Design of Digital Circuits (S1)

Capter 1, Part 1

Modelling and Simulation

Section 1.1 Design Processes

to 1.2 Modells & Simulation

Prof. G. Kemnitz

Institute of Informatics, Clausthal University of Technology

8. Mai 2012

What is Computer Engineering

Two relatively independent Zwei relativ independent sub fields:

- technical application of computer science
  - informationstechnische Erfassung, Modellierung und Steuerung technischer Systeme
- technical basis of computer science
  - Digitaltechnik: Modellierung, Simulation, Entwurf und Test digitaler Schaltungen

Die LV »Entwurf digitaler Schaltungen« legt die Grundlagen für den zweiten Teil. Lernziele: theoretisches Grundverständnis und praktische Fertigkeiten für den rechnergestützten Entwurf.

- Voraussetzung imperative Programmierung (C, Pascal o.ä.)
- Prüfung schriftlich
- Hausübungen
- begleitendes Praktikum

Entwurfsprozess

- Digitale Schaltkreise, Baugruppen und Systeme bestehen aus Millionen von logischen Bausteinen.
- Kein Mensch ist mehr in der Lage, diese Bausteine auch nur zu zählen.
- Wie werden solche Schaltungen erfolgreich entworfen?

ähnlich wie Software:

- rechnergestützt
- mit einer hierarchischen Struktur aus parametrierten Bausteinen
- Synthese aus einer Funktionsbeschreibung

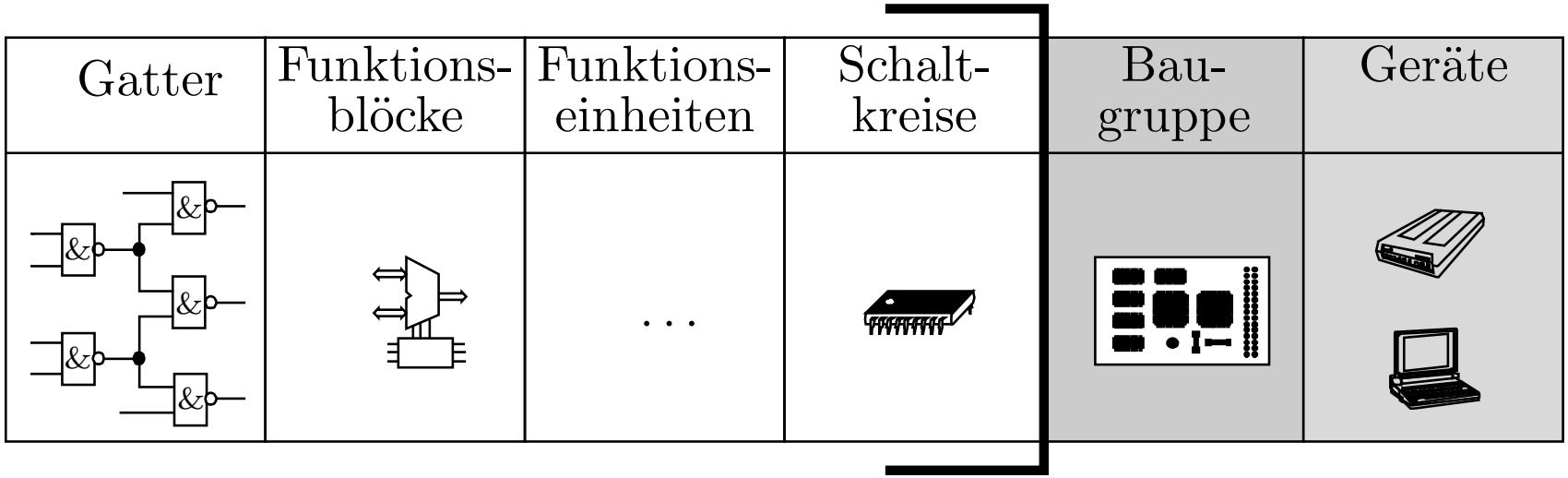
Umgang mit der Komplexität

Digitale Systeme sind groß

Wie kann man eine digitale Schaltung mit Millionen von Gattern so entwerfen, dass sie am Ende funktioniert?

- rechnergestützt in einer Programmiersprache
- hierarchisch
- nach formal überprüfbaren Entwurfsregeln
- mit Hilfe von
  - Simulation
  - Schaltungsgenerierung
  - rechnergestützter Synthese und Logikoptimierung
  - (teil-) automatischer Platzierung und Verdrahtung

Hierarchie digitaler Systeme



- Baukastenprinzip; Bibliothek zusammenpassender Bausteine + Regeln für den Zusammenbau; Mehrfachverwendung
- Entwurfskomplexität wird weniger von der Bauteilanzahl, sondern von der Typenanzahl bestimmt
- parametrisierte Bausteine, z.B. Rechenwerke mit variabler Operandenbreite + Schaltungsgeneratoren ⇒ weniger Typen

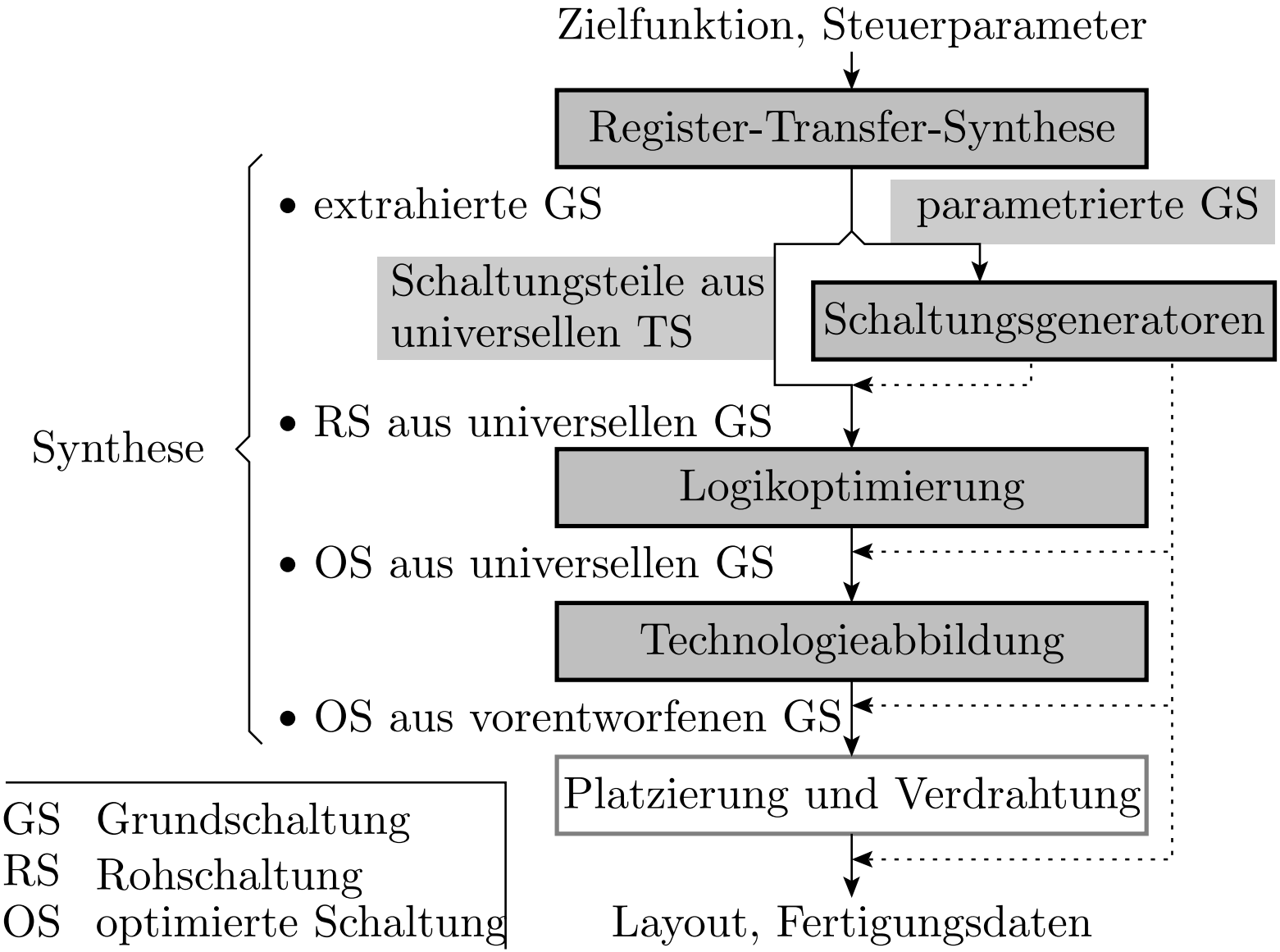
Parametrierte Schaltungen

Parametrierte Schaltungen werden als Datensätze für Algorithmen zur Schaltungsgenerierung modelliert. Beispiele parametrierter Schaltungen:

- UND-Gatter mit einem Parameter für die Eingangsanzahl und einem Parameter für die maximale Lastanzahl (Last – am Ausgang angeschlossene Gattereingänge)
- Speicher unterschiedlicher Größe und Organisation (Adressbusbreite, Datenbusbreite, ...)
- Rechenwerke mit variabler Operandenbreite

Minderung der Entwurfskomplexität durch Verringerung der Anzahl der zu unterscheidenden Bausteintypen; in der Vorlesung etwa zwei Duzend

Synthese


Zielfunktion, Steuerparameter
Register-Transfer-Synthese
• extrahierte GS
parametrierte GS
Schaltungsteile aus universellen TS
Schaltungsgeneratoren
Synthese
• RS aus universellen GS
Logikoptimierung
• OS aus universellen GS
Technologieabbildung
• OS aus vorentworfenen GS
Platzierung und Verdrahtung
GS Grundschaltung
RS Rohschaltung
OS optimierte Schaltung
Layout, Fertigungsdaten

- Register-Transfer-Synthese: Analysieren der Hochsprachenbeschreibung; Extrahieren der »Rohschaltung« (Schaltung aus Grundschaltungen (NAND2, NOR2, ...) und parametrierten Schaltungen)
- Schaltungsgenerierung: Nachbildung parametrierter Teilschaltungen durch Grundschaltungen
- Logikoptimierung: Schaltungsminimierung mittels Schaltalgebra, Konstanteneliminierung, Verschmelzung, ...
- Technologieabbildung: Nachbildung der (universellen) Grundschaltungen durch vorentworfene Teilschaltungen für eine bestimmte Fertigungstechnologie
- Platzierung: 2D-Anordnung der vorentworfenen Teilschaltungen
- Verdrahtung: Anordnung der Verbindungen

- Steuerparameter: Festlegung von Optimierungszielen (Obergrenzen für die Größe und Geschwindigkeit), Empfehlungen für Syntheseentscheidungen etc.

Simulation

- Simulationsmodell für das Testobjekt:
  - Berechnung der Ausgabesignalverläufe und der internen Signalverläufe aus den Eingabesignalverläufen
- Simulationsmodelle der Testumgebung:
  - Bereitstellung der Eingaben
  - Auswertung der Ausgaben und Zwischenergebnisse
- Jede Simulation ist ein Kompromiss zwischen Programmieraufwand, Rechenaufwand (Kosten) $\Leftrightarrow$ Anteil der erkennbaren Fehler (Nutzen)
- Kosten: Test und Fehlersuche sind bei der Entwicklung von Hardware immer am teuersten

Summary

Beherrschung der Entwurfskomplexität

- rechnergestützt
- durch die Beschreibungshierarchie
- durch parametrierte Funktionsbausteine
- durch Synthese und
- durch Simulation.

Modelle und Entwurfsraum

Der Modellbegriff in der Informatik

Ein Modell ist ein Mittel, um einen Zusammenhang zu veranschaulichen; stellt die wesentlichen Sachverhalte dar und verbirgt unwesentliche Details

formales Modell eindeutige Beschreibung der wesentlichen Merkmale (Simulationsmodell, Entwurfsergebnis)

informales Modell Beschreibung wesentlicher Zusammenhänge in einer anschaulichen Form ohne Rücksicht auf Eindeutigkeit oder Vollständigkeit (Entwurfsskizzen, Spezifikation, Lehrbuchdarstellungen)

halbformales Modell Beschreibung, aus der Fachleute in der Regel dasselbe herauslesen (Datenblätter)

Design Space

3D-Raum zur Klassifizierung der Beschreibungen (Modelle), die im Entwurfsprozess entstehen:

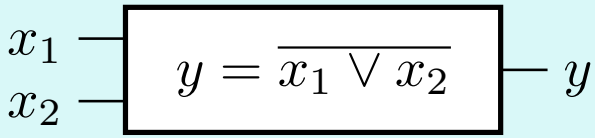
- Hierarchieebene
- hervorzuhebender Sachverhalt (Beschreibungsart)

Hierarchieebene \ Beschreibungsart	Funktionsbeschreibung	Strukturbeschreibung	geometrische Beschreibung	Testbeschreibung
Gesamtsystem				
Teilsysteme				
Funktionsblöcke		Objektraum		
Gatter				
Transistoren				

- dritte Dimension: informal, halbformal, formal

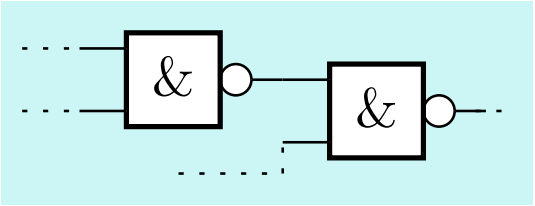
Beschreibungsart

Funktionsbeschreibung



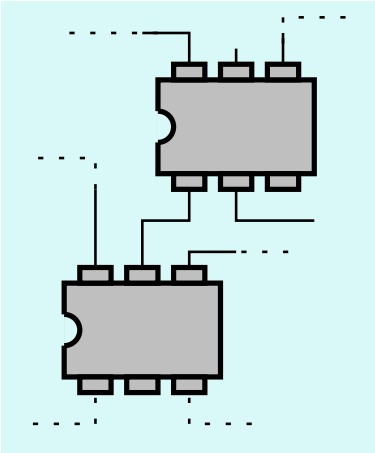
- Algorithmus oder Formalismus, zur Berechnung der Systemausgabe aus den Eingaben (Gleichungssystem, Tabellen, Programm)
- informal: Kurzbeschreibung
- halbformal: Datenblatt, Applikationsbeschreibungen
- formal: Simulationsmodell

Strukturbeschreibung



- Abstraktionsebene »Schaltung«
- Beschreibung der Zusammensetzung einer Entwurfseinheit aus Teilsystemen; hierarchisch
- Basis für einen Großteil der Schaltungsoptimierungen
- Durch Einsetzen der Funktionsbeschreibungen der Teilsysteme ⇒ Funktionsbeschreibung (eindeutige Abbildung)
- Synthese (»Funktion ⇒Struktur«) hat einen unbegrenzt großen Lösungsraum; Suche mit Kostenfunktion

geometrische Beschreibung



- Geometrische Abmessungen der einzelnen Komponenten, ihre Anordnung in einem Gerät, auf einer Leiterplatte oder in einem Schaltkreis
- Grundlage für die Generierung der Fertigungsdaten
- Beinhaltet die Strukturbeschreibung, ist aber für strukturbasierte Algorithmen viel zu überladen
- Jede Strukturbeschreibung kann durch unbegrenzt viele unterschiedliche geometrische Beschreibungen nachgebildet werden.

Eingabe	Ausgabe
0010	101
1001	011
...	...

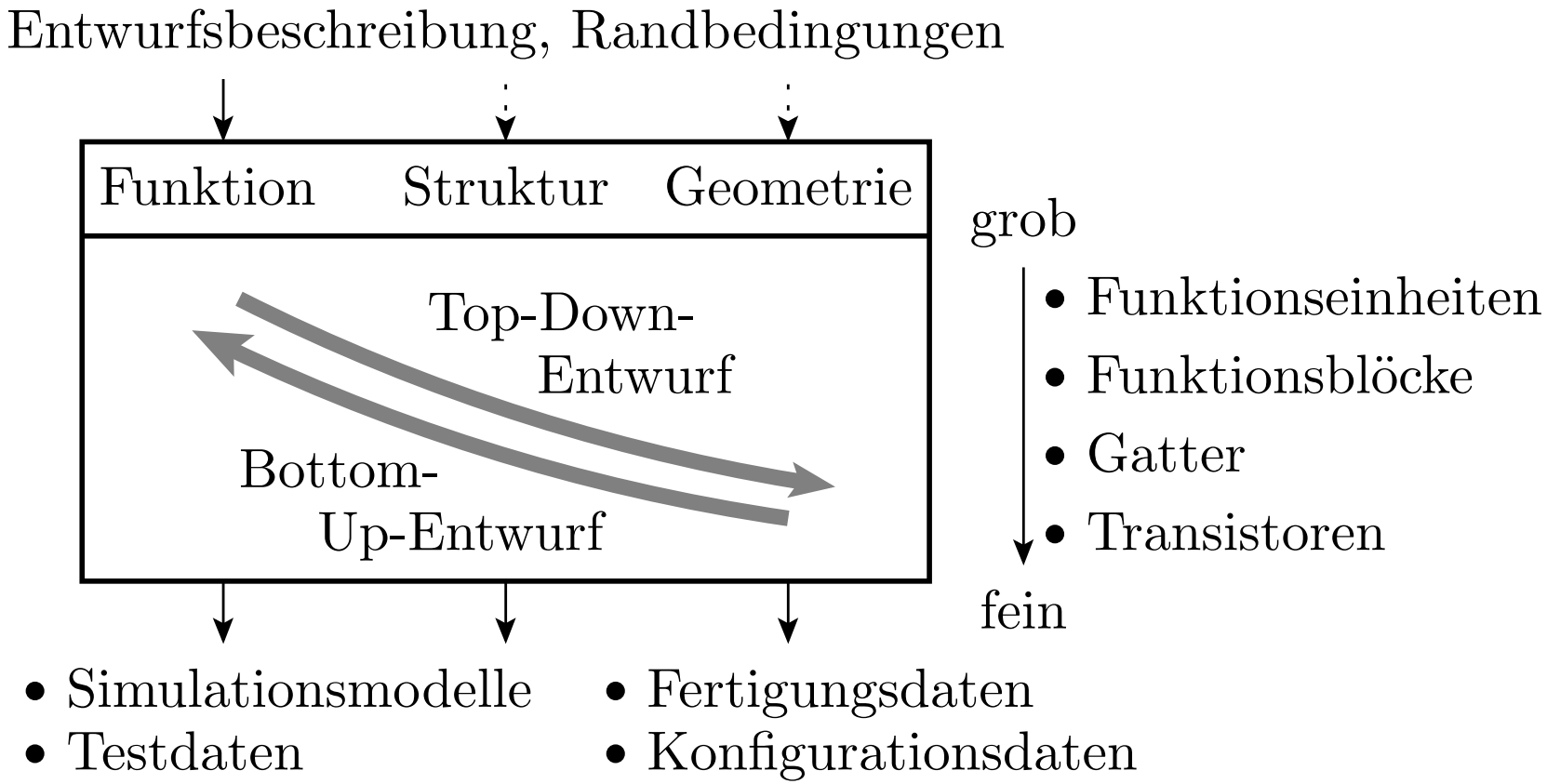
Testbeschreibung

- Eingabewerte, Algorithmen zur Bereitstellung von Eingabewerten oder Regeln für die Auswahl von Eingabewerten
- zugehörige Soll-Werte, Algorithmen oder Regeln zur Kontrolle der Ausgabewerte
- die Testvorbereitung, die Testdurchführung und die Fehlerbeseitigung verursachen immer den größten Teil des Entwurfsaufwands

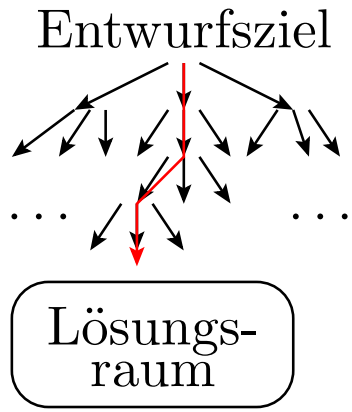
Entwurfsfluss

Entwurfsfluss

Entwurfsprozesses als »Füllen des Entwurfsraums«



Top-Down



Entwurfsfluss von oben links nach unten rechts
Bei den nachfolgenden Abbildungen gibt es jeweils unbegrenzt viele Möglichkeiten:

- Entwurfsziel (informal) ⇒ Entwurfsskizzen (Funktion, Struktur, ...; informal)
- Entwurfsskizzen (informal) ⇒Simulationsmodell (Funktion, Systemebene, formal)
- für alle Hierarchieebenen
  - Funktion ⇒Struktur aus Teilfunktionen (⇒Geometrie)
- Gesamtfunktion Struktur ⇒ Geometrie

Was tun, um sich nicht zu verirren?

- vorsichtiges Vortasten mit ständigen Kontrollen, ob noch im Lösungsraum

Phase I (auf dem Papier und im Kopf des Entwerfers):

- Spezifikation: Auflistung der Anforderungen
- schrittweise Entwicklung einer Skizze der Soll-Funktion

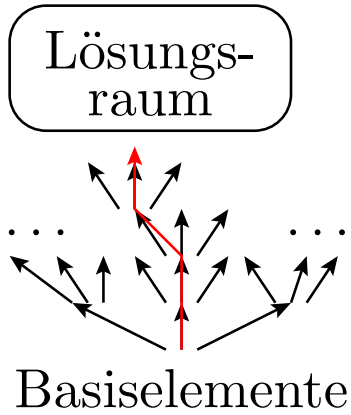
Phase II (am Rechner mit einer simulierbaren Beschreibung)

- Nachbesserungsiteration, bis die Zielfunktion o.k ist
- Nachbesserungsiteration, bis die Zielfunktion realisierbar ist
- Nachbesserungsiteration, bis das System auch alle Anforderung bezüglich der Geschwindigkeit, des Schaltungsaufwands, des Stromverbrauchs und der Testbarkeit erfüllt.

Jede Nachbesserungsiteration beinhaltet:

- umfangreiche Simulationen und Analysen
- die Entwicklung weiterer Simulationsbeispiele und
- kann Nachbesserungen auf der Ebene drüber verlangen.

Bottom-Up



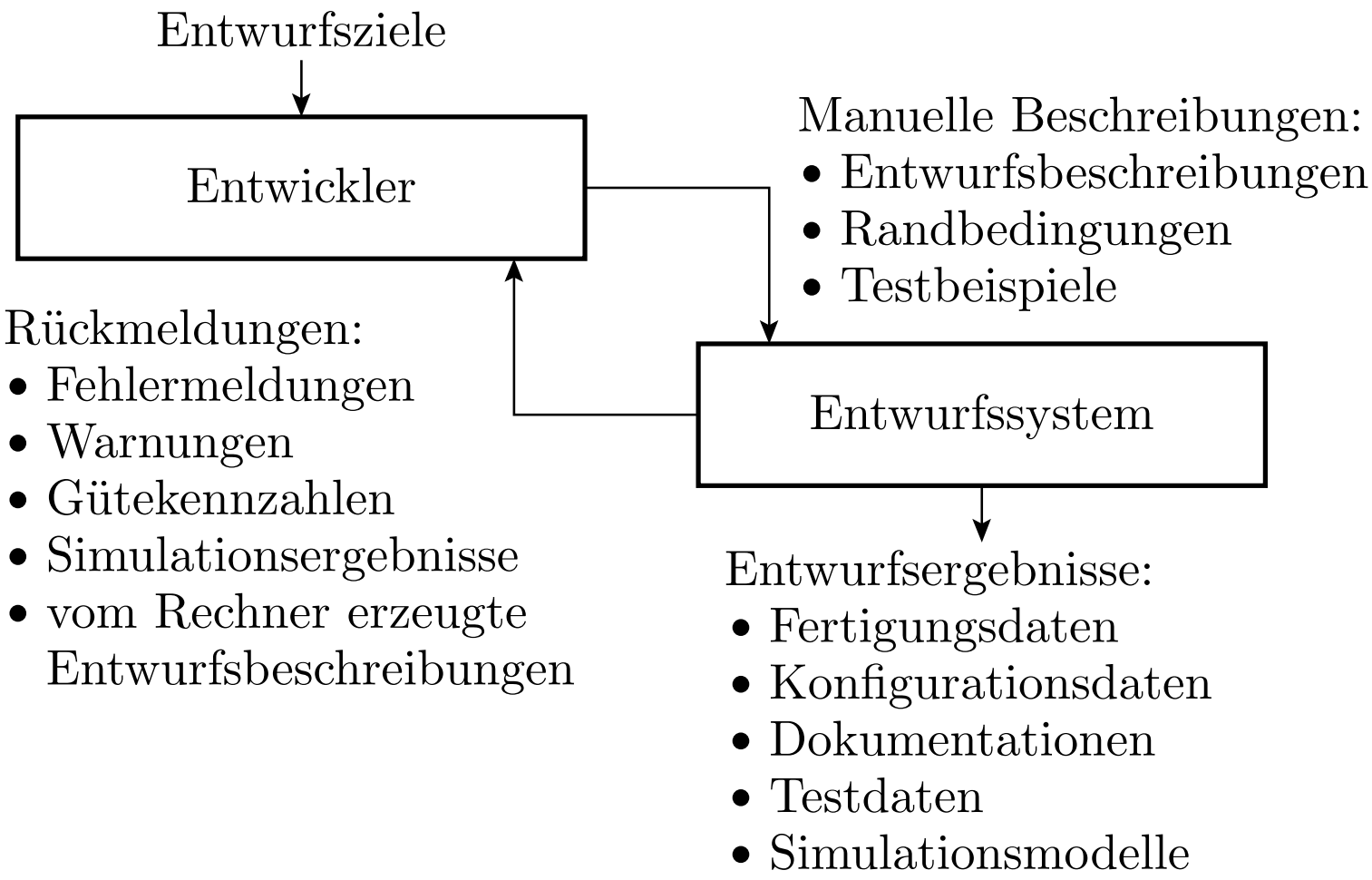
Entwurfsfluss von unten rechts nach oben links.

- Ausgangspunkt: Elemente mit bekannter Funktion und bekannter Realisierung
- Prinzip: Zusammensetzen großer aus kleinen Funktionseinheiten
- Vorteil: gute Optimierungsmöglichkeiten hinsichtlich der
  - der Größe, der Geschwindigkeit
  - des Stromverbrauchs und der Testbarkeit
- Problem: vorgegebene Zielfunktion schwer zu erreichen
- Hauptanwendung: Konstruktion von Teilschaltungen auf Vorrat (Entwurfsbibliotheken); Entwurf von parametrierten Schaltungsmodellen + Generierungsalgorithmen

---

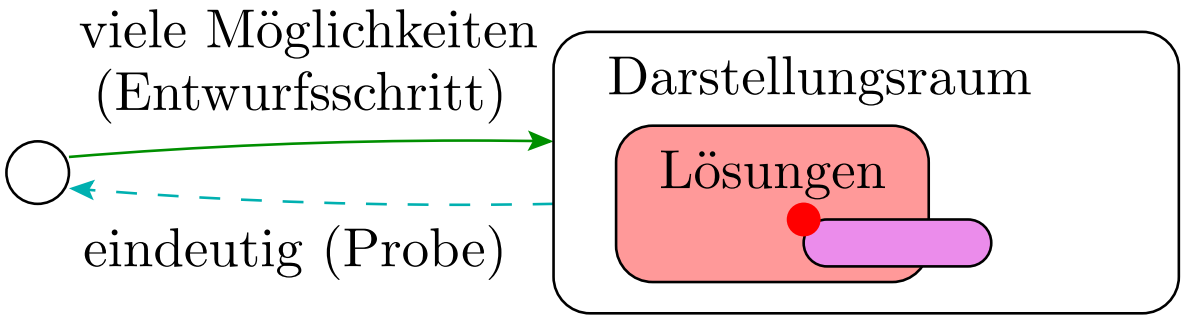
Praxis: Mischstrategie aus Top-Down- und Bottom-Up-Entwurf

Mensch-Maschine-Interaktion

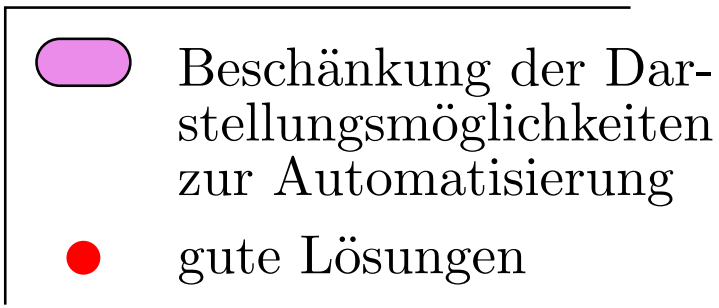


- Die Zielfunktion wird vom Entwerfer vorgegeben, der den gesamten Entwurfsprozess, in dem viele Megabyte an Beschreibungsinformationen entstehen, kontrolliert und steuert.
- Das Entwurfssystem analysiert und simuliert die Beschreibung; synthetisiert, generiert, platziert und verdrahtet die Schaltung; die Softwarewerkzeuge liefern zusätzliche Hinweise, Warnungen und Fehlermeldungen
- bei unzureichendem Ergebnis ändert der Entwerfer die Beschreibung und/oder die Steuerparameter der Werkzeuge
- Ähnlichkeit mit einem Regelkreis mit dem Entwerfer als Regler

Entwurfsautomatisierung



Vorgabe		Ziel
Aufgabe	→	Funktion
Funktion	→	Struktur
Struktur	→	Geometrie



- Lösungsraumbeschränkung schließt auch gute Lösungen aus
- Einzelergebnis im Mittel schlechter als »handoptimierte« Lösungen
- Untersuchung von viel mehr Lösungsmöglichkeiten; beste Lösung oft besser als Handentwurf

Synthesebasierte Lösungssuche (ähnlich wie genetischer Algorithmus):

- Variation auf der Ebene der manuellen Beschreibung
- Kombination guter Lösungsdetails
- manuelle Nachbesserung, wenn die Automatismen keine ausreichend gute Lösung finden.

Generelle Vorteile der Entwurfsautomatisierung:

- deutlich kürzere Gesamtentwurfszeiten
- besser vorhersagbare Güte- und Aufwandskennzahlen für die Projekt- und Kostenplanung
- eine geringere zu erwartende Anzahl von Entwurfsfehlern.

Zusammenfassung

Mega- bis Gigabyte Entwurfsdaten; 3D-Klassifikation:

- informal, halbformal, formal
- Gesamtsystem, Teilsysteme, Funktionsblöcke, Gatter, ...
- Funktion, Struktur, Geometrie und Test

Bewegung im Entwurfsraum

- Top-Down: von der Zielfunktion zur Schaltung
- Bottom-Up: Vorratsentwicklung günstiger Schaltungen

Mensch-Maschine-Kommunikation

- Entwerfer sieht einen strukturierten Entwurfsraum und Tools
- wendet Bearbeitungsmethoden auf Entwurfsobjekte an und kontrolliert die Ergebnisse; ähnlich wie Regelkreis mit Entwerfer als Regler
- Optimieren durch Probieren auf High-Level-Ebene

Beschreibungssprache (VHDL)

VHDL als formale Beschreibungsplattform

Das Akronym VHDL:

- V VHSIC (Very High Speed Integrated Circuits)
- H Hardware
- D Description
- L Language

- in Europa verbreitetste Hardwarebeschreibungssprache
- erweiterte imperative Sprache (ADA)
- andere Hardwarebeschreibungssprachen: Verilog, System-C

- Mit VHDL werden Funktionen, Strukturen und Testabläufe vom Gesamtsystem bis zum Gatterniveau beschrieben

Ebene in der Hirarchie \ Beschreibungsart	Funktions-beschreibung	Struktur-beschreibung	geometrische Beschreibung	Test-Beschreibung
Gesamtsystem	+	+		+
Teilsysteme	+	+		+
Funktionsblöcke	+	+		+
Gatter	+	+		+
Transistoren	·	·		·

+ üblich
· möglich

- Bestandteile eines VHDL-Projekts:
  - Entwurfseinheiten
  - Packages
  - Bibliotheken

Beschreibungsstruktur einer Entwurfseinheit:

```
 1: library IEEE;
 2: use IEEE.STD_LOGIC_1164.all;
--- Schnittstellenbeschreibung
 3: entity Schaltungsname is
 4:  [generic (List_der_Konfigurationsparameter);]
 5:  [port(Liste_der_Anschlusssignale);]
 6: end entity;
--- Beschreibung des Innenlebens
 7: architecture Beschreibung of Schaltungsname is
 8:  –Vereinbarung;"
 9: begin
10:  –Anweisung;"
11: end architecture;
```

Zeile 1, 2: Library- und Use-Anweisung; Einblendung von Bibliotheksobjekten in den Namensraum

Zeile 3-6: Schnittstellenbeschreibung; Vereinbarung der Konfigurationsparameter und Anschlusssignale

Zeile 7-11: Beschreibung der Entwurfseinheit

Zeile 8: Vereinbarungen von Datentypen, Signale, Unterprogramme etc.

Zeile 10: Anweisungen zur Beschreibung des Innenlebens

Beschreibungsmittel für das Innenleben:

- Einbindung von Teilschaltungen (⇒ Strukturbeschreibung)
- Prozesse: Rahmen zur Verhaltensbeschreibung mit imperativen Mitteln (⇒ Funktionsbeschreibung)
- Kurzschreibweisen für Verhaltensmodelle für Grundschaltungen (Gatter, Multiplexer, ...)

Im Weiteren verwendete Schreibweisen/Farben

key	Schlüsselwort
[...]	optionales Element, darf Null mal oder einmal enthalten sein
{...}	optionales Element, darf beliebig oft enthalten sein
...|...	Alternative, eines der aufgezählten Elemente muss enthalten sein
Symbol	Methasymbol (Nichtterminalsymbol): Symbol das nach weiteren Regeln zu ersetzen ist.
Name, IEEE	eigener Bezeichner, standardisierter Bezeichner
1,"ab", ...	Wertangaben (Zahlen, Zeichenketten, ...)
--- Text	Kommentar

Bezeichner sind explizit zu definieren

- keine Mehrfachdefinition
- keine Schlüsselworte

in VHDL

- beginnen Bezeichner mit einem Buchstaben
- bestehen nur aus den Buchstaben 'A' bis 'Z', 'a' bis 'z', den Ziffern '0' bis '9' und dem Unterstrich '_'
- dürfen nicht nicht mit einem Unterstrich enden und keine zwei Unterstriche hintereinander enthalten
- keine Unterscheidung zwischen Groß- und Kleinschreibung

Simulierbare Entwurfseinheit (Testrahmen)

- keine Anschlusssignale
- imperative Teilbeschreibungen in Prozesse gekapselt

```
 1: entity hallo_welt is
 2: end entity;

 3: architecture a of hallo_welt is
 4: begin
 5:   process
 6:   begin
--- Meldung ausgeben und Prozess beenden
 7:     report "Hallo Welt";
 8:     wait;
 9:   end process;
10: end architecture;
```

⇒WEB-Projekt: HalloWelt

- Simulationsausgabe:

```
HalloWelt.vhdl:7:5:@0ms:(report note): Hallo Welt
```

Größere Teilbeschreibungen, die keine Entwurfseinheiten sind, werden in Packages ausgelagert: Typenvereinbarungen, Konstanten, Unterprogramme, ...

```
--- Definitionssteil, exportierte Vereinbarungen
 1: package HalloWelt_pack is
 2:  constant c: STRING:="Hallo Zeichenkette";
 3:  function SchreibeText(s: STRING) return STRING;
 4: end package;
--- Beschreibung der Unterprogramme
 5: package body HalloWelt_pack is
 6:  function SchreibeText(s: STRING) return STRING is
 7:  begin
 8:   return s;
 9:  end function;
10: end package body;
```

⇒WEB-Projekt: HalloWelt_pack

Nutzung eines Packages

```
 1: use WORK.HalloWelt_pack.all;
 2: entity HalloWelt1 is
 3: end entity;
 4: architecture a of HalloWelt1 is
 5: begin
 6:   process
 7:   begin
 8:     report "Hallo Welt";
 9:     wait for 1 ns;
10:     report c;
11:     wait for 1 ns;
12:     report SchreibeText("Hallo Funktionsaufruf");
13:     wait;
14:   end process;
15: end architecture;
```

⇒WEB-Projekt: HalloWelt

Aufgaben

Aufgabe 1.1:

- Was ist eine Strukturbeschreibung, eine Funktionsbeschreibung und eine geometrische Beschreibung?
- Wozu dient der Testrahmen in einem Simulationsmodell?
- Was sind die Merkmale, die Unterschiede, die Vorteile und die Nachteile formaler und informaler Modelle?

Modelle & Simulation

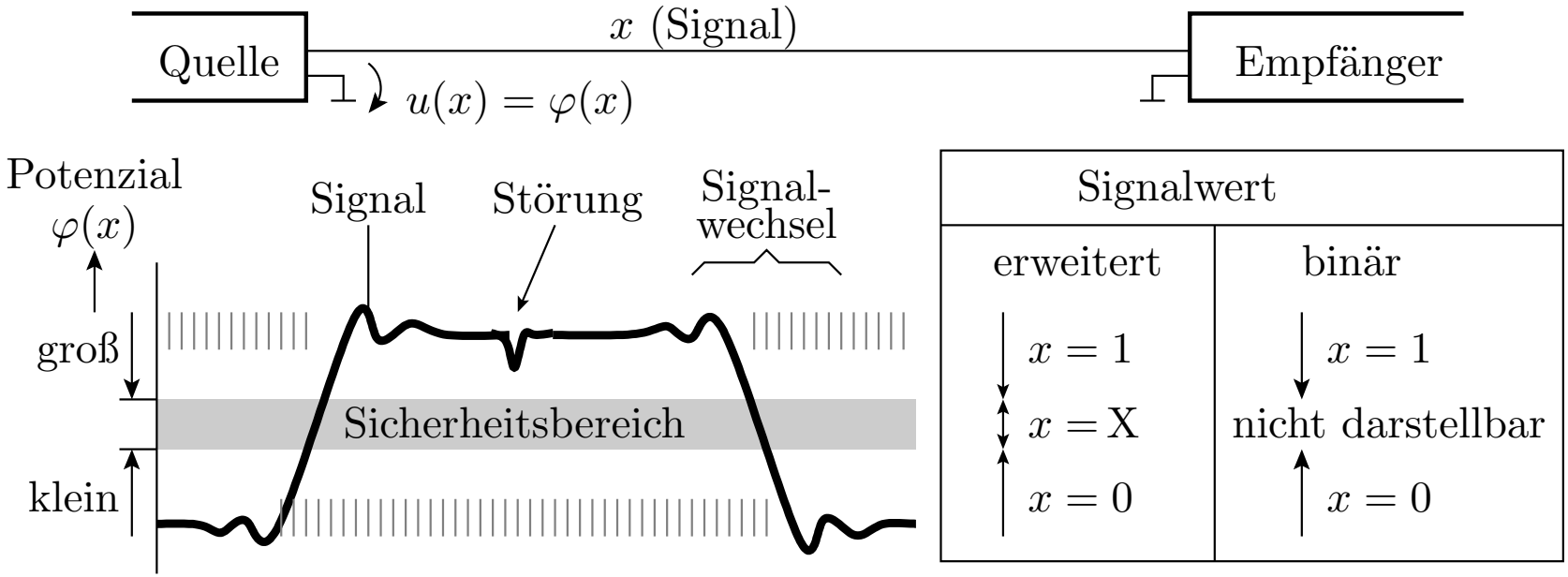
Begriffe

Eine digitale Schaltung ist ein System aus signalverarbeitenden Bausteinen (Teilsystemen), die über Signale miteinander kommunizieren. Grundbegriffe:

- Kombinatorische Schaltung (engl. combinational circuit, dt. auch Schaltnetz): Schaltung ohne Speicherverhalten; Ausgabe nur abhängig von den aktuellen Eingaben
- Sequentielle Schaltung (engl. sequential circuit, dt. auch Schaltwerk): Schaltung mit Speicherverhalten; Ausgabe auch abhängig von früheren Eingaben
- Signal: zeitlicher Werteverlauf einer physikalischen Größe.
- Datentyp: beschreibt den Wertebereich eines Datenobjekts (Signal, Variable, Konstante) und optional die Wertedarstellung

Signale, Datentypen

Binäre Signale und Datentypen



Der Datentyp beschreibt Wertebereich [+ Wertedarstellung]

Bit: zwei Wert + ungültig[1]

- Logik: $\mathcal{B}_\mathrm{L} = \{\text{falsch}, \text{wahr}\}$
- Digitaltechnik: $\mathcal{B} = \{0, 1\,[, \mathrm{X}]\}$

[1] Signale dürfen nur innerhalb der Gültigkeitsfenster ausgewertet werden

VHDL-Datentypen für die Darstellung von Bits

- In VHDL sind Bit-Typen Aufzählungstypen:

```
type BIT is ('0', '1');
type BOOLEAN is (FALSE, TRUE);
type STD_LOGIC is ('U', 'X', '0', '1', ...)[2];
```

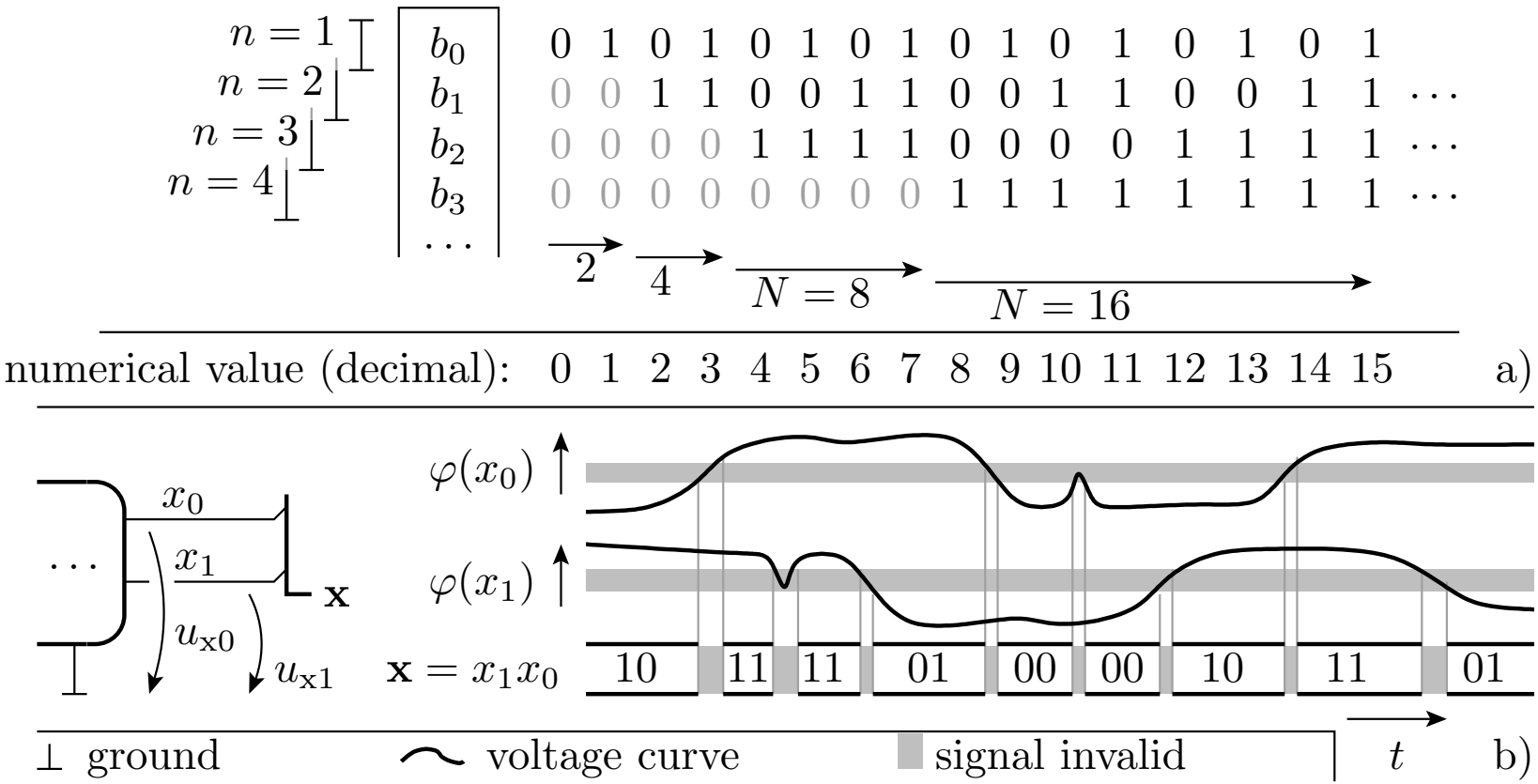
'0', '1' – druckbare Zeichen

FALSE, TRUE – symbolische Konstanten

```
variable b1, b2: BOOLEAN := TRUE;
...
b1 := b1 and b2;
```

[2] im Package IEEE_STD_LOGIC_1164 definiert; 'U', 'X', ... – Pseudo-Signalwerte für Simulationsmodelle, können ungenutzt bleiben; vom Standard als Ersatz für der Typ BIT empfohlen

Bitvektoren



- n Bits $\Rightarrow N \leq 2^n$ unterscheidbare Werte
- $N > 2$ zu unterscheidende Werte $\Rightarrow n \geq \log_2(N)$
- ein BV ist nur gültig, wenn alle Bits gültig sind

Bitvektortypen und Zahlentypen

- Bitvektoren 1D-Felder aus Bits:

```
type BIT_VECTOR is array (NATURAL range <>) of BIT;
type STD_LOGIC_VECTOR is array(NATURAL range<>) of
  STD_LOGIC;
```

- Bitvektorkonstanten sind Zeichenketten, z.B. "0101","0X1"
- Die Indexbereiche sind Zahlentypen:

```
type INTEGER range -2**31 to 2**31-1;
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
```

- (*Zahlentyp* range<>) – Obergrenze des Indexbereichs

```
signal slv: STD_LOGIC_VECTOR(3 downto 0);
variable idx: NATURAL;
...
slv(idx+1) <= slv(idx);
```

Ohne Textverarbeitung geht es nicht

```
type character is (      -- 128 Zeichen des ASCII character set
nul, soh, stx, etx, eot, enq, ack, bel, bs,  ht,  lf,  vt,  lf,  cr,  so,  si,
dle, dc1, dc2,  dc3  dc4 nak, syn, etb, can,  em,  sub, esc, fsp, gsp, rsp, usp,
' ', '!', '"', '#', '$', '%', '&', ''', '(', ')', '*', '+', ',', '-', '.', '/',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?',
'@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\', ']', '^', '_',
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', del,
  ...);                  -- 128 additional characters (ISP 8859 Latin-1)
```

```
type STRING is array(POSITIVE range <>) of CHARACTER;
```

- (POSITIVE range <>) $\Rightarrow 1 \leq$ idx $\leq$ INTEGER'HIGH
- »Ä«, »Ö«, »Ü« und »ß« selbst in Kommentaren unzulässig

Die Zeit hat einen eigenen Typ

```
type TIME is range Minimalwert to Maximalwert
    units fs;

        ps  = 1000 fs;    us  = 1000 ps;
        ms  = 1000 us;    sec = 1000 ms;
        min =   60 sec;   hr  =   60 min;
    end units;
```

Untertyp für positive Zeiten:

```
subtype DELAY_LENGTH is TIME range 0 fs to TIME'HIGH;
```

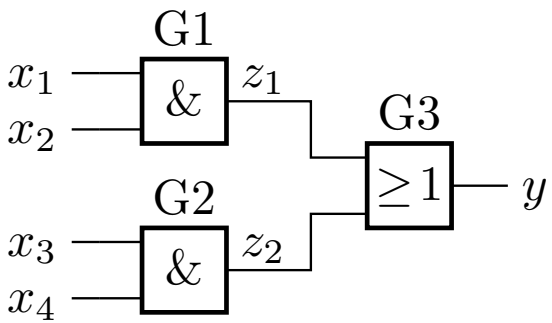
```
signal x, y: STD_LOGIC_VECTOR(3 downto 0);
constant td: DELAY_LENGTH := 2 ns;
...
y <= x after td;
```

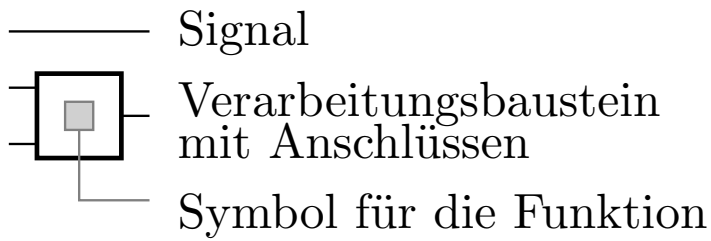
Signalflussplan

Schaltungsbeschreibung durch einen Signalflussplan

Strukturbeschreibung / Graphendarstellung

- Teilsysteme ⇒ Knoten
- Signale ⇒ Kanten, gerichtet vom Sender zum Empfänger

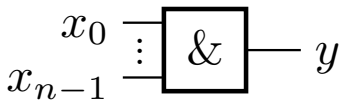
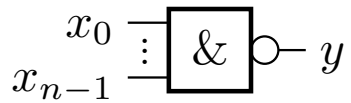
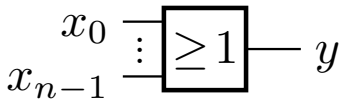
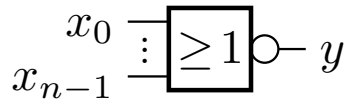
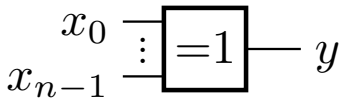
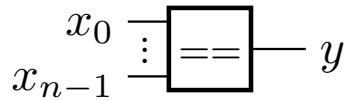
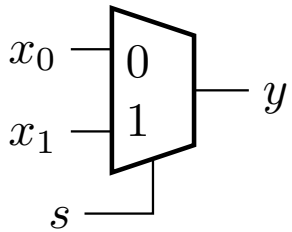
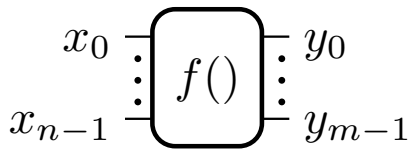




- vorerst Beschränkung auf kombinatorische Schaltungen:
  - Bausteine ohne Speicherverhalten
  - gerichteter azyklischer (schleifenfreier) Signalfluss

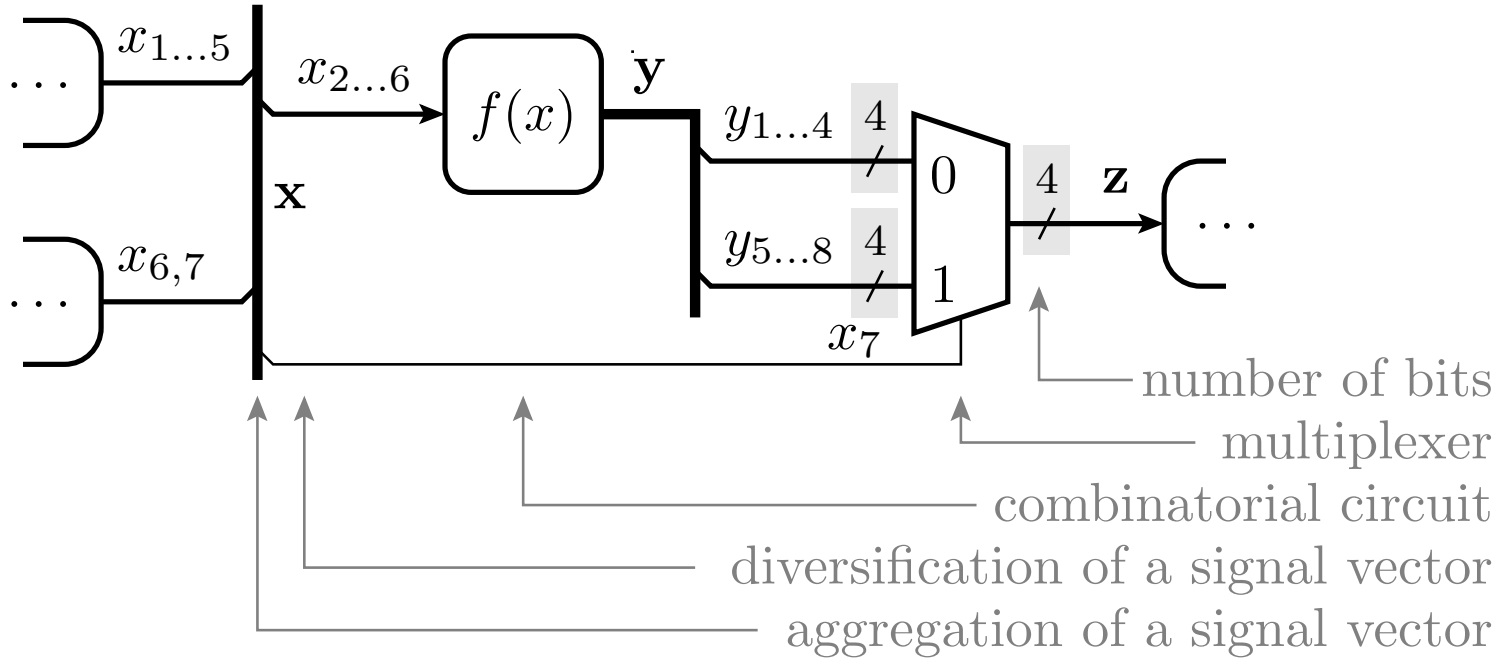
- Signale haben einen Bezeichner und einen Typ

Logische Grundbausteine

logische Funktion	Symbol	logische Funktion	Symbol
Treiber (Identität): $y <= x$		Inverter: $y <= \bar{x}$	
UND: $y <= x_0 \wedge x_1 \wedge \ldots$		NAND: $y <= \overline{x_0 \wedge x_1 \wedge \ldots}$	
ODER: $y <= x_0 \vee x_1 \vee \ldots$		NOR: $y <= \overline{x_0 \vee x_1 \vee \ldots}$	
XOR: $y <= x_0 \oplus x_1 \oplus \ldots$		XNOR: $y <= \overline{x_0 \oplus x_1 \oplus \ldots}$	
Multiplexer: wenn $s = 0$ dann $y <= x_0$ sonst $y <= x_1$		allg. Funktion: $\mathbf{y} <= f(\mathbf{x})$	

generische Modelle mit n als Parameter; Anschlusstyp BIT oder STD_LOGIC; links Eingänge rechts Ausgänge

Signalflusspläne auf der Funktionsblockebene



- Zusammenfassung von Bits zu Bussen; Typ BIT_VECTOR, STD_LOGIC_VECTOR, später auch weiter Typen
- Teilschaltungen mit Funktions- oder Strukturbeschreibung
- in bitorientierte Signalflusspläne auflösbar
- oft nur halbformal oder informal (Skizze von Teilaspekten)

Imperative Modelle

Imperative Funktionsmodelle

Die Funktion der Teilschaltungen wird imperativ beschrieben; als normales Programm für einen normalen Rechner; wie normale Programm testbar. Testrahmen zum Experimentieren:

```
--- Vorspann
library IEEE;
use IEEE.STD_LOGIC_1164.all;
library Tuc;
use Tuc.Ausgabe.all;
--- leere Schnittstelle
entity test is end entity;
...
```

- Package IEEE.STD_LOGIC_1164 stellt Typen und Operationen für STD_LOGIC[_VECTOR] bereit
- Package »Tuc.Ausgabe« enthält Typen + Unterprogramme zur Textausgabe

```
architecture a of test is
--- Vereinbarungsteil der Entwurfseinheit
    Signalvereinbarungen
begin
 process
 --- Vereinbarungsteil Prozess
    Variablenvereinbarungen
 begin
 --- Anweisungsteil Prozess
    zu_testende_Anweisungen
  wait; --- beendet Simulation
 end process;
end architecture;
```

```
--- Vereinbarungsteil Prozess
variable a, b: STD_LOGIC;
--- Anweisungsteil Prozess
a := '1'; b:='X';
write("a and b:" & str(a and b));
```

a :='1'; b :='X';	Wertzuweisungen an Variablen
write(*Text*);	Ausgabe auf dem Standardterminal
"a and b:"	konstante Zeichenkette
Text1 & *Text2*	Verkettung zweier Zeichenketten
str(*Ausdruck*)	Umwandlung des Ausdrucks in eine Textdarstellung

---

Auf den Folien stehen im Weiteren nur die Vereinbarungen und Anweisungen; Download der kompletten Programme + Download-Link für Simulator + Hilfe zum Übersetzen und Ausführen:

http://techwww.in.tu-clausthal.de/site/Lehre/VHDL-Web-Projekte/

Simulator: freie Software und auch im Übungsraum installiert.

Definition von Datenobjekten und Zuweisungen

constant *Bezeichner−,Bezeichner″*: *Typ* [:=*aw*];
variable *Bezeichner−,Bezeichner″*: *Typ* [:=*aw*];
signal *Bezeichner−,Bezeichner″*: *Typ* [:=*aw*];

aw – Anfangswert; ohne Angabe erster Wert der Typendefinition

BIT	STD_LOGIC	BOOLEAN	INTEGER	NATURAL	POSITIVE
'0'	'U'	FALSE	INTEGER'LOW	0	1

```
signal i: INTEGER;
signal b: STD_LOGIC;
signal p: POSITIVE;
...
write("i=" & str(i) & " b=" & str(b) & " p=" & str(p));
```

Was wird ausgegeben?

- die Indexbereich von Bitvektoren kann absteigend oder aufsteigend vereinbart werden
- Elementezuordnung bei Wertezuweisungen von links nach rechts

```
signal s: STD_LOGIC_VECTOR(0 to 3)      := "1100";
signal f: STD_LOGIC_VECTOR(3 downto 0) := "1100";
```

Vektorelement	s(0)	s(1)	s(2)	s(3)	f(3)	f(2)	f(1)	f(0)
Anfangswert	1	1	0	0	1	1	0	0

- Ohne Anfangswertangabe Elementeinitialisierung mit Standardwerten

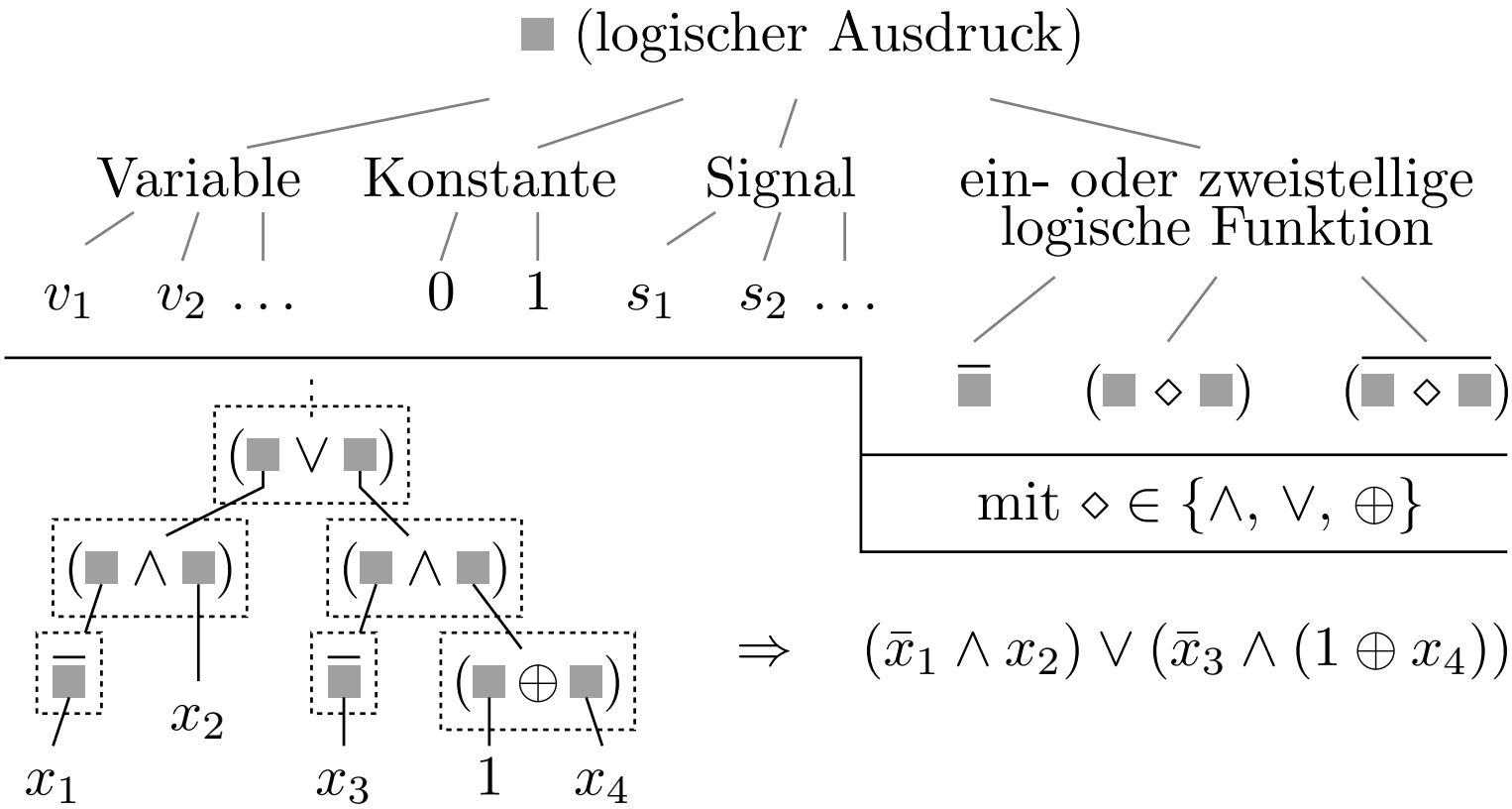
---

```
signal slv: STD_LOGIC_VECTOR(3 downto 0);
...
write("slv=" & str(slv)); --- Was wird ausgegeben?
```

Variablenzuweisungen und Ausdrücke

Variablenname := *Ausdruck*;

Ein Ausdruck ist eine Textbeschreibung für einen baumartigen Berechnungsfluss:



$x_2\,x_1$	Inverter $\bar{x}_1$	UND $x_1 \wedge x_2$	NAND $\overline{x_1 \wedge x_2}$	ODER $x_1 \vee x_2$	NOR $\overline{x_1 \vee x_2}$	XOR $x_1 \oplus x_2$	XNOR $\overline{x_1 \oplus x_2}$
0 0	1	0	1	0	1	0	1
0 1	0	0	1	1	0	1	0
1 0		0	1	1	0	1	0
1 1		1	0	1	0	0	1
VHDL-Schlüsselwort	not	and	nand	or	nor	xor	xnor

- »not« hat Vorrang
- 2-stellige VHDL-Operatoren sind gleichberechtigt
- bei unterschiedlichen Operatoren Ausführungsreihenfolge mit Klammern festlegen

---

Was wird ausgegeben?

```
write(str((TRUE or FALSE) and TRUE));   --- TRUE
write(str(FALSE or (FALSE and TRUE)));  --- FALSE
write(str(TRUE or FALSE and TRUE));     --- Fehler
```

Verarbeitung ungültiger Bitwerte

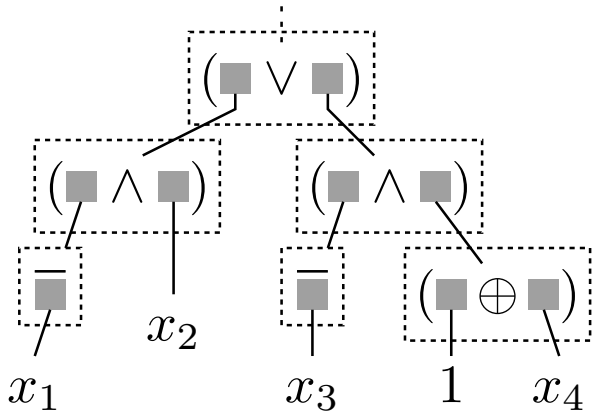
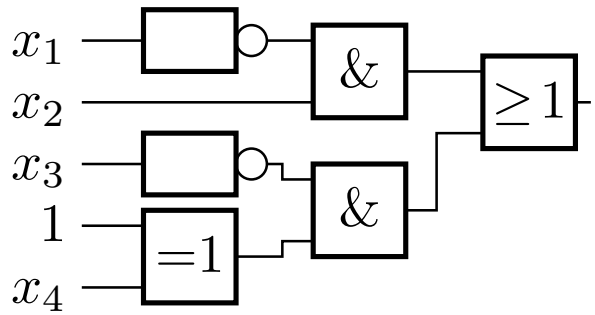
```
type STD_LOGIC is ('U', 'X', '0', '1', ...)
```

- ungültig (X): Wert kann »0« oder »1« sein
- Simulationsregeln:

$$\overline{\mathrm{X}} = \mathrm{X} \qquad \mathrm{X} \wedge 0 = 0 \qquad \mathrm{X} \vee 0 = \mathrm{X} \qquad \mathrm{X} \oplus 0 = \mathrm{X}$$
$$\mathrm{X} \wedge 1 = \mathrm{X} \qquad \mathrm{X} \vee 1 = 1 \qquad \mathrm{X} \oplus 1 = \mathrm{X}$$
$$\mathrm{X} \wedge \mathrm{X} = \mathrm{X} \qquad \mathrm{X} \vee \mathrm{X} = \mathrm{X} \qquad \mathrm{X} \oplus \mathrm{X} = \mathrm{X}$$

```
variable x: STD_LOGIC_VECTOR(3 downto 0);
variable y: STD_LOGIC;
...
x := "1001"; y := (x(3) and x(2)) or (x(1) and x(0));
x := "10X1"; y := (x(3) and x(2)) or (x(1) and x(0));
x := "X011"; y := (x(3) and x(2)) or (x(1) and x(0));
```

Ausdruck ⇔ Berechnungsfluss ⇔ Signalfluss

Ausdruck	Berechnungsbaum	Signalflussgraph
$(\bar{x}_1 \wedge x_2) \vee ($ $\bar{x}_3 \wedge (1 \oplus x_4))$		
	Ergebnisweitergabe Operation	Signale Verarbeitungselement

- formal ineinander umrechenbare Beschreibungsformen
- typ. Entwurfsablauf für kombinatorische Schaltungen
  - Funktionsdefinition als simulierbarer Berechnungsfluss
  - Optimierung als logische Ausdrücke

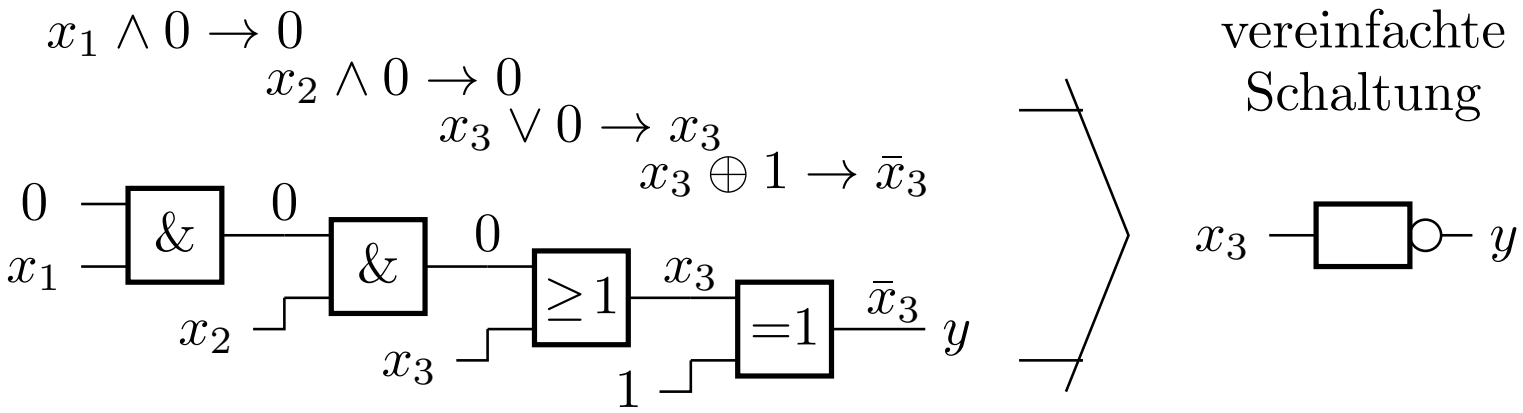
Optimierung durch Konstantenelimination

Operationen mit Konstanten durch Ergebnis ersetzen

- Beschreibungsebene »logische Ausdrücke«

$$\begin{aligned}((((x_1 \wedge 0) \wedge x_2) \vee x_3) \oplus 1) &= (((0 \wedge x_2) \vee x_3) \oplus 1) \\ &= ((0 \vee x_3) \oplus 1) = (x_3 \oplus 1) = \bar{x}_3\end{aligned}$$

- Beschreibungsebene »Signalfluss/Schaltung«



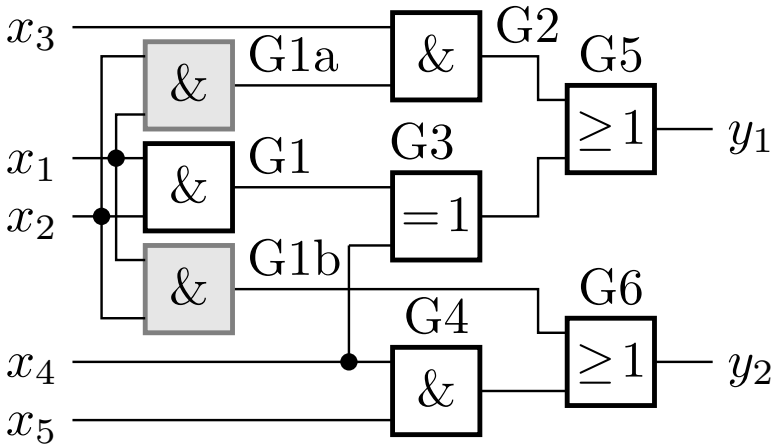
Optimierung durch Verschmelzung

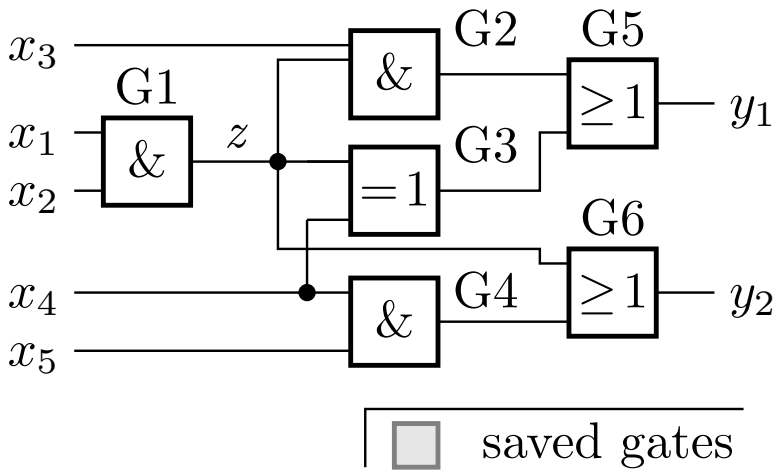
mehrfach benötigte Teilergebnisse nur einmal zu berechnen

- Beschreibungsebene »logische Ausdrücke«

$$y_1 := x_3x_2x_1 \vee (x_4 \oplus x_2x_1)$$
$$y_2 := x_5x_4 \vee x_2x_1$$

- Beschreibungsebene »Signalfluss/Schaltung«





Signalzuweisung

- einem Signal werden Werte-Zeit-Tupel zugewiesen

  Signalname <= [*VM*] *Ausdruck* [after *td*]
  [, *Ausdruck* [after *td*]]*;

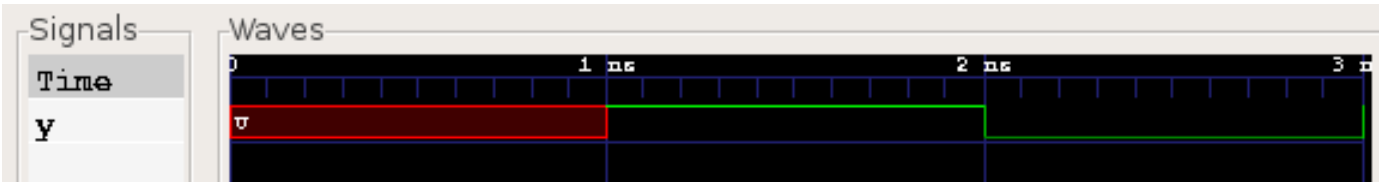
  VM -- Verzögerungsmodell; *Ausdruck* -- Ausdruck Wert; *td* -- Ausdruck Verzögerungszeit

---

```
signal y: std_logic;
...
y <= '1' after 1 ns, '0' after 2 ns, '1' after 3 ns;
```

- Visualisierung:

```
ghdl -r Signal1 --wave=Signalverlauf.ghw
gtkwave Signalverlauf.ghw [gtkwave_setup.sav]
```



Warteanweisungen und Weckbedingungen

Eine Warteanweisung legt den Prozess, bis die Weckbedingung erfüllt ist, schlafen. Weckbedingungen:

- die Änderung eines Signals aus einer Weckliste

  ```
  wait on Weckliste;
  ```

- das Weiterrücken der Simulationszeit um eine Verzögerungszeit

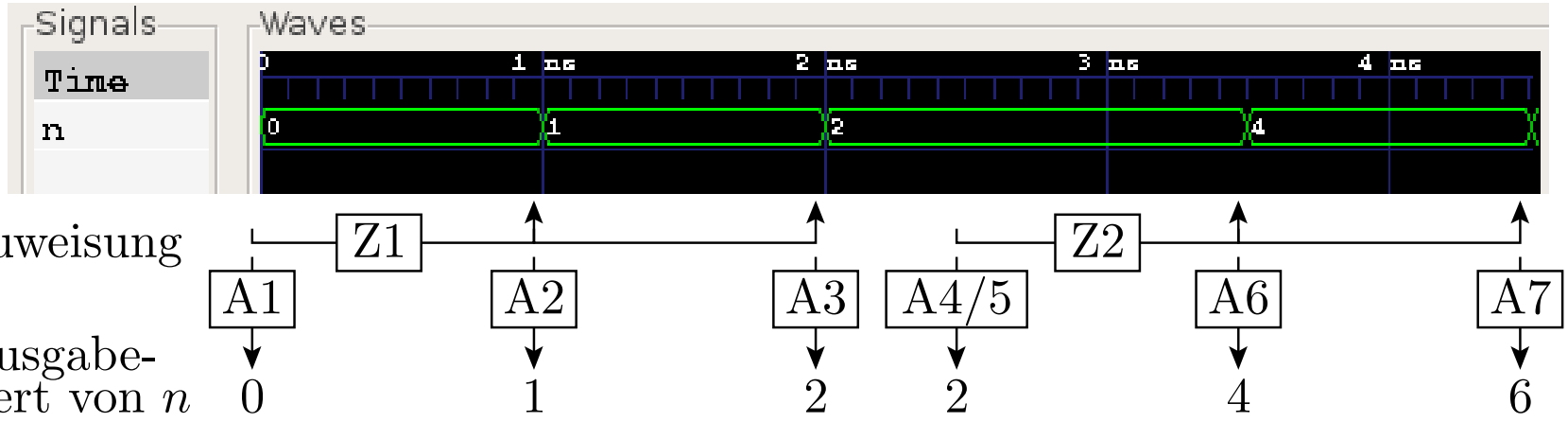
  ```
  wait for Verzögerungszeit;
  ```

- eine beliebige Bedingung, die Signalwerte oder die Simulationszeit auswerten:

  ```
  wait until Bedingung;
  ```

- Zuweisungszeitpunkt einer Änderung: NOW + t_d
  (NOW – aktuelle Simulationszeit; t_d – Verzögerungszeit)

```
signal n: NATURAL;
...
Z1: n <= n+1 after 1 ns, n+2 after 2 ns;
                write("A1:" & str(now)&':'&str(n));
wait on n;      write("A2:" & str(NOW)&':'&str(n));
wait on n;      write("A3:" & str(NOW)&':'&str(n));
wait for 0.5 ns;write("A4:" & str(NOW)&':'&str(n));
Z2: n <= n+2 after 1 ns, n+4 after 2 ns;
                write("A5:" & str(NOW)&':'&str(n));
wait on n;      write("A6:" & str(NOW)&':'&str(n));
wait on n;      write("A7:" & str(NOW)&':'&str(n));
```



Summary

- Testrahmenschablone für einzelne Anweisungen; Testausgaben

  ```
  write(Text & str(Datenobjekt) & ...);
  ```

- Datenobjekte: Konstanten, Variablen und Signale; Vereinbarung:

  ```
  Objekttyp Bezeichner-,Bezeichner": Typ [:=aw];
  ```

- Variablenzuweisung/Ausdrücke:

  ```
  Variablenname := Ausdruck;
  ```

  logische Ausdrücke ⇔ Berechnungsfluss ⇔ Signalfluss
- Signalzuweisungen

  ```
  Signalname <= [VM] Ausdruck [after td]
  wait on/for/until Weckbedingung
  ```

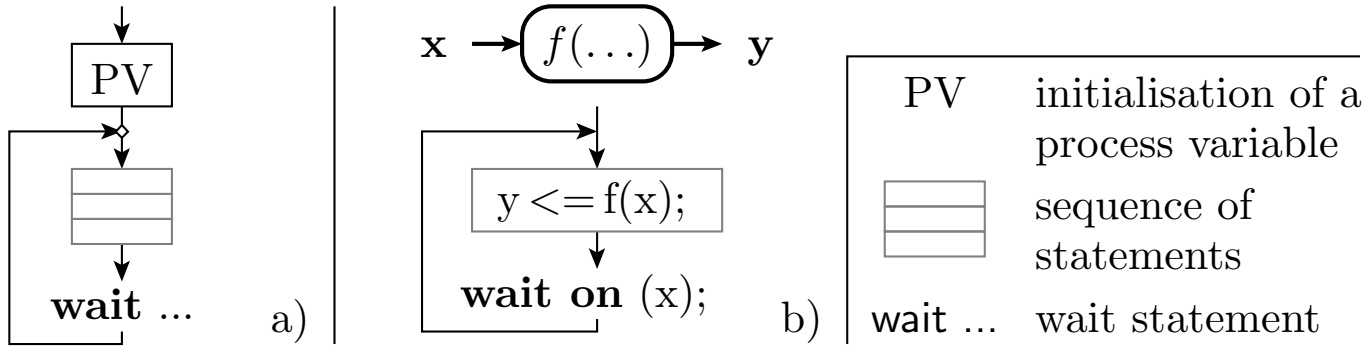
  viel komplizierter als die Beschreibung von Funktionen mit einem sequentiellen Berechnungsfluss

Event Driven Sim

Event Driven Simulation

- Each subcircuit monitors permanently it's input signals
- In case of transition the new output values are calculated and delayed assigned
- Output transitions Ausgabeänderungen trigger new calculation in subsequent circuits

Simulation model of a subcircuit

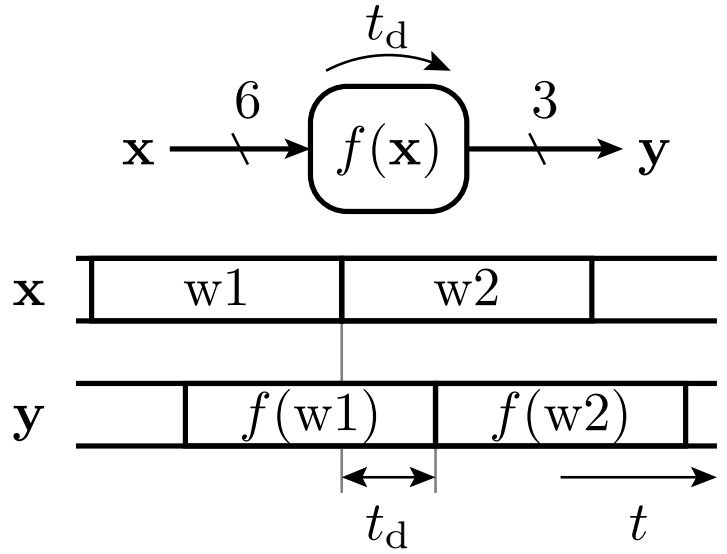


infinite loop:

- calculation of output values out of the input values
- delayed assignment to the output signal
- wait for an input transition

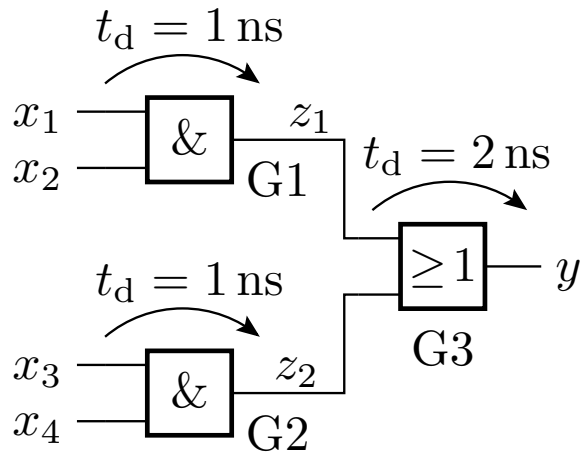
Example

```
signal x: STD_LOGIC_VECTOR(5 downto 0);
signal y: STD_LOGIC_VECTOR(2 downto 0);
constant td: DELAY_LENGTH := 1 ns;
...
process
begin
 y <= f(x) after td;
 wait on x;
end process;
```



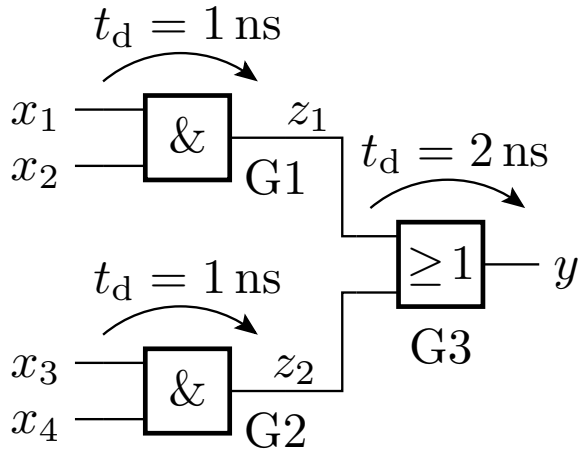
- f(x) function /algorithm to calculate the output: expression, multiple statements, subroutine, ...

Circuit of three Gates



- one process per gate
  + input process

```
entity SimModell is
end entity;

architecture Sim of SimModell is
 signal x1, x2, x3, x4, z1, z2, y: STD_LOGIC:='0';
begin

--- Simulation G1
G1: process
begin
 z1<=x1 and x2 after 1 ns;
 wait on x1, x2;
end process;

--- Simulation G2
G2: process
begin
 z2<=x3 and x4 after 1 ns;
 wait on x3, x4;
end process;
```

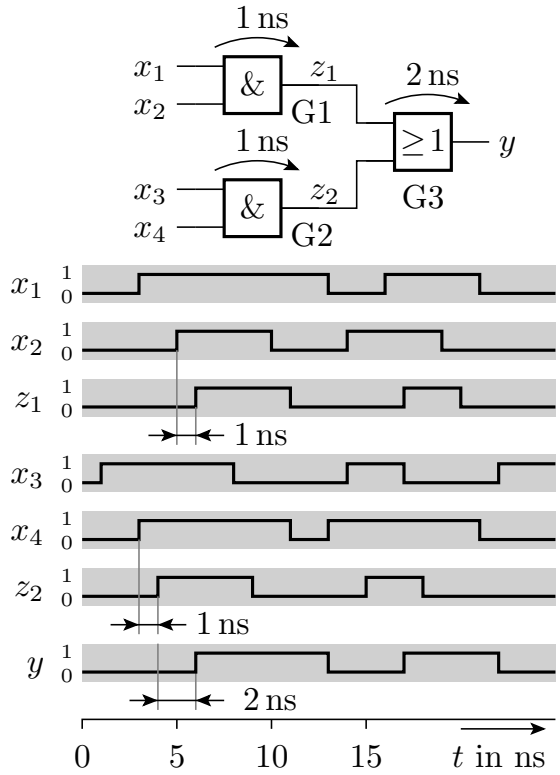



```
--- Simulation G3
G3: process
 begin
  y<=z1 or z2 after 2 ns;
  wait on z1, z2;
 end process;
--- Eingabeprozess
Input_Process: process
 begin
  wait for 1 ns; x3 <= '1';
  wait for 2 ns; x1 <= '1'; x4 <= '1';
    ...
  wait;
 end process;
end architecture;
```

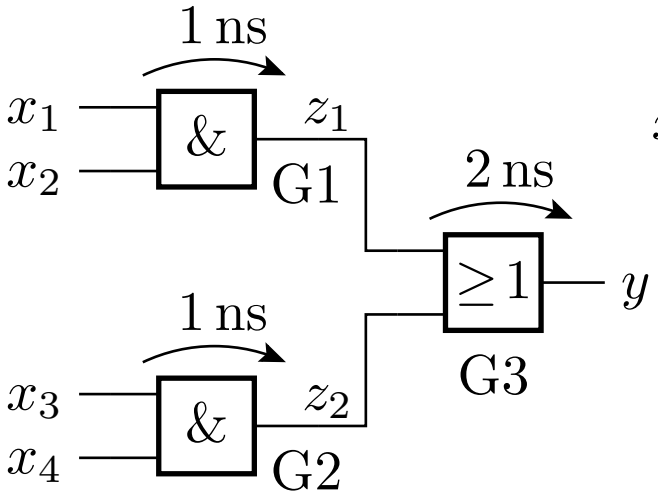
- Simulation of the example

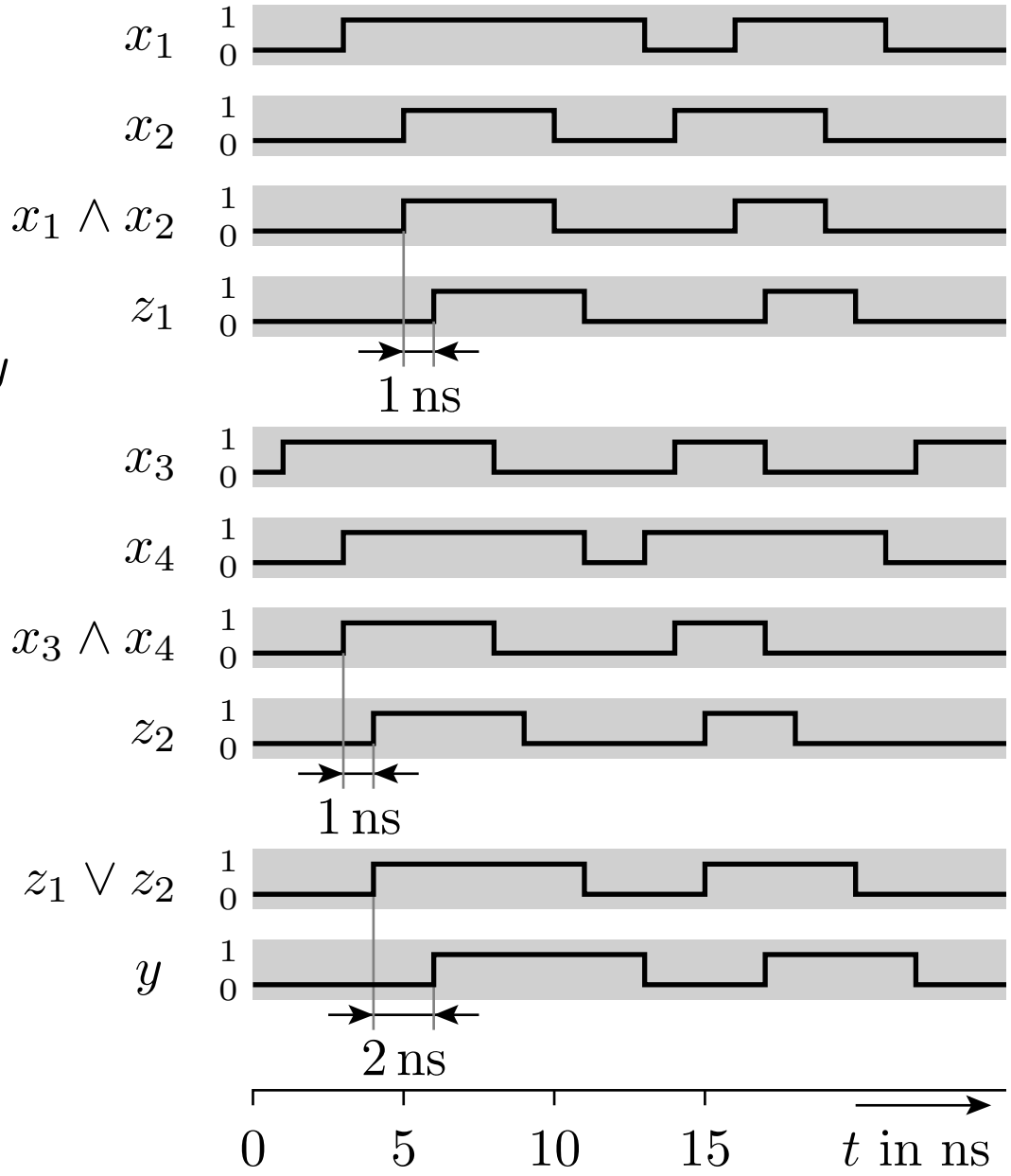
t_{sim}	process	signal assignment	scheduled transactions	signal value $x_4\ x_3\ x_2\ x_1\ z_2\ z_1\ y$
0 ns	G1	$z_1 \Leftarrow 0 \wedge 0$ nach 1 ns	———	0 0 0 0 0 0 0
0 ns	G2	$z_2 \Leftarrow 0 \wedge 0$ nach 1 ns	———	0 0 0 0 0 0 0
0 ns	G3	$y \Leftarrow 0 \vee 0$ nach 2 ns	———	0 0 0 0 0 0 0
0 ns	Eingabe		@1 ns: E	0 0 0 0 0 0 0
1 ns	E	$x_3 \Leftarrow 1$	@1 ns: $x_3 \to 1$ + @3 ns: E	0 1 0 0 0 0 0
1 ns	G2	$z_2 \Leftarrow 1 \wedge 0$ nach 1 ns	@3 ns: E	0 1 0 0 0 0 0
3 ns	E	$x_1 \Leftarrow 1$	@3 ns: $x_1 \to 1$	0 1 0 1 0 0 0
		$x_4 \Leftarrow 1$	@3 ns: $x_1 \to 1$ + @3 ns: $x_4 \to 1$	1 1 0 1 0 0 0
3 ns	G1	$z_1 \Leftarrow 1 \wedge 0$ nach 1 ns	@3 ns: $x_4 \to 1$	1 1 0 1 0 0 0
3 ns	G2	$z_2 \Leftarrow 1 \wedge 1$ nach 1 ns	@4 ns: $z_2 \to 1$	1 1 0 1 0 0 0
4 ns	G3	$y \Leftarrow 0 \vee 1$ nach 2 ns	@6 ns: $y \to 1$	1 1 0 1 1 0 0
6 ns	···	···	···	1 1 0 1 1 0 1

initialisation — no sheduled transaction E input process



- simulation result





Abbreviated form

- Process with only one wait statement on signal transitions

```
G1: process
begin
 z1<=x1 and x2 after 1 ns;
 wait on x1, x2;
end process;
```

⇒ abbriviated form: process with senstitivity list

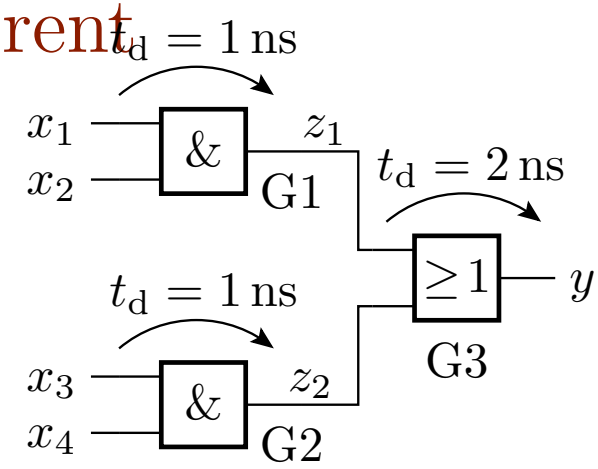
```
G1kurz: process (x1, x2)
begin
 z1<=x1 and x2 after 1 ns;
end process;
```

- all input signals in the sensitive list
  ⇒combinatorial process (process to describe a combinatorial circuit)
- combinatorial process with only one signal assignment
  ⇒ abbriviated notation

```
z1<=x1 and x2 after 1 ns;
```

- the abbriviated notation avoids
  - error »missing wait statement«
  - error »forgotten signal in the sensitivity list«
  - What effect have those errors? Why difficuilt to detect?

Example description with concurrent signal assignments



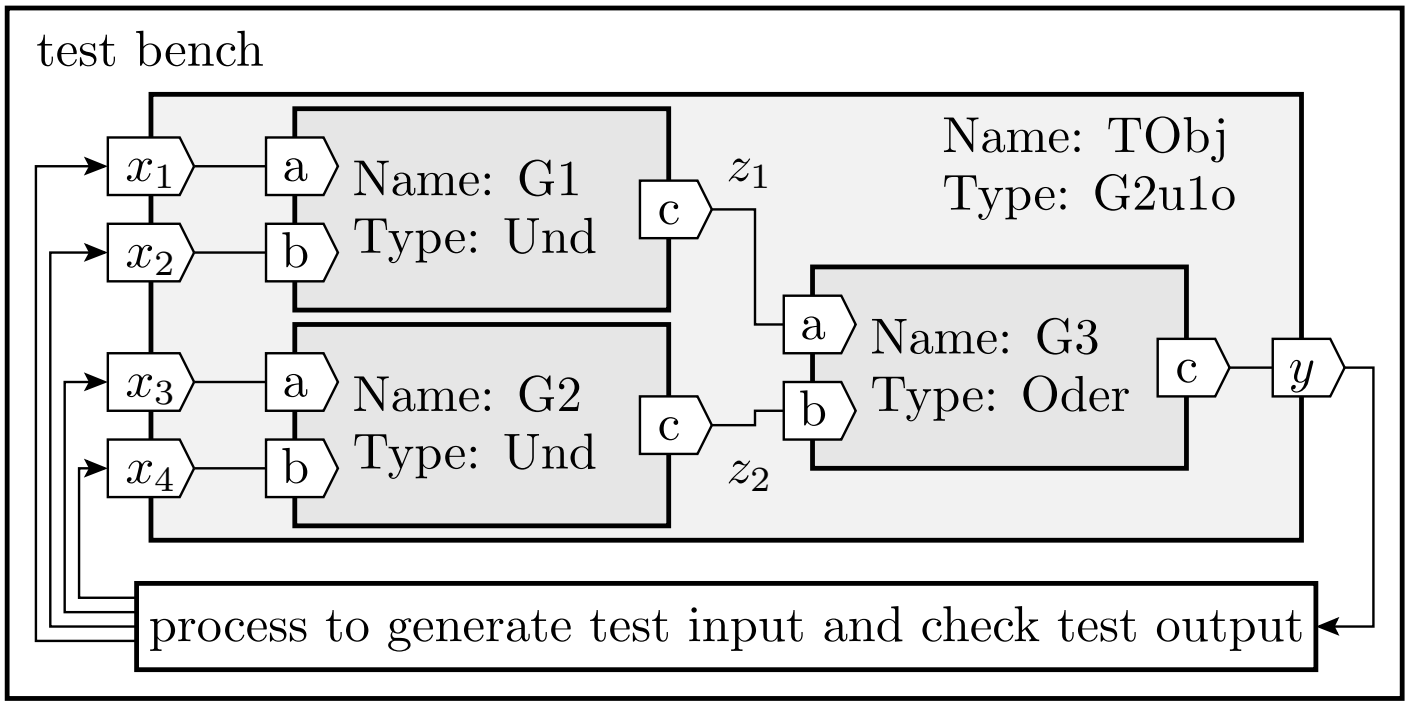

```
entity SimModel is
end entity;

architecture ConcSigAssign SimModel is
 signal x1, x2, x3, x4, z1, z2, y: STD_LOGIC := '0';

begin
--- Schaltung
 G1:z1<=x1 and x2 after 1 ns;
 G2:z2<=x3 and x4 after 1 ns;
 G3: y<=z1 or z2 after 2 ns;
--- Eingabeprozess
 Input_Process: process
 begin
  wait for 1 ns; x3<='1';
  wait for 2 ns;
  x1<='1'; x4<='1';
  wait;
 end process;
end architecture;
```

Summary

- The temporal flow is modeled by signal assigments and wait statements
- The functional model of a combinatorial circuit ist a process, which calculates the values of the output signals out of the values of the input signals, assigns them delayed, sleeps till the next transition of an input signal and repeats the all after waking up.
- one process per subcircuit + one process to provide the input signals + ...

Structural description

Structural description



- graphical presentation: data flow plan
- representation in the computer: list of subciruits, in witch to each pin a signal is assigned
- testbench: often a structural description with a device under test and test circuitry

Entity declaration

```
entity circuit is
  [generic (liste_of_configuration_parameters);]
  [port (pin_list)];
 end entity;
```

- pin list: comma-separated list of signals + flow direction + data type:

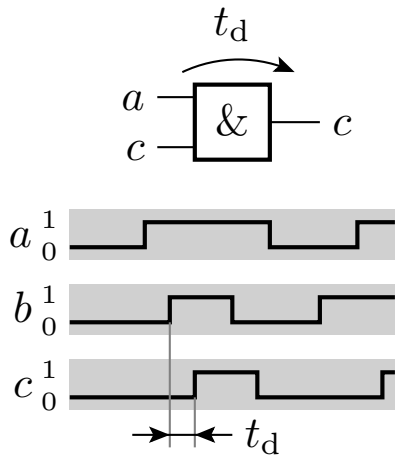
  signal −,*signal*″:*direction data_type*
  [:=*default_value*]

- signal flow direction:

`in`	`Input`
`out`	`Output`
`inout`	`In- and Output`
`buffer`	`Output with value can be read back`

- default value: initial value, if no other value is assigned

Description of a single AND-Gate



--- Schnittstellenvereinbarung

```
entity And2 is
 generic(td: DELAY_LENGTH:=0 ns);
 port(a, b: in STD_LOGIC;
        c: out STD_LOGIC);
end entity;
```

--- Beschreibungsversion mit Verzögerungszeit

```
architecture with_td of And2 is
begin
  c <= a and b after td;
end architecture;
```

--- Beschreibungsversion ohne Verzögerung

```
architecture without_td of And2 is
begin
  c <= a and b;
end architecture;
```

Subcircuit instance

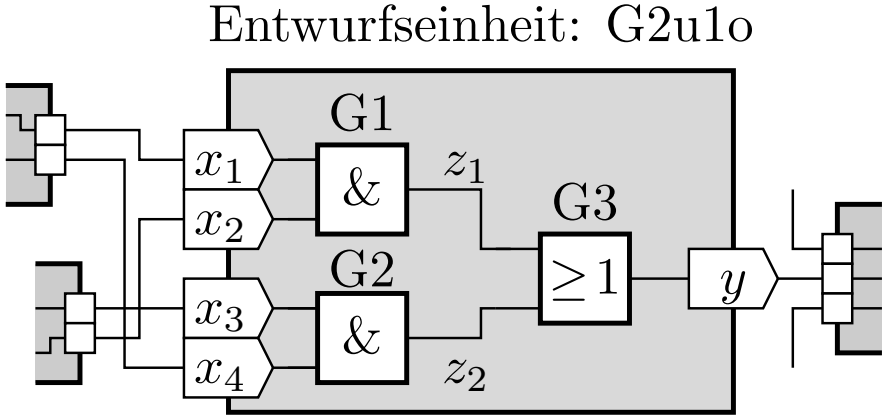
Syntax for a subcircuit entity:

[subcircuit_name:] entity [*Library*.]
entity_name [(*description_name*)]
[generic map(*allocation_map_parameters*)]
port map(*allocation_map_interface_signals*);

Allocation list:

- mapping by name:namensbasiert: comma-separated list of Tupels »*object_to_be_assigned* => *assigned_object*«
- mapping by position: comma-separated list of the assigned objects

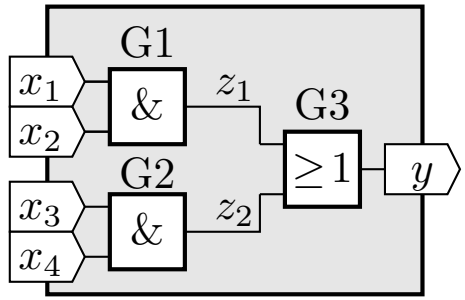
Struktural description of the 3-gate-circuit



Schnittstellenvereinbarung:

```
entity G3E is
 port(x1, x2, x3, x4: in STD_LOGIC;
        y: out STD_LOGIC);
end entity;

architecture Struktur of G3E is
--- Vereinbarung der interenen Signale
 signal z1, z2: STD_LOGIC;
begin
--- Instanziierung und Verbindung der Gatter
 ...
end architecture;
```

```
--- Instanziierung und Verbindung der Gatter
G1: entity WORK.Und2(mit_td)
 generic map(1 ns)(1)
 port map(x1, x2, z1)(1);

G2: entity WORK.Und2(mit_td)
 generic map(td => 1 ns)(2)
 port map(a=>x3, b=>x4, c=>z2)(2);

G3: entity WORK.Oder2(mit_td)
 generic map(td => 2 ns)(2)
 port map(a=>z1, b=>z2, c=>y)(2);
end architecture;
```



⇒WEB-Projekt: G3/G3_Struk

---

(1) positionsbasierte Zuordnung (kurz, fehlerträchtig)
(2) namensbasierte Zuordnung (länger, verständlicher)

Instanzbildung mit Platzhaltern

- Der Design-Wizard von ise empfiehlt die ältere, umständlichere Art der Instanziierung:

  [Teilschaltungsname:] [component] *Entwurfseinheit*
  [generic map(*Zuordnungsliste*_Parameter)]
  port map(*Zuordnungsliste_Anschlüsse*);

- erfordert eine zusätzliche Deklaration der Schnittstelle als Komponente (in einem Package oder im Deklarationsteil der Entwurfseinheit)

  component *Entwurfseinheit* is
  [generic (*Liste_der_Konfigurationsparameter*);]
  [port (*Anschlussliste*)];
  end [component][*Entwurfseinheit*];

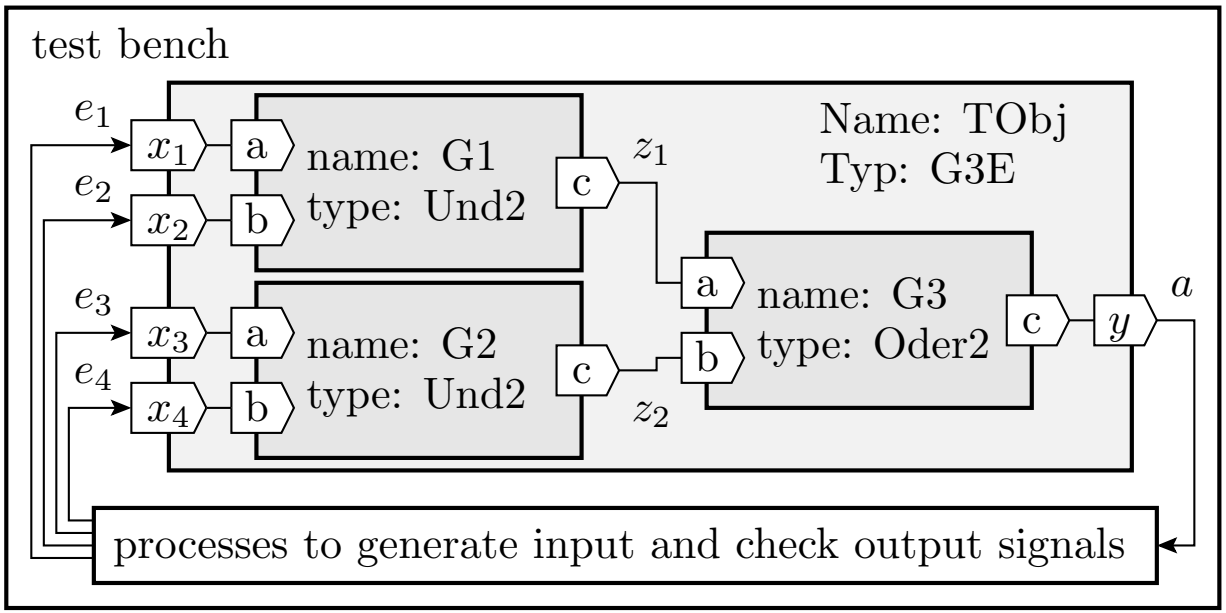
(Kopie der Schnittstellenbeschreibung und Ersatz von »entity« durch »component«)

Zuordnung von Entwurfseinheiten und Entwurfsbeschreibungen zu Komponenteninstanzen:

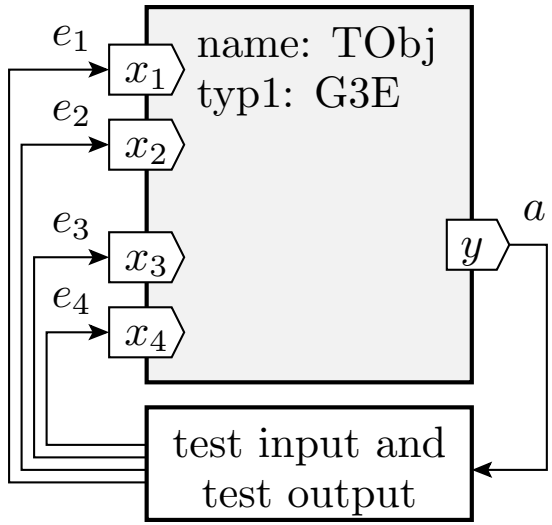
- implizit: gleichnamige Entwurfseinheit mit derselben Schnittstelle, die zuletzt analysiert wurde mit der zuletzt analysierten Beschreibung
- explizit: Zuordnung in einer zusätzlichen Konfigurationsbeschreibung (wird in der Vorlesung nicht verwendet)

Testrahmen

Testrahmen: oberste Hierarchieebene; keine Anschlüsse; Testobjekt als Instanz; zusätzliche Prozesse zur Testdurchführung



```
entity Test_G3E is
end entity;

architecture a of Test_G3E is
 signal e1,e2,e3,e4,a: STD_LOGIC;
begin
 TObj: entity WORK.G3E(Struktur)
  port map(x1=>e1, x2=>e2, x3=>e3,
           x4=>e4, y=>a);

Eingabe: process
  begin
   wait for 1 ns;  e3<='1';            --- 1 ns
   wait for 2 ns;  e1<='1'; e4<='1'; --- 3 ns
   ...
  wait;
 end process;
```



⇒WEB-Projekt: G3/Test_G3E

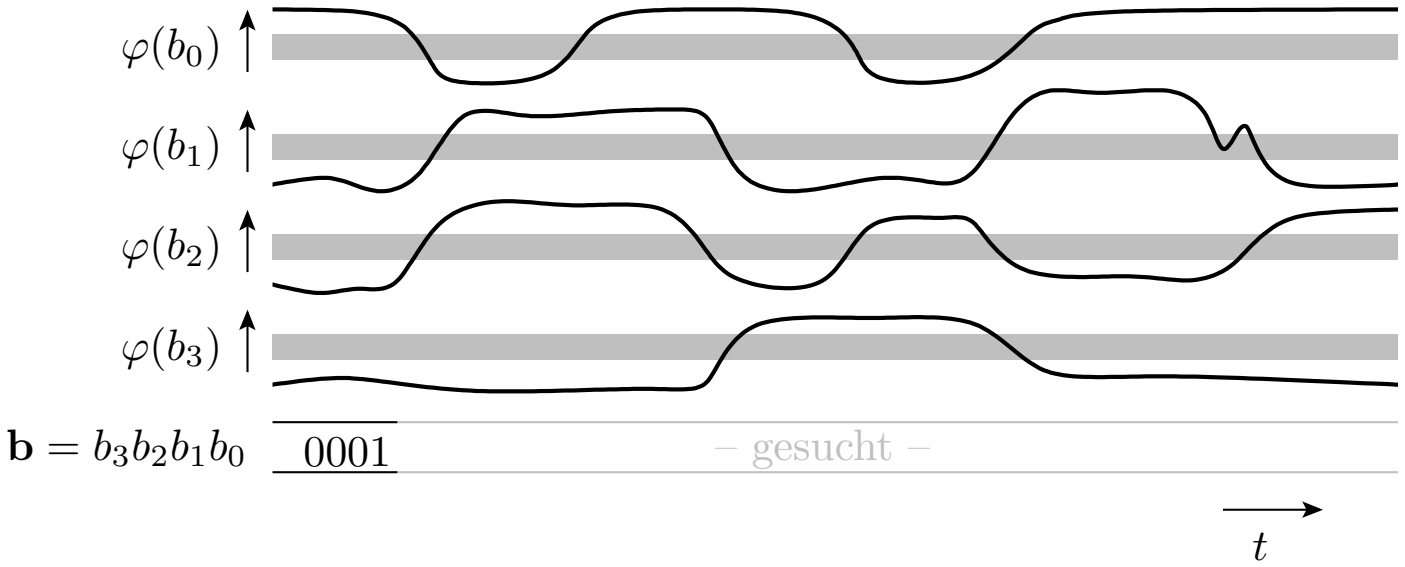
Analyse mit »ghdl -a Test_G3E.vhdl« löst Hierarchie auf.
Ergebnis 4-Prozess-Simulationsmodell wie bereits behandelt.

Zusammenfassung

- Zur hierarchischen Strukturierung werden die Prozesse zu Entwurfseinheit zusammengefasst, mit Schnittstellen versehen und in übergeordnete Entwurfseinheiten als Instanzen eingebunden.
- Die oberste Ebene in der Beschreibungshierarchie eines Simulationsmodells ist der Testrahmen, eine Entwurfseinheit ohne Anschlüsse.
- Bei der Schaltungsanalyse vor der Simulation wird die Hierarchie wieder in einzelne über Signale kommunizierende Prozesse – das eigentliche Simulationsmodell – aufgelöst.

Aufgaben

Aufgabe 1.2: Signalverlauf



- In welchen Zeitbereichen ist der Signalvektor ungültig?
- Welchen Wert hat er in den Gültigkeitsfenstern?

Aufgabe 1.3: Typen und Datenobjekte

Vereinbaren Sie folgende Datenobjekte:

- ein Signal mit dem Typ INTEGER und dem Anfangswert »30«
- eine Konstante vom Typ BOOLEAN und dem Anfangswert TRUE
- eine Bitvektorvariable vom STD_LOGIC_VECTOR, Indexbereich von 3 bis 0 absteigend und dem Anfangswert »alles null«.

Aufgabe 1.4: Ausdruck $\Leftrightarrow$ Berechnungsfluss $\Leftrightarrow$ Signalfluss

$$\overline{x_1 \wedge x_2 \wedge \bar{x}_3} \wedge \overline{\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3}$$

- Zeichnen des Berechnungsbaums unter Verwendung der ein- und zweistelligen logischen Operationen Invertierung, UND und ODER.
- Zeichnen Sie den Signalflussplan unter Verwendung von Invertern sowie UND- und NAND-Gattern mit n Eingängen.

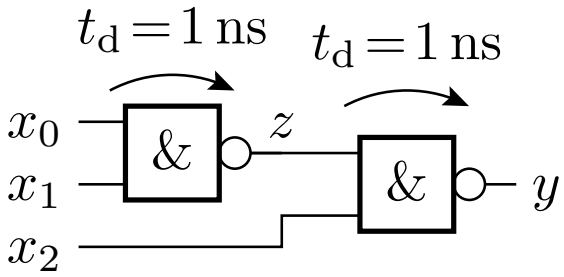
Aufgabe 1.5: Signale und Variablen

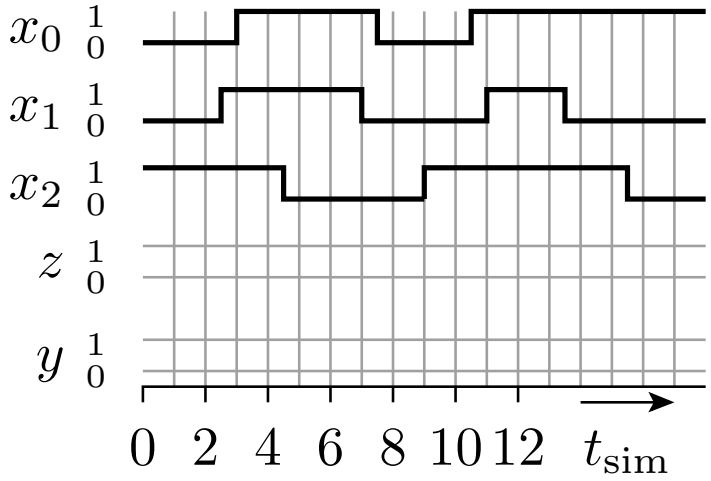
Welche Werte schreiben die einzelnen Write-Anweisungen?

```
--- Vereinbarungsteil der Entwurfseinheit
  signal a: INTEGER := 0;
--- Vereinbarungsteil des Prozesses
  variable b: INTEGER := 0;
--- Anweisungsteil des Prozesses
  A1: a <= a +1 after 0.5 ns; b := b+1; *
  A2: wait for 1 ns; *
  A3: a <= a +1 after 0.5 ns; b := b+1; *
  A4: a <= a +1 after 0.5 ns; b := b+1; *
  A5: wait for 1 ns; *
```

```
* write("a=" & str(a) & " b=" & str(b));
```

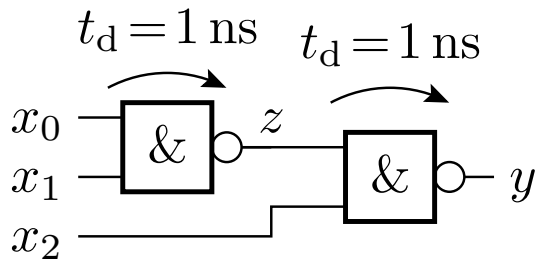
Aufgabe 1.6: Bestimmung der Signalverläufe von z und y

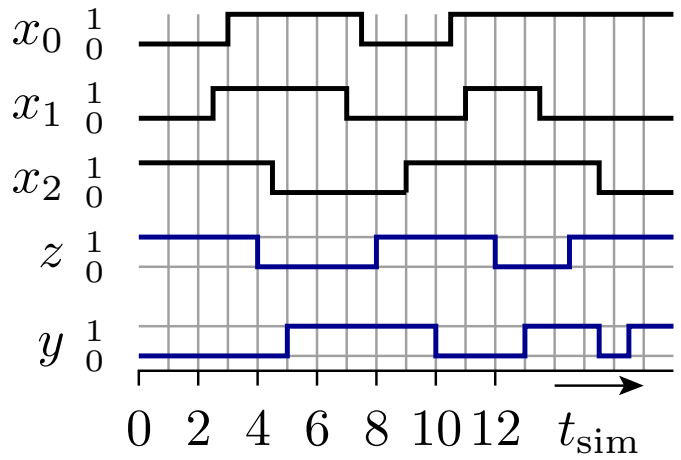




- Tabelle mit allen Änderungen
- Zeitverläufe zeichnen

Zeit	x_2	x_1	x_0	z	y
0,0 ns	1	0	0	1	0
2,5 ns		1			
3,0 ns			1		
4,0 ns				0	
4,5 ns	0				
5,0 ns					1
7,0 ns		0			
7,5 ns			0		
8.0 ns				1	
9,0 ns	1				
10,0 ns					0
10,5 ns			1		
11,0 ns		1			
12,0 ns				0	
13,0 ns					1
13,5 ns		0			
14,5 ns				1	
15,5 ns	0				0
16,5 ns					1

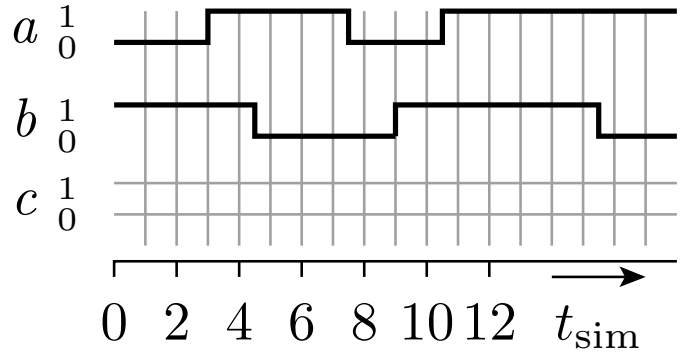




Änderungspfortpflanzung und Verzögerung um 1 ns

Aufgabe 1.7: Entwurfsfehler

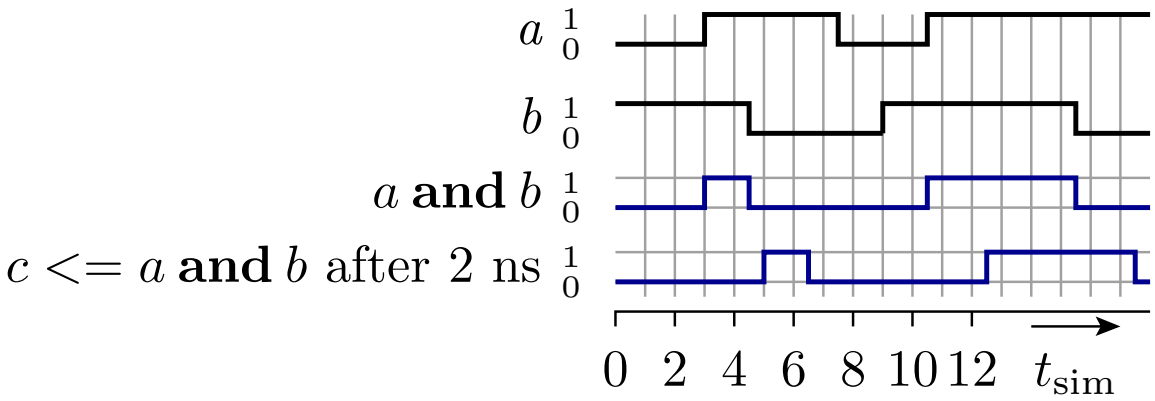
```
signal a, b, c: STD_LOGIC;
...
process
begin
 c <= a and b after 2 ns;
 wait on a, b;
end process;
```



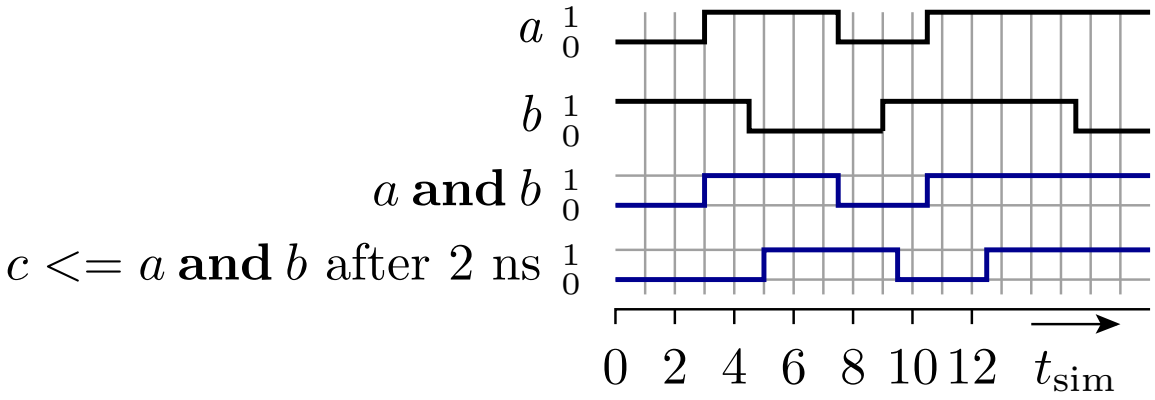
- Ausgabesignalverlauf des korrekten Simulationsmodells
- Ausgabesignal, wenn in der Warteanweisung das Signal b fehlt
- Was passiert, wenn die gesamte Warteanweisung fehlt? Warum ist der Fehler schwer zu lokalisieren?

Lösung

korrekte Beschreibung

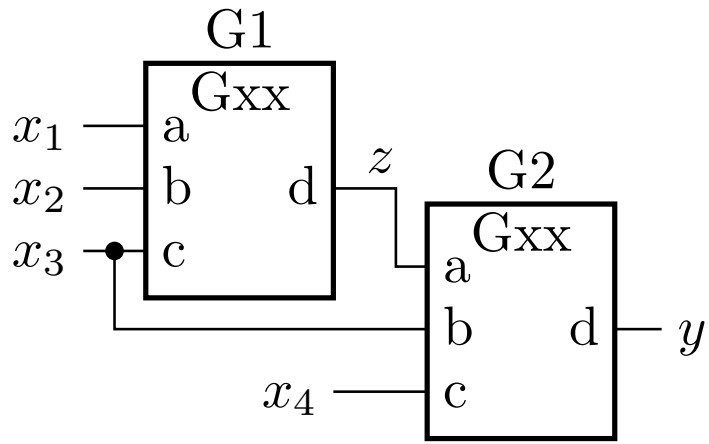


ohne Neuberechung bei Änderung von b



Aufgabe 1.8: Hierarchie auflösen, Testrahmen

Gegeben sind die Strukturbeschreibung der Gesamtschaltung und die Funktionsbeschreibung des Teilschaltungstyps.




```
entity Gxx is
  port(a, b, c: in STD_LOGIC; d: out STD_LOGIC);
end entity Gxx;
architecture Verhalten of Gxx is
begin
   d<=(a and not c) or (b and c) after 1 ns;
end architecture;
```

Gesucht:

- Strukturbeschreibung der Gesamtschaltung in VHDL
- Funktionsbeschreibung der Gesamtschaltung mit nebenläufigen Signalzuweisungen
- Testrahmen, der nacheinander im Abstand von 5 ns folgende Eingabewerte an $x_4 \ldots x_1$ anlegt: 0000, 0001, 0011, 0111