbetriebnahme von Programmen ist des weiteren das Debug-Controller

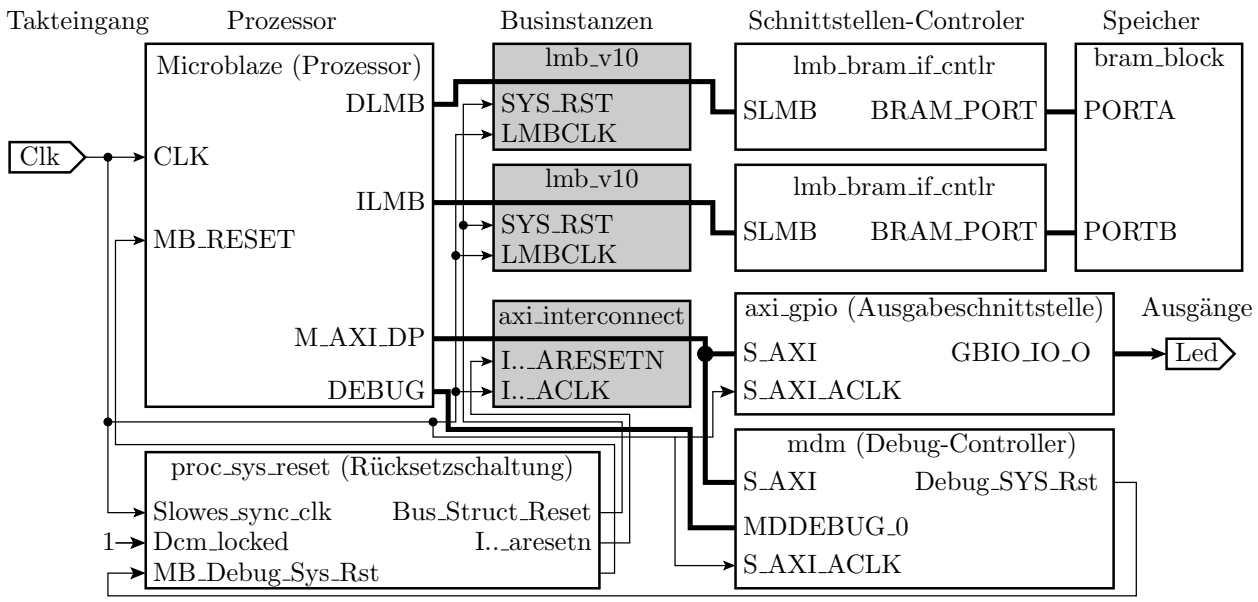


Abbildung 5: Minimalsystem

(mdm) unentbehrlich, über den der Schrittbetrieb, Unterbrechnungspunkte etc. gesteuert werden..

Die nachfolgenden mhs-Einträge beziehen sich den Rechner im Bild. Der Takteingang und der LED-Ausgang werden wie folgt beschrieben:

```
PORT CLK = CLK, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
PORT Led = Led, DIR = O, VEC = [7:0]
```

Die Zuordnung der Gehäuseanschlüsse erfolgt in der ucf-Datei und sind in der Datei # zu finden:

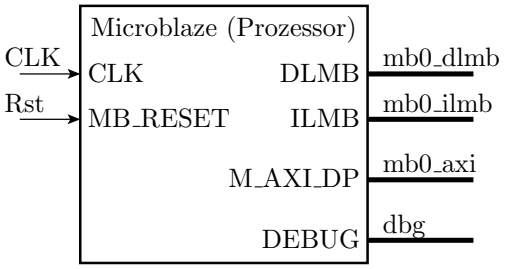
```
NET "Clk" LOC = "V10" | IOSTANDARD = "LVCMOS33";
NET "CLK" PERIOD = 10ns;
NET Led[0] LOC = "U16"  |  IOSTANDARD = "LVCMOS33";
NET Led[1] LOC = "V16"  |  IOSTANDARD = "LVCMOS33";
NET Led[2] LOC = "U15"  |  IOSTANDARD = "LVCMOS33";
NET Led[3] LOC = "V15"  |  IOSTANDARD = "LVCMOS33";
NET Led[4] LOC = "M11"  |  IOSTANDARD = "LVCMOS33";
NET Led[5] LOC = "N11"  |  IOSTANDARD = "LVCMOS33";
NET Led[6] LOC = "R11"  |  IOSTANDARD = "LVCMOS33";
NET Led[7] LOC = "T11"  |  IOSTANDARD = "LVCMOS33";
```

Die Rücksetzschaltung benötigt den (langsamsten) Takt und eine »1« am Eingang »Dcm_locked« (Takt gültig) sowie das Rücksetzsignal vom Debugger als Eingaben und liefert die Rücksetzsignale für den Prozessor und seine Busschnittstellen.

```
BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_0
 PARAMETER HW_VER = 3.00.a
 PORT Dcm_locked = net_vcc            # solange null, bleibt das System rückgesetzt
 PORT Bus_Struct_Reset = Bus_Reset    # Rücksetzsignal für Prozessor und die LMB's
 PORT Interconnect_aresetn = Iarst    # negiertes Rücksetzsignal für den AXI-Bus
 PORT Slowest_sync_clk = CLK          # (langsamster) Systemtakt, hier der 100 HHz Takt
 PORT MB_Debug_Sys_Rst = Debug_Rst    # Rücksetzsignal vom Debug-Controller
END
```

Der im Beispiel konfigurierte Prozessor hat ein Zusatzrechenwerk für Blockverschiebeoperationen, kein Gleitkommarechenwerk, Debug-Interface aktiviert, weder Befehls noch Daten-Cache, je einen lokalen Speicherbus für Befehle und Daten, einen AXI-Bus, eine Verbindung zum Debug-Controller, ein Rücksetzeingang und einen Takteingang:

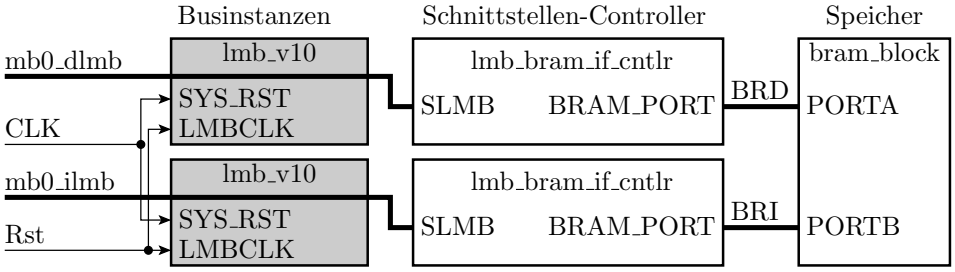
```
BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER HW_VER = 8.50.b
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 0
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_USE_ICACHE = 0
 PARAMETER C_USE_DCACHE = 0
 BUS_INTERFACE ILMB = mb0_ilmb
 BUS_INTERFACE DLMB = mb0_dlmb
 BUS_INTERFACE M_AXI_DP = mb0_axi
 BUS_INTERFACE DEBUG = mb0_debug
 PORT MB_RESET = Debug_Rst
 PORT CLK = CLK
END
```



An die lokalen Speicherbusse sind über Speicher-Controller der Blockspeicher für die Adressen und Daten angeschlossen. Die Adressbereiche und damit auch die Speichergröße sind Parameter der Speicher-Controller. An den Businstanzen sind Takt- und Rücksetzsignal gesondert anzuschließen.

```
BEGIN lmb_v10
 PARAMETER INSTANCE = mb0_ilmb
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = Bus_Reset
 PORT LMB_CLK = CLK
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = mb0_ibram_ctrl
 PARAMETER HW_VER = 3.10.c
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001fff
 BUS_INTERFACE SLMB = mb0_ilmb
 BUS_INTERFACE BRAM_PORT = BRAM_IPort
END

BEGIN lmb_v10
 PARAMETER INSTANCE = mb0_dlmb
 PARAMETER HW_VER = 2.00.b
```

```
 PORT SYS_RST = Bus_Reset
 PORT LMB_CLK = CLK
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = mb0_dbram_ctrl
 PARAMETER HW_VER = 3.10.c
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001fff
 BUS_INTERFACE SLMB = mb0_dlmb
 BUS_INTERFACE BRAM_PORT = BRAM_DPort
END

BEGIN bram_block
 PARAMETER INSTANCE = mb0_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = BRAM_IPort
 BUS_INTERFACE PORTB = BRAM_DPort
END
```

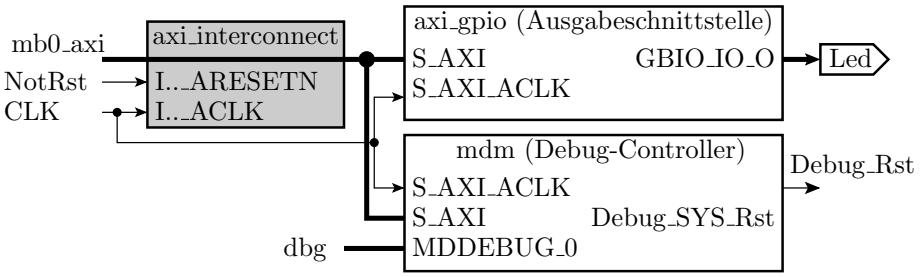


Die Ausgabe- und die Debug-Schnittstelle sind über den AXI-Bus mit dem Prozessor verbunden, an den gleichfalls ein Takt und ein (invertiertes) Rücksetzsignal anzuschließen ist. Jeder peripheren Einheit ist ein Adressbereich für seine Ein- und Ausgaberegister zugeordnet. Im Beispiel sind das 64k-Teiladressräume, obwohl die die Einheiten viel weniger Register haben. Der Debug-Controller ist zusätzlich über den Debug-Bus mit dem Prozessor verbunden und kann über seinen Rücksetzausgang das System rücksetzen. Der Parameter »C_USE_UART = 1« bewirkt, dass in den Debug-Controller zusätzlich eine über den JTAG-Bus ansteuerbare UART für die Standardein- und Ausgabe »stdin« und »stdout« eingebunden wird[2].

```
BEGIN axi_interconnect
 PARAMETER INSTANCE = mb0_axi
 PARAMETER HW_VER = 1.06.a
 PARAMETER C_INTERCONNECT_CONNECTIVITY_MODE = 0
 PORT INTERCONNECT_ARESETN = NotRst
 PORT INTERCONNECT_ACLK = CLK
END

BEGIN mdm
 PARAMETER INSTANCE = debug_module
 PARAMETER HW_VER = 2.10.a
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_UART = 1
 PARAMETER C_BASEADDR = 0x41400000
 PARAMETER C_HIGHADDR = 0x4140ffff
 BUS_INTERFACE MBDEBUG_0 = dbg
```

[2]Erforderlich z.B. für die Ausführung der HalloWelt-Programme, die in SDK automatisch erzeugbar sind.

```
 BUS_INTERFACE S_AXI = mb0_axi
 PORT Debug_SYS_Rst = Debug_Rst
 PORT S_AXI_ACLK = CLK
END

BEGIN axi_gpio
 PARAMETER INSTANCE = axi_gpio_0
 PARAMETER HW_VER = 1.01.b
 PARAMETER C_BASEADDR = 0x40000000
 PARAMETER C_HIGHADDR = 0x4000ffff
 PARAMETER C_GPIO_WIDTH = 8
 BUS_INTERFACE S_AXI = mb0_axi
 PORT S_AXI_ACLK = CLK
 PORT GPIO_IO_O = Led
END
```



Die komplette mhs-Datei und die ucf-Datei sind unter # zu finden. Die detaillierten Entwurfsschritte für das beschrieben Minimalsystem mit den vorgegebenen Datein oder dem Wizzard sind in # beschrieben. Bei der Erzeugung der Schaltungsbeschreibung mit dem Wizzard werden andere Bezeichner für die Signale und Instanzen vergeben, die Instanzbeschreibungen enthalten ungenutzte Signale und es wird eine zusätzliche Taktversorgungsschaltung eingebunden, die keine Funktion hat.

Aufgabe 1.3: E

rzeugung eines funktionsgleichen Minimalsystems

Erzeugen Sie das im Abschnitt beschriebene System mit EDK-Wizzards im System.

Vergleichen Sie das mhs-File aus ihrem Entwurf mit dem der Vorgabedatei #. Beide Beschreibungen sollen dieselben Beisteinetypen, Verbindungen und Konfigurationsparameter besitzen. Die automatisch vergebenen Bezeichner der Bausteine und Leitungen sowie die Beschreibungsreihenfolgen werden sich unterscheiden. Konfigurationsparameter mit Standardwerten können, aber müssen nicht in der Beschreibung stehen.

Aufgabe 1.4: M

inimales Rechnersystem mit 7-Segmentausgabe

Ändern Sie den (vorgegebenen) Beispielentwurf so ab, das die Ausgabeschnittstelle die 7-Segmentanzeige ansteuert. Empfohlener Lösungsweg ist der Import des vorgegebenen mhs-Filesan

1.4 Einbau von integrierten Logikanalysatoren

Das System erlaubt den Einbau mehrerer integrierter Logikanalysatoren

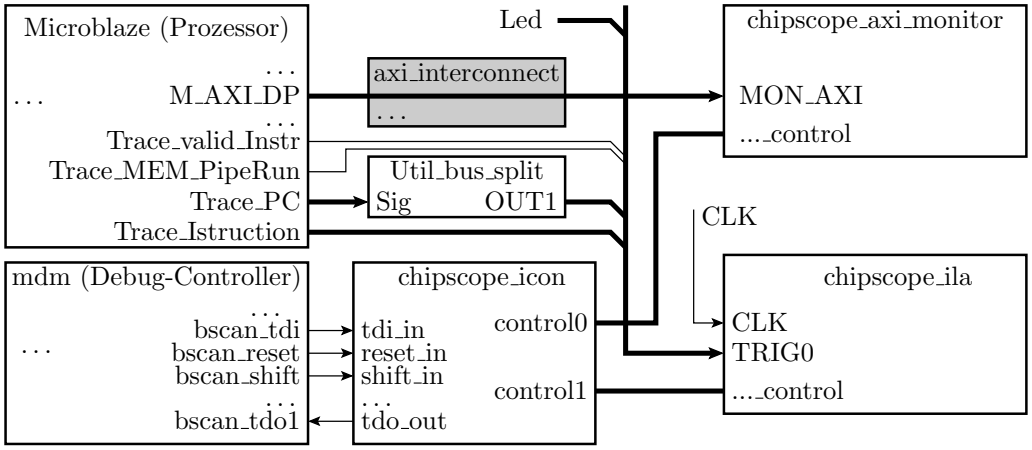


Abbildung 6: Anschluss zweier integrierter Logianalysatoren

Das Rechnersystem aus

Mit Import mhs; nach Import Fehlermledung die sich auf Virtex bezieht, geht weg, bei Wiederholung »Run DRC's«

mdm, BSCAN location auf ICON stellen

Export to SDK bringt beim ersten Versuch eine Fehlermeldung »Java?«. zweites mal ohne Neuübersetzung exportieren

Dieser Abschnitt beschreibt das Rechnersystem Trace vergangener Abschnitt.

- Schaltereingabe
- LED-Ausgabe
- MDB mit JTAG-UART
- und Logikanalysator für die Trace-Schnittstelle

<Bild mit den Verbindungen der Komponenten und der UCF-Einträge>

Am Microblaze sind zusätzliche Signale an die zu beobachtenden Trace-Ausgänge anzuschließen. Die Reduktion des der Befehlsworte auf die niederen 16 Bit verlangt einen Baustein zum Aufspalten des Busses:

```
BEGIN microblaze
 ...
 PORT Trace_Valid_Instr = ValInstr
 PORT Trace_MEM_PipeRun = MEMRun
 PORT Trace_PC = PC32
 PORT Trace_Instruction = Instr
END
BEGIN util_bus_split
 PARAMETER INSTANCE = util_bus_split_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SIZE_IN = 32
 PARAMETER C_SPLIT = 16
 PORT Sig = PC32
 PORT OUT1 = PC16
END
```

Das Debugmodul ist so zu konfigurieren, dass es für den ICON einen Boundary-Scan-Port bereitstellt. Die Verbindungen zum ICON lassen sich in der aktuellen EDK-Version nur im mhs-File ergänzen. Der ICON stellt für beide ILA-Instanzen die Kontrollbusse bereit:

```
BEGIN mdm
 ...
 PARAMETER C_USE_BSCAN = 1
 ...
 PORT bscan_tdi = v_tdi
 PORT bscan_reset = v_reset
 PORT bscan_shift = v_shift
 PORT bscan_update = v_update
 PORT bscan_capture = v_capture
 PORT bscan_sel1 = v_sel1
 PORT bscan_drck1 = v_drck1
 PORT bscan_tdo1 = v_tdo
END
BEGIN chipscope_icon
 PARAMETER INSTANCE = chipscope_icon_0
 PARAMETER HW_VER = 1.06.a
 PARAMETER C_NUM_CONTROL_PORTS = 2
 PORT control0 = icon0_ctrl
 PORT control1 = icon1_ctrl
 PORT tdi_in = v_tdi
 PORT reset_in = v_reset
 PORT shift_in = v_shift
 PORT update_in = v_update
 PORT capture_in = v_capture
 PORT sel_in = v_sel1
 PORT drck_in = v_drck1
 PORT tdo_out = v_tdo
END
```

An den IP-Core für den AXI-Bus wird der erste Kontrollbus und an den ILA der zweite Kontrollbus angeschlossen. Der AXI-Busmonitor ist fertig konfiguriert hat einen AXI-Busanschluss. Der ILA hat im Beispiel eine Triggereinheit, eine Aufzeichungstiefe von 1024 Takten und 75 Trigger-Eingänge, die gleichzeitig Dateneingänge sind. Am Trigger und Aufzeichungseingang sind als konkatinierter Bitvektor angeschlossen die Trace-Ausgänge des Prozessors »Befehl gültig! (1 Bit), »Operanden-Pipeline arbeitet« (1 Bit), »Speicherinterface arbeitet« (1 Bit), der

ICON Beispiele für die textuelle Beschreibung von Verbindungen. Verweis auf die Beispiel mhs- und ucf-Datei. Schritt-für-Schritt-Anleitung für die Erstellung mit den Wizzards.

Workaround, wenn der LA nach nach Neuladen des Programms nach wenigen Takten nur noch einsen aufzeichnet: »JTAG-Chain>Close Cable« und neu verbinden.,

Aufgabe 1.5: E

inbau eines integrierten Logikanlysators für den 7-Segment-Decoder

Erweitern Sie ihr Minimalsystem aus der Aufgabe zuvor

1.5 Softwareentwurf

Der Softwareentwurf bekommt vom Hardwareentwurf folgende Daten exportiert:

- ein Bitfile für die Programmierung der Hardwarefunktionalität
- ein bmm-File mit den Informationen wohin die Programme zu laden sind
- eine xml-Datei mit einer Beschreibung der Hardware-Komponenten zur Generierung der Treiber, für die Auswahl der Compileroptionen, auch welche Befehle implementiert sind und welche durch Programme nachzubilden sind.

Der erste Schritt zur Programmierung des entworfenen Rechners ist die Erstellung eines Bourd-Support-Packages. Dabei werden Treiber erzeugt. Für jede Harware unterschiedliche komplexe / abstrakte Funktionsschichten. Die Adresskonstanten für die Hardware-Zugriffe findet man in yparameters.h. Von den Gerätetreibern werden wir in derRegel die einfachsten benutzen, im ersten Beispiel die Makros

- XIo_In8(<Eingabeadresse>) für das Lesen von der parallelen
- XIo_out8(<Ausgabeadresse>, <Bytewert>) für das schreiben der parallelen Schnittstellen aus xio.h

sowie

- XUartLite_SendByte(<Ausgabeadresse>, <Bytewert>)
- XUartLite_RecvByte(<Ausgabeadresse>) für die serielle byteweise Aus- und Eingabe von Bytes.

Die Ein- und Ausgabeadressen ergeben sich jeweils aus der Basisadresse der Hardwareeinheit plus einem optionalen Offset des Registers, auf das schreibend oder lesend zugegriffen werden soll. Die Offsets stehen in der Hardware-Beschreibung der einzelnen Cores bzw. es gibt spezielle Treiber, in denen der Offset einprogrammiet ist.

Bei Anlegen eines C-Projekt muss immer eine Hardware-Plattform, ein Prozessor und ein Board-Support-Package mit treibern zugeordnet werden, damit das Programm übersetz- und ausführbar ist. Das nachfolgende Testprogramm versendet einen Begrüßungstext an die Konsole und kopiert in einer Endlosschleife die Schalterwerte aus dem Eingaberegister in das Ausgaberegister für die LEDs. Erstellung und Test dieses Programm ist in der Schritt-für-Schritt-Anleitung zu finden.

1.6 Arbeit mit dem Debugger

Größer Programme arbeiten selten gleich beim ersten Versuch fehlerfrei. Ein Debugger erlaubt es, ein Programm in Einzelschritten oder Stücken abzuarbeiten. Dazu soll eine etwas komplexere Berechnung der LED-Werte aus den Schalterwerten dienen:

<Programm mit Warteschleifen UP>

Damit der Comiler die zum Teil überflüssigen Zwischenschritte nicht wegoptimiert, Übersetzung mit Optimierung »O0«. Der Debugger stoppt die Progammabarbeitung zum Eintrittspunkt in das eigene Programm und danach immer, wenn die nächste Haltebedingung erreicht ist und erlaubt im angehaltenem Zustand das Lesen und Verändern der Variablen und Registerinhalte. Zur Programmfortsetzung nach jedem Halt gibt es die Optionen:

- Step over: Abarbeitung einer Einzelanweisung oder eines Unterprammaufrufs
- Step into: Abarbeitung einer Einzelanweisung oder eines Unterprammaufrufs
- Step out:

- Resume: Abarbeitung bis zum nächsten Unterbrechungspunkt oder manuellem Stop.

Unterbrechungepunkte können entweder Programm- oder Datenadressen zugeordnet werden. Der vorkonfigurierte Core hat ...

1.7 Trace

Ein Trace ist eine Signalaufzeichnung in Echtzeit und erfordert einen Logikanalysator. In dem vorbereiteten Core ist der Logikanalysator mit in das System eingebaut. Das Backend auf dem PC kommuniziert genau wie der Debugger und die Konsole über den JTAG-Testbus. Für die Aufzeichnung ist eine Triggerbedingung zu definieren, der LA zu starten und das Programm zu starten. Der LA übernimmt nach Start in jedem Takt die Eingabebitwerte und füllt und

Aufgabe 1.6: K

onfiguration eines Microblaze-Prozessors

Öffnen Sie EDK, legen Sie ein neues leeres Projekt an mit den Einstellungen ... Wählen Sie im IP-Katlog unter Prozessoren einen Microblaze aus und und konfigurieren Sie ihn wie folgt ... Welche Parameterwerte ergeben sich daraus in der mhs-Datei.

Aufgabe 1.7: E

ntwurfsfluss

Suchen Sie die Menüs/Icons für die Kontrolle der Entwurfsbeschreibung, die Generierung der Bitdatei für die Programmierung der Hardware-Funktionalität und den Export der Ergebnisses des Hardware-Entwurfs an den Software-Entwurf.

Aufgaben

- 7-Segmentanzeige an GPIO
- VHDL-Controller für eine 7-Segmentausgabe an ein GPIO
- Timer
- Interrupt
- zusätzliche UART

Extraausarbeitungen

Schritt-für-Schritt-Anleitungen zur Einarbeitung in die Aufgabenstellungen

- Schritt-für Schritt: übersetzte HW, SW-Projekt anlegen, Test mit Debugger
- Schritt-für Schritt: übersetzte HW, SW-Projekt anlegen, Test mit LA
- Schritt-für-Schritt Minimalsystem mit Wizard Kontrolle mhs, Export, Test
- Schritt-für-Schritt neues Projekt, mhs- + ucf, LA einbauen, Export, Test mit LA
- Schritt-für-Schritt neues Projekt, mhs- + ucf, Timer ergänzen, Test mit LA
- Schritt-für-Schritt neues Projekt, mhs- + ucf, Tasteneingabe + IR-Kontroller, Test mit LA

- Schritt-für-Schritt neues Projekt, mhs- + ucf, Tasteneingabe + IR-Kontroller, Test mit LA
- Seg7-Ausgabeeinheit entwerfen; ab mhs-/ucf HW-Einbindung, Test mit LA

Bugs and Workarounds

- Bei Export Design and Launch SDK: SDK beendet sich mit einer Java-Fehlermeldung. Workaround: Export and Launch SDK nochmal ohne »Bitstream regeneration«.
- Bei »Run As ...« fehlt die Auswahl »Launch on Hardware«. Workaround: Wiederholung von »Built-all«.
- Fehlermeldung nach Programmstart, ...: Nochmal gegebenenfalls mehrfach neu starten.

2 LMB-System

- 8 Eingänge für die Schalter
- 5 Eingänge für Taster
- 8 Ausgänge LED
- 4+7 Ausgänge 7-Segment
- UART 9600 kBaud
- Timer mit 1s-Interrupts für Uhr
- Timer 1ms Interrupt für Mux-Ausgabe
- Timer für eine Sleep-Funktion