Rechnerorganisation Praktikum

Neunwertige Logik, Entity & Architecture

Philipp Habermann, Uffke Drechsler, Jonas Tröger

Architektur Eingebetteter Systeme
Institut für Technische Informatik und Mikroelektronik
Technische Universität Berlin

WS 2015/16

Gliederung

Einführung

- Basis: Aussagen, welche entweder *wahr* oder *falsch* sind.
- Verknüpfung der Aussagen zum Bilden weiterer Aussagen
- $\overline{A}$ Invertierung, $A \wedge B$ Und-Verknüpfung, $A \vee B$ Oder-Verknüpfung

Beispiel

Es ist nass, wenn ich nicht schwitze und meine Haut feucht ist.
Es regnet, wenn Wolken am Himmel sind und es nass ist.

$A =$ **Wolken am Himmel**
$B =$ **Ich schwitze**
$C =$ **Haut ist feucht**
$X =$ **Es ist nass**
$Y =$ **Es regnet**
$X = \overline{B} \wedge C$
$Y = A \wedge X$

Wahrheitstabellen

- Tabellarische Darstellung einer logischen Verknüpfung
- enthält den Wert der Aussage für **alle** Eingangskombinationen
- Statt *wahr* und *falsch* wird meist abkürzend 1 und 0 verwendet

Beispiel

A	B	C	X	Y
falsch	falsch	falsch	falsch	falsch
falsch	falsch	wahr	wahr	falsch
falsch	wahr	falsch	falsch	falsch
falsch	wahr	wahr	falsch	falsch
wahr	falsch	falsch	falsch	falsch
wahr	falsch	wahr	wahr	wahr
wahr	wahr	falsch	falsch	falsch
wahr	wahr	wahr	falsch	falsch

$A =$ **Wolken am Himmel**
$B =$ **Ich schwitze**
$C =$ **Haut ist feucht**
$X =$ **Es ist nass**
$Y =$ **Es regnet**
$X = \overline{B} \wedge C$
$Y = A \wedge X$

Wahrheitstabellen

- Tabellarische Darstellung einer logischen Verknüpfung
- enthält den Wert der Aussage für **alle** Eingangskombinationen
- Statt *wahr* und *falsch* wird meist abkürzend 1 und 0 verwendet

Beispiel

A	B	C	X	
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

$A =$ **Wolken am Himmel**

$B =$ **Ich schwitze**

$C =$ **Haut ist feucht**

$X =$ **Es ist nass**

$Y =$ **Es regnet**

$X = \overline{B} \wedge C$

$Y = A \wedge X$

Ermitteln einer Formel

- Herleitung der Formel aus Wertetabelle
- Der einfache Weg:
  - Betrachtung aller Eingänge für die Ausgabe *wahr* (1)
  - Aufstellen der Und-Verknüpfung für jede einzelne Zeile
    - Eingangs-Variable = 0 $\rightarrow$ Invertierung der Variable
  - Oder-Verknüpfung der Verknüpfungen der einzelnen Zeilen
    - Und-Verknüpfungen werden immer zuerst ausgewertet

Beispiel

A	B	Y	
0	0	1	$\rightarrow \overline{A} \wedge \overline{B}$
0	1	0	
1	0	1	$\rightarrow A \wedge \overline{B}$
1	1	1	$\rightarrow A \wedge B$

$$(\overline{A} \wedge \overline{B}) \vee (A \wedge \overline{B}) \vee (A \wedge B)$$

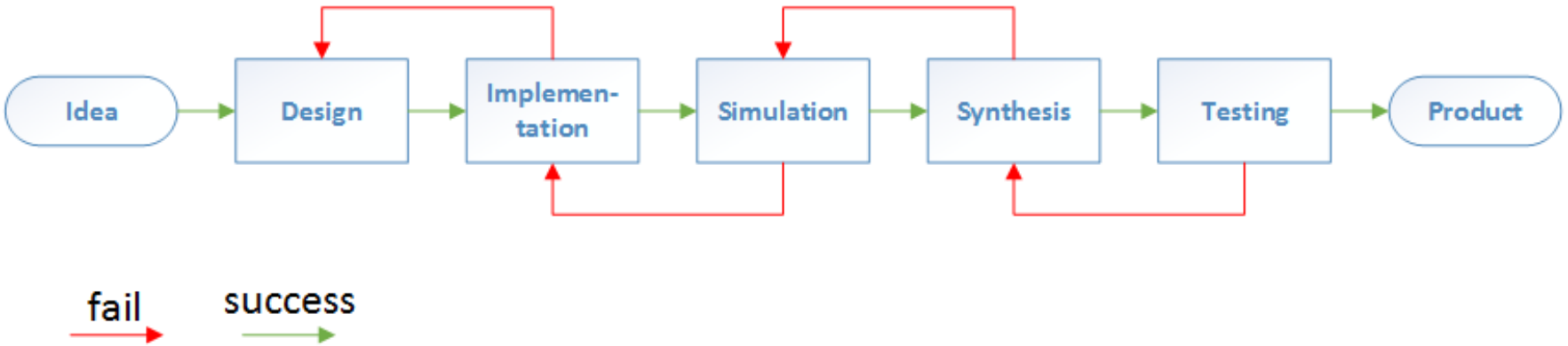
Hardwarebeschreibungssprachen

- **H**ardware **D**escription **L**anguage (kurz: **HDL**)
- Formale Sprache zur
  - Beschreibung
  - Optimierung
  - und zum Testen

  des Verhaltens integrierter Schaltungen

- **VHDL**: **V**HSIC **Hardware Description Language**
- **VHSIC**: **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit
- VHSIC: Projekt des US-Verteidigungsministeriums in den 80ern
- Seit 1987: IEEE standardisiert, erweitert 1993, 2002 und 2008
- Quasi-Standard in Europa, in Amerika ist *Verilog* sehr verbreitet

Entwicklungszyklen



Im Rahmen dieses Praktikums beschäftigen wir uns *ausschließlich* mit den Schritten der Implementation und Simulation.

Definition

Entity:

Definiert die *Schnittstellen* einer Zelle.

```
entity entity_name is
[generic (generic_list)] [port (port_list);]
entity_declarative_part
[begin
  passive_concurrent_statement]
end [entity] [entity_name];
```

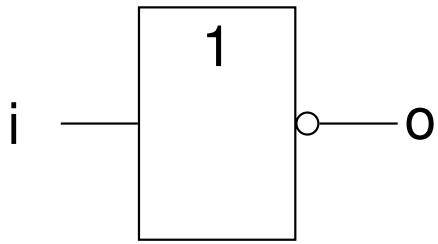
Architecture:

Definiert die *Funktionalität* einer Zelle.

```
architecture architecture_name of entity_name is
  architecture_declarative_part
begin
  all_concurrent_statements
end [architecture] [architecture_name];
```

Die Angabe der eckig geklammerten Parameter ist optional!

Am Beispiel: NOT-Gatter



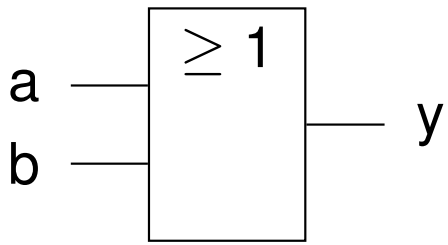
i	o
0	1
1	0

```
library ieee;
use ieee.std_logic_1164.all;

entity not1 is
    port(i : in std_logic;
         o : out std_logic);
end not1;

architecture behavioral of not1 is
begin
    o <= not i;
end behavioral;
```

Am Beispiel: OR-Gatter



a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

```
library ieee;
use ieee.std_logic_1164.all;

entity or2 is
    port(a : in std_logic;
         b : in std_logic;
         y : out std_logic);
end or2;

architecture behavioral of or2 is
begin
    y <= a or b;
end behavioral;
```

Einbindung in VHDL

- Die neunwertige Logik (auch *multivalued logic*, MVL9) ist kein Bestandteil von VHDL; deshalb muss die entsprechende IEEE-Bibliothek erst eingebunden werden:

```
library ieee;
use ieee.std_logic_1164.all;
```

- Sie enthält:
  - **Logiktypen**
    z.B.: `std_logic`, `std_logic_vector`, `std_ulogic`, `std_ulogic_vector`, ...
  - **logische Operatoren**
    z.B.: `and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not`
  - **Konvertierungsfunktionen zwischen Typen**
    z.B.: `to_bit`, `to_StdLogicVector`, ...

Definition - Teil 1

Um elektrisches Verhalten exakter nachbilden zu können, arbeitet die neunwertige Logik nicht nur mit den beiden Werten `0` und `1`:

`0`	*strong low*	Treiberausgang definiert
`1`	*strong high*	Treiberausgang definiert
`L`	*weak low*	Pull-down
`H`	*weak high*	Pull-up
`X`	*strong unknown*	Treiberausgang undefiniert
`W`	*weak unknown*	Bus-Keeper uninitialisiert
`Z`	*high impedance*	Tri-State
`-`	*don't care*	Pegel bedeutungslos
`U`	*uninitialized*	FF-Ausgang uninitialisiert

Definition - Teil 2

Durch die unten tabellarisch dargestellte Auflösungsfunktion sind mehrere Signaltreiber für ein und das selbe Signal möglich:

U	X	0	1	Z	W	L	H	-	
U	U	U	U	U	U	U	U	U	U
U	X	X	X	X	X	X	X	X	X
U	X	0	X	0	0	0	0	X	0
U	X	X	1	1	1	1	1	X	1
U	X	0	1	Z	W	L	H	X	Z
U	X	0	1	W	W	W	W	X	W
U	X	0	1	L	W	L	W	X	L
U	X	0	1	H	W	W	H	X	H
U	X	X	X	X	X	X	X	X	-

Fehler-Erkennung

Ein genaues Verständnis der neunwertigen Logik ermöglicht uns zudem eine bessere Fehler-Erkennung während der Simulation:

```
library ieee;
use ieee.std_logic_1164.all;

entity dummy is
  port ( i1, i2 : in std_logic;
         o1, o2 : out std_logic );
end entity;

architecture dataflow of dummy is
begin
  o1 <= i1;
  o1 <= i2;
end architecture;
```