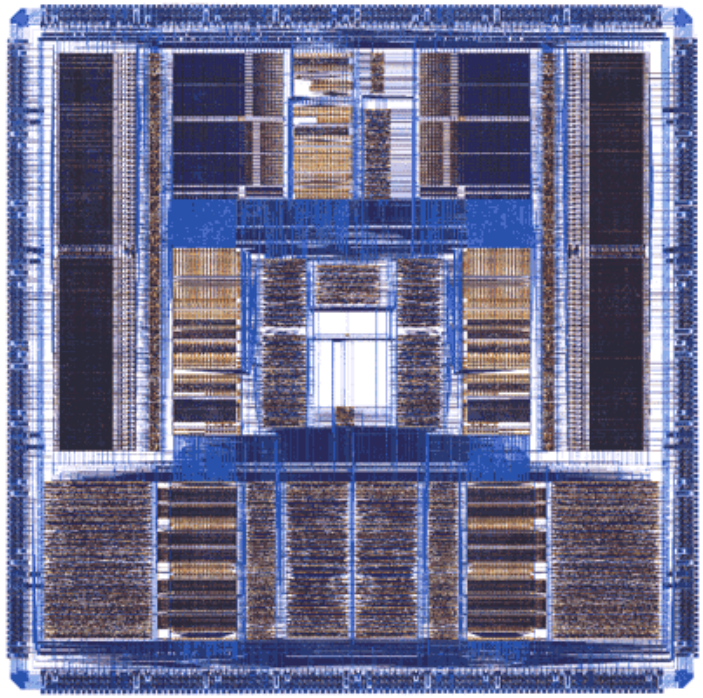


Einführung in VHDL

Dipl.-Ing. Franz Wolf

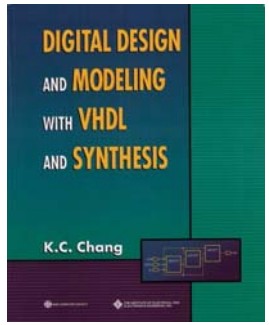
Literatur

Digital Design and Modeling with VHDL and Synthesis

Kou-Chuan Chang

Wiley-IEEE Computer Society Press

ISBN 0818677163



Rechnergestützter Entwurf digitaler Schaltungen

Günter Jorke

Hanser Fachbuchverlag

ISBN: 3446228969

VHDL-Synthese

Jürgen Reichhardt, Bernd Schwarz

Oldenbourg

ISBN: 3486273841

Entwicklung von VHDL

Very High Speed Integrated Circuit
Hardware
Description
Language

VHDL ist Beschreibungssprache für **digitale** Hardware

ab 1980 VHSIC-Projekt im US-DoD (Amerikanisches Department of Defence)
: Entwicklung einer Standard-HDL (VHDL)
1983 IBM, Texas Instruments und Intermetrics nehmen teil

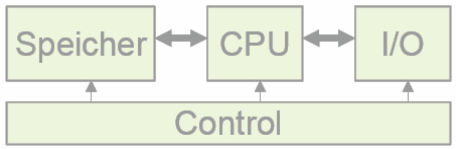
1987 IEEE setzt Industriestandard 1076 fest

1988 DoD legt fest:Alle elektronischen Schaltungen müssen in VHDL beschrieben werden.

1993 IEEE 1076 wird überarbeitet

Abstraktionsebenen beim Entwurf digitaler Systeme

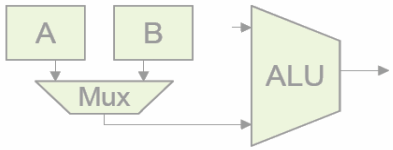
- Architektur
- Funktional
- Register-Transfer
- Logik
- Transistor
- Physikalische



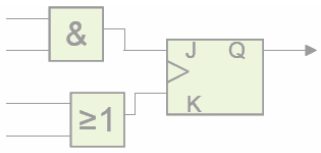
Keine Aussagen über Signale, Zeitverhalten und funktionales Verhalten

```
A := 3*B + C
If P=False Then
   B := B-1
Else B := B+1
```

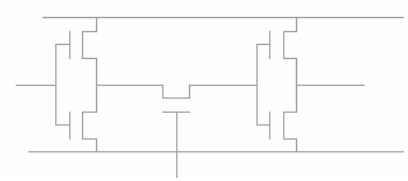
Beschreibung des Systems **durch Algorithmen wie Funktionen, Prozeduren und Prozesse**

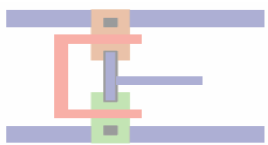


Beschreibung einer Schaltung durch **Operationen (z.B. Addition)** und durch den **Transfer der verarbeiteten Daten zwischen Registern**



Beschreibung durch logische Verknüpfungen und deren **zeitlichen Verhalten** (z.B. Verzögerungszeiten) Signale nehmen nur bestimmte, **vordefinierte Logikwerte an** (high, low, ...)



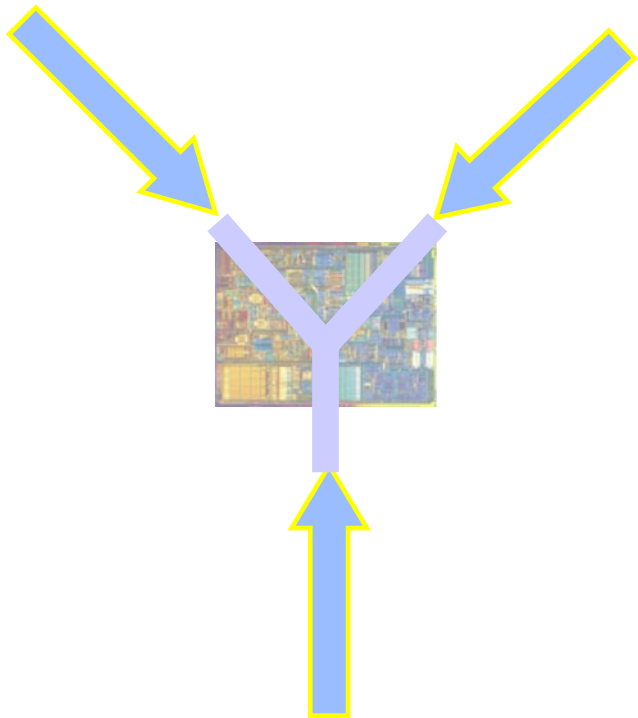


Schaltkreisebenen für ASICs notwendig aber für FPGAs irrelevant (vorgegeben)!
Einzelne **Module** werden nicht durch eine logische Funktion mit Verzögerungszeiten beschrieben, sondern **durch ihren Aufbau aus Bauelementen** wie Transistoren, Widerstände
und Kapazitäten; und die einzelnen **Bauelemente** werden durch ihre **Geometrie** bestimmt

Y-Diagramm

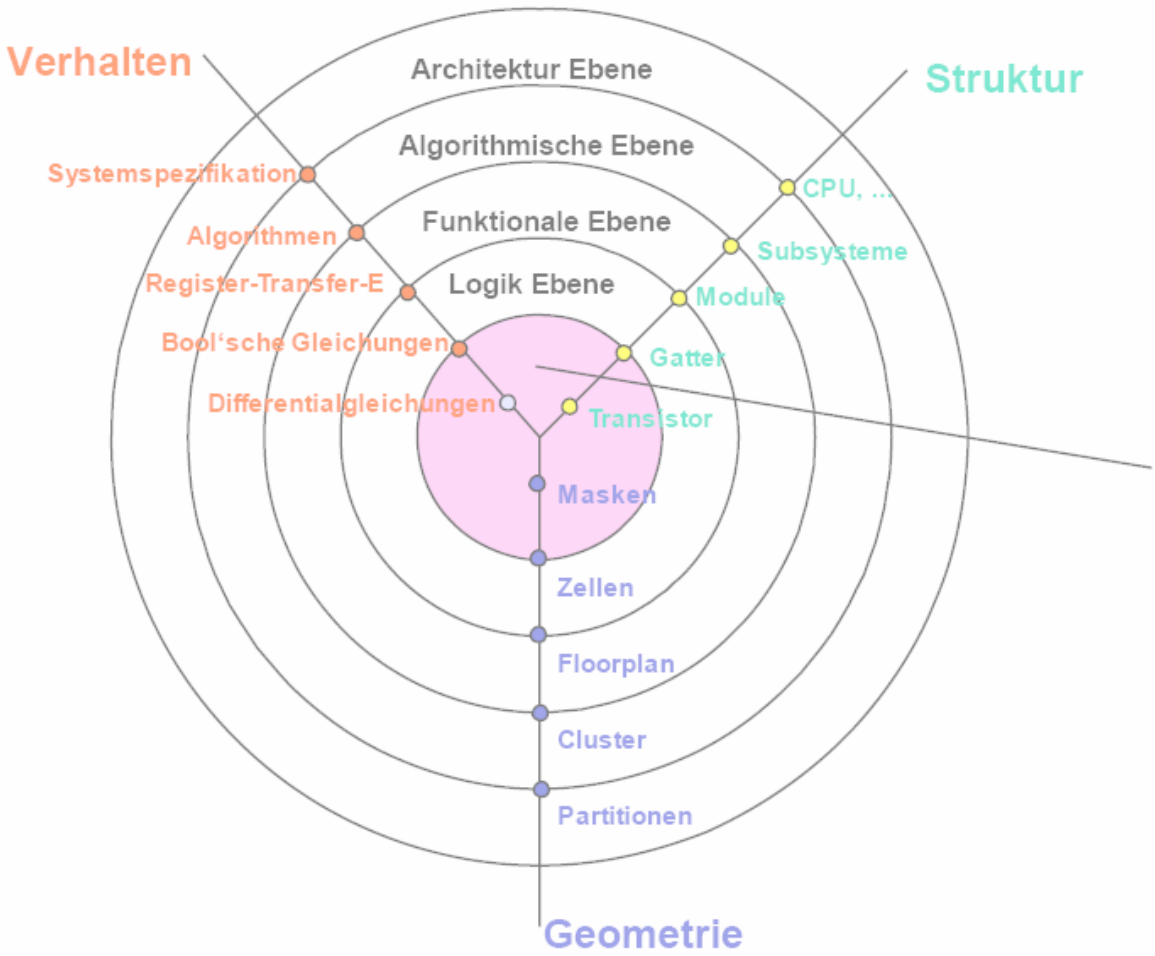
Wie verhält sich der Chip ?
Verhalten

Was ist im Chip ?
Struktur



Wo sind die Komponenten platziert ?
Geometrie

Entwurfsschichten



3 Sichtweisen:

- Verhalten
- Struktur
- Geometrie

Großer Radius
⇔ hohe Abstraktion

Schaltkreisebene
nicht für FPGAs

Y-Diagramm nach Gaisky-Kuhn

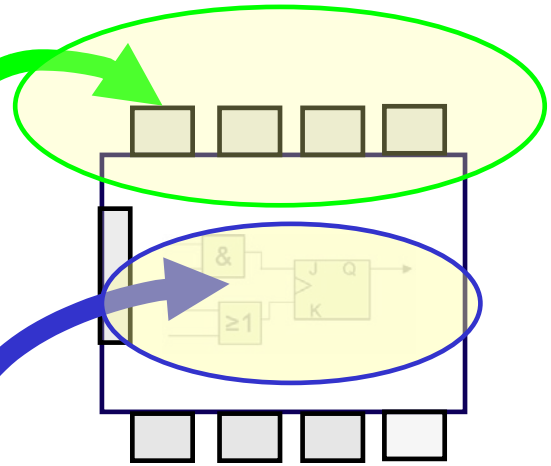
Aufbau einer VHDL Beschreibung

Package
Bereitstellen von Bibliotheken

Entity
Schnittstellenbeschreibung

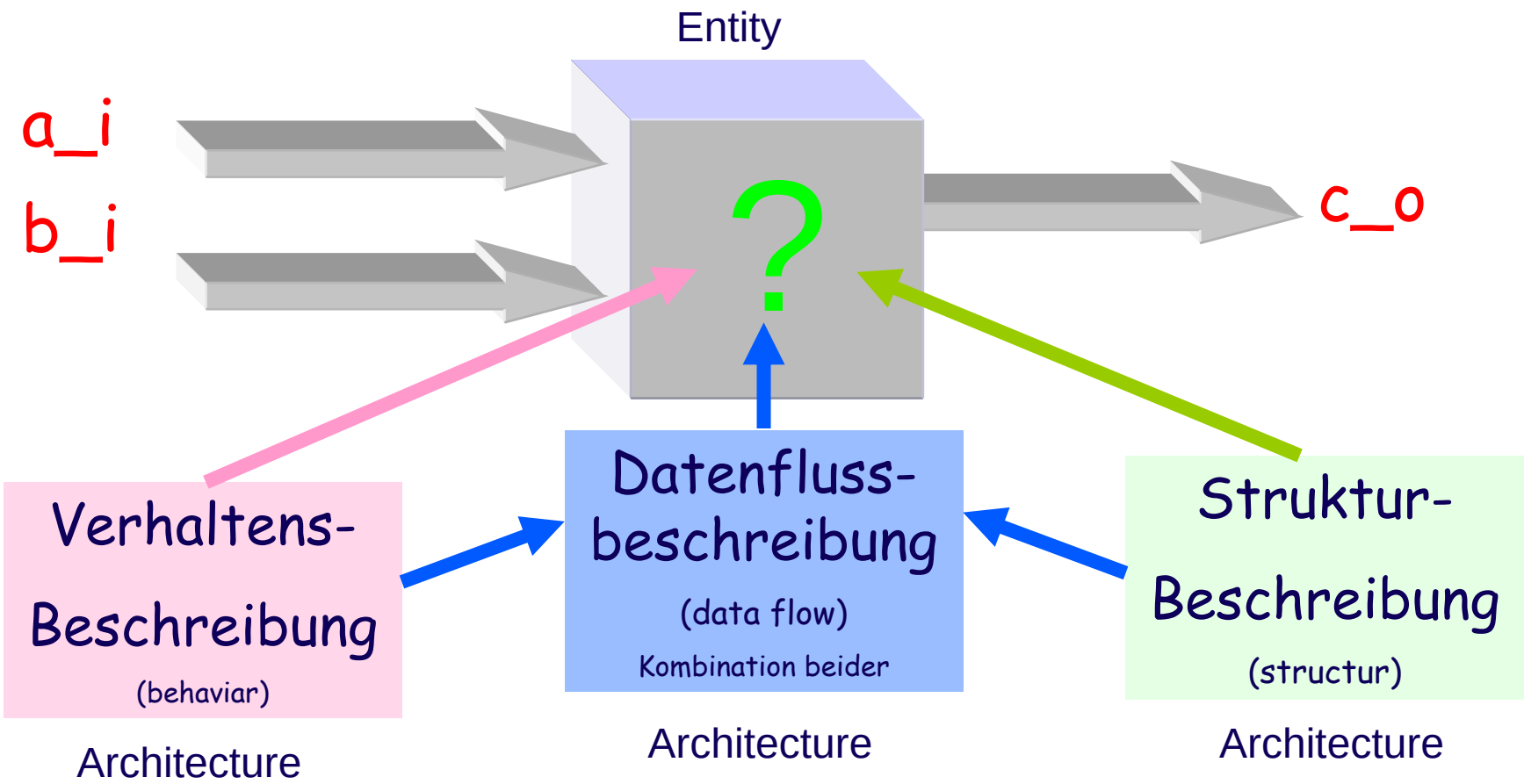
Architecture
Implementierung der Entity

Configuration
Zuordnung zwischen
Entity und Architeckure



Die Minimalkonfiguration eines VHDL-Designs besteht aus einer ENTITY und der eigentlichen Designeinheit, der ARCHITECTURE .

Aufbau einer VHDL Beschreibung



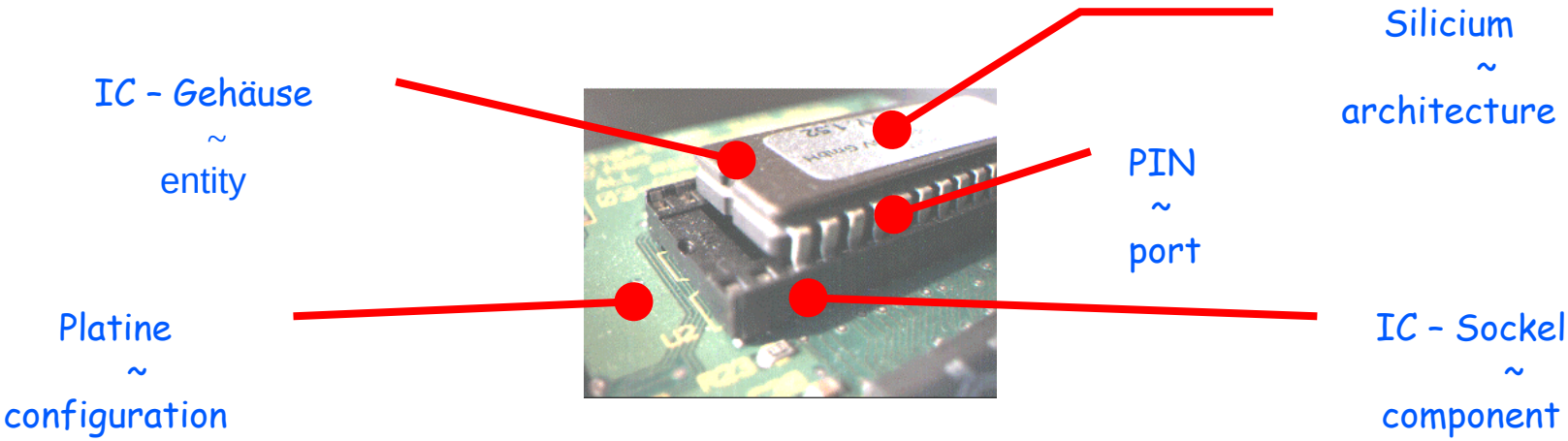
Anweisungen:

☑sequentielle ("sequential statements") ⇨ kann ***nur innerhalb von Prozessen*** verwendet werden!

☑ nebenläufige ("concurrent statements") ⇨ • Signalzuweisung • mehrer Prozesse parallel usw.

Begriffe in der Elektronik ⇔Äquivalenz zu VHDL

Elektronik	VHDL
Silizium (als Technologie für den IC)	architecture
IC	entity
Pin	port
Norm, Bauteilkatalog	package
unbestückter IC-Sockel	component instantiation
Leiterbahn mit zugehörigen Lötaugen	signal
Bestückungsplan	configuration



Signal ⇔ Variable

Variablen

- Unterscheiden sich nicht von Variablen anderer höheren Programmiersprachen
- Variablen eignen sich zur Speicherung von Zwischenergebnissen
- **Eine Variable hat "kein Gegenstück in der Hardware"**

Signale

- Sind in anderen Programmiersprachen nicht zu finden.
- **Ein Signal kann als verbindende physikalische Leitung angesehen werden**

	Variable	Signal
Symbol	**:=**	**<=**
Deklaration	werden in Prozessen/Unterprogrammen	nur bei Architecture, Block, Entity
Gültigkeit	nur im Prozessen/Unterprogrammen auch nur dort sichtbar	bei erreichen des nächsten wait, vorher nur vorgemerkt, können jederzeit wieder überschrieben werden
Simulationszeitpunkt	mehrere Werte annehmen	nur einen Wert
Vergangenheit	hat keine	hat eine Wert in der Vergangenheit, diesen kann man abfragen

Beispiel: Halbaddierer

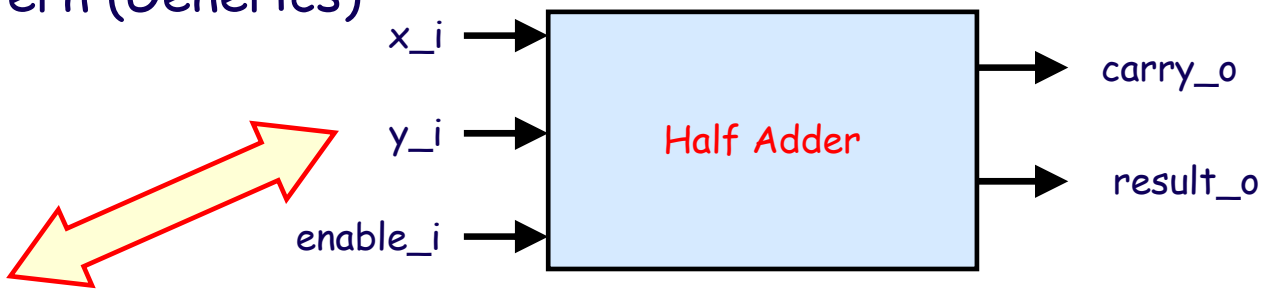
Spezifikation:

- Ein- und Ausgänge sind jeweils 1 Bit
- Wenn enable ist high ➔ Berechnung
- Wenn enable ist low ➔ Ausgänge low

Entity

Die Entity beschreibt die Schnittstelle der zu modellierenden Komponente

- Eingänge und Ausgänge werden deklariert
- Übergabe von Parametern (Generics)




```
ENTITY half_adder IS
  PORT(
        x_i, y_i, enable   : IN std_logic;
        carry_o, result_o: OUT std_logic
       );
END half_adder;
```

std_logic ist ein Datentyp eines ports. Standard logic ist definiert in IEEE 1164. Dieser Datentyp implementiert eine 9-wertige Logik :

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

Architecture - Datenflussbeschreibung

Die Architektur enthält in VHDL die Beschreibung der Funktionalität des Modells

- Verschiedene Varianten der Beschreibung sind möglich und können auch miteinander kombiniert werden
- Erste Variante:

Beschreibung der Architektur mit Hilfe von Logik-Gleichungen

```
ARCHITECTURE half_adder_b OF half_adder IS
        BEGIN
                carry_o <= enable_i AND (x_i AND y_i);
                result_o <= enable_i AND (x_i XOR y_i);
        END half_adder_b;
```

Das Modell kann simuliert werden, um die korrekte Funktion der Komponente zu verifizieren

Architecture-Verhaltensbeschreibung

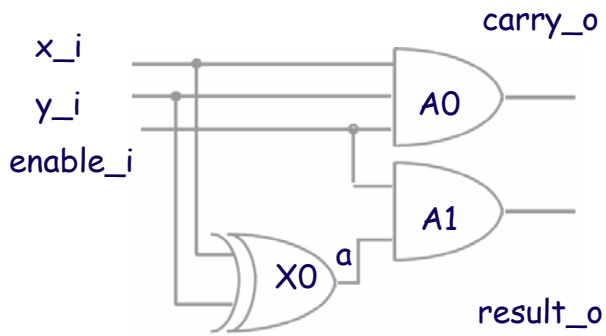
```
ARCHITECTURE half_adder_a OF half_adder IS
    BEGIN
        PROCESS (x_i, y_i, enable_i)
                BEGIN
                        IF enable_i = '1' THEN
                                result_o <= x_i XOR y_i;
                                carry_o <= x_i AND y_i;
                        ELSE
                                carry_o <= '0';
                                result_o <= '0';
                        END IF;
                END PROCESS;
END half_adder_a;
```

Sensitivity List

Eine abstrakte Verhaltensbeschreibung kann genutzt werden, um die Funktion des Addierers zu beschreiben

Architecture-Strukturelle Beschreibung

```
ARCHITECTURE half_adder_c OF half_adder IS

        COMPONENT and2
                PORT (in0, in1 : IN std_logic;
                        out0 : OUT std_logic  );
        END COMPONENT;

        COMPONENT and3
                PORT (in0, in1, in2 : IN std_logic;
                        out0 : OUT std_logic);
        END COMPONENT;

        COMPONENT xor2
                PORT (in0, in1 : IN std_logic;
                        out0 : OUT std_logic);
        END COMPONENT;

-- Deklaration interner Signale
SIGNAL a : std_logic; -- internal signal

BEGIN
        A0 : and2 PORT MAP (enable_i, a, result_o);
        A1 : and3 PORT MAP (x_i, y_i, enable_i, carry_o);
        X0 : xor2 PORT MAP (x_i, y_i, a);
END half_adder_c;
```



Strukturelle Beschreibung aus vordefinierten Komponenten, die zu verschaltenden Komponenten können einer Modulbibliothek entnommen werden

Process

Die process - Anweisung dient zur Beschreibung eines Funktionsblockes.
Sie kommt innerhalb der Architektur zum Einsatz, und eignet sich insbesondere zur Verhaltensbeschreibung.

- Sie wird selbst parallel zu anderen Funktionsblöcken abgearbeitet.
- Sie beschreibt die Funktion eines Baublocks nur mit sequentiellen Anweisungen.

```
process [(Sensitivitätsliste)]
    [ Prozeß-Deklarationsteil]
begin
    [ Prozeß-Anweisungsteil]
end;
```

Innerhalb eines Prozesses können die nebenstehende Anweisungen zur Steuerung der Abarbeitung des Algorithmusses genutzt werden.

if - Anweisung

```
if bedingung then
        Anweisungen
elsif bedingung then
        Anweisungen
else
        Anweisungen
end if;
```

case - Anweisung

```
case ausdruck is
        when auswahl => Anweisungen;
        when auswahl => Anweisungen;
        ...
end case;
```

loop - Anweisung

```
loop
        Anweisungen
end loop;
```

kombinierte loop - Anweisungen

```
while bedingung loop
        Anweisungen
end loop
```

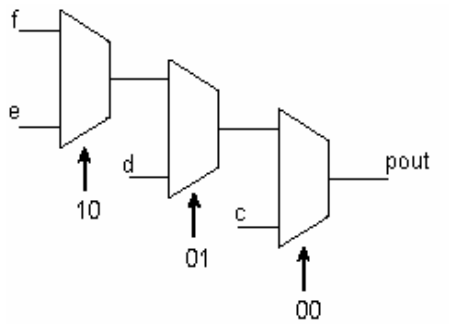
```
for variable in bereich loop
        Anweisungen
end loop;
```

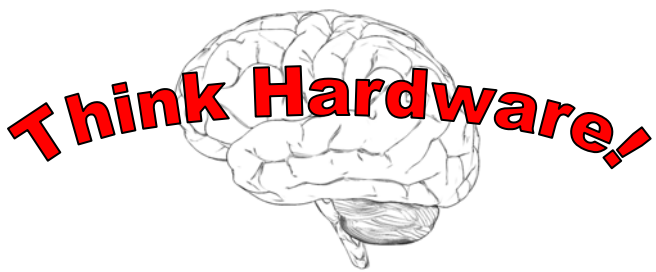
Multiplexer

```
myif_pro: process (s, c, d, e, f)
  begin
    if s = "00" then
            pout <= c;
    elsif s = "01" then
            pout <= d;
    elsif s = "10" then
            pout <= e;
    else
            pout <= f;
    end if;
end process myif_pro;
```

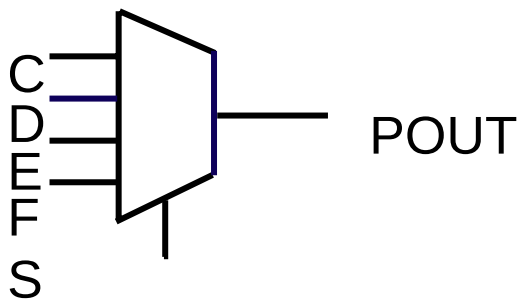
```
mycase_pro: process (s, c, d, e, f)
 begin
   case s is
            when "00" =>
                     pout <= c;
            when "01" =>
                     pout <= d;
            when "10" =>
                     pout <= e;
            when others =>
                      pout <= f;
   end case;
 end process mycase_pro;
```

Prioritätsdecoder:
ersten Bedingung höchste Priorität

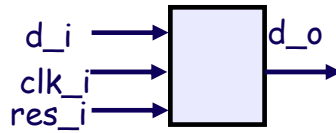




Keine Priorität !!!



Taktzustand- ⇔ Taktflankengesteuert



Latch

```
entity D_Latch_FF is
  port ( res_i : in std_logic;
           d_i : in std_logic;
         clk_i : in std_logic;
           d_o : out std_logic);
end D_Latch_FF;
```

D-Flip-Flop

```
architecture RTL of D_Latch_FF is
  begin
    REG: process (res_i, clk_i, d_i)
         begin
            if (res_i = '1') then
                d_o <=  '0';
            elsif (clk_i = '1') then
                d_o <= d_i;
            end if;
         end process REG;
end RTL;
```

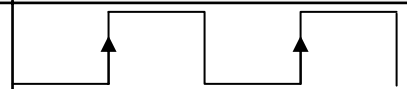
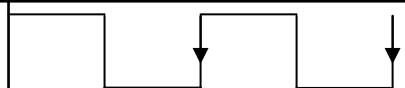
```
architecture RTL of D_Latch_FF is
  begin
    REG: process (res, clk_i)
       begin
          if (res_i = '1') then
                d_o <=  '0';
          elsif (clk_i'event and clk_i = '1') then
                d_o <= d_i;
          end if;
       end process REG;
end RTL;
```

Asynchroner Teil

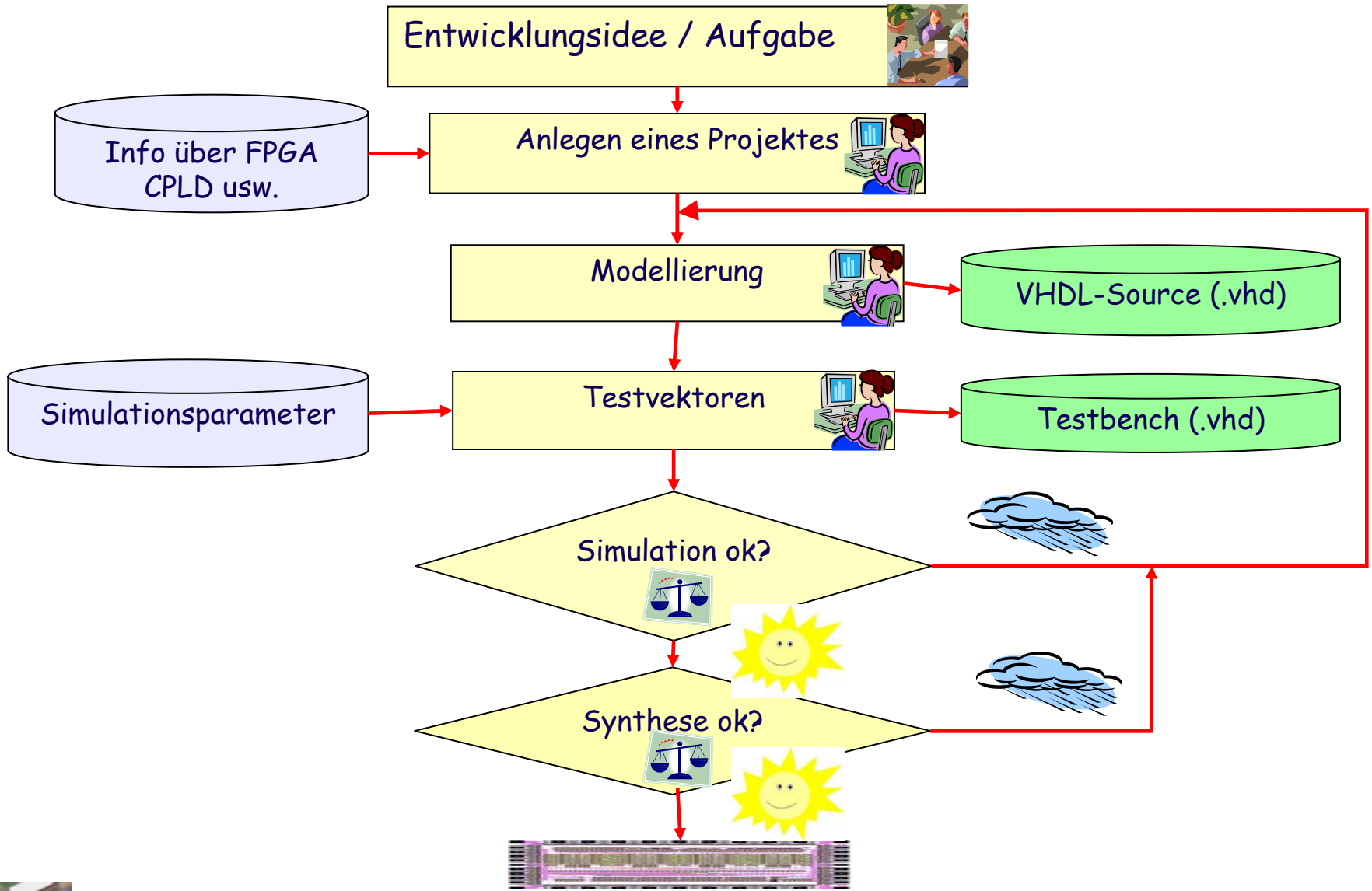
Synchroner Teil

Latches sind taktzustandsgesteuerte Datenspeicher, d.h. die Datenübernahme erfolgt während der Taktphase clk ='1'.

Register im engeren Sinn sind taktflankengesteuerte Datenspeicher. Bei ihnen erfolgt die Datenübernahme während der L/H- bzw. der H/L-Flanke des Taktes.

Syntax	Hardware
elsif (clk_i'**event** and clk_i = '1') **then** ….	
elsif (clk_i'**event** and clk_i = '0') **then** ….	

Designablauf

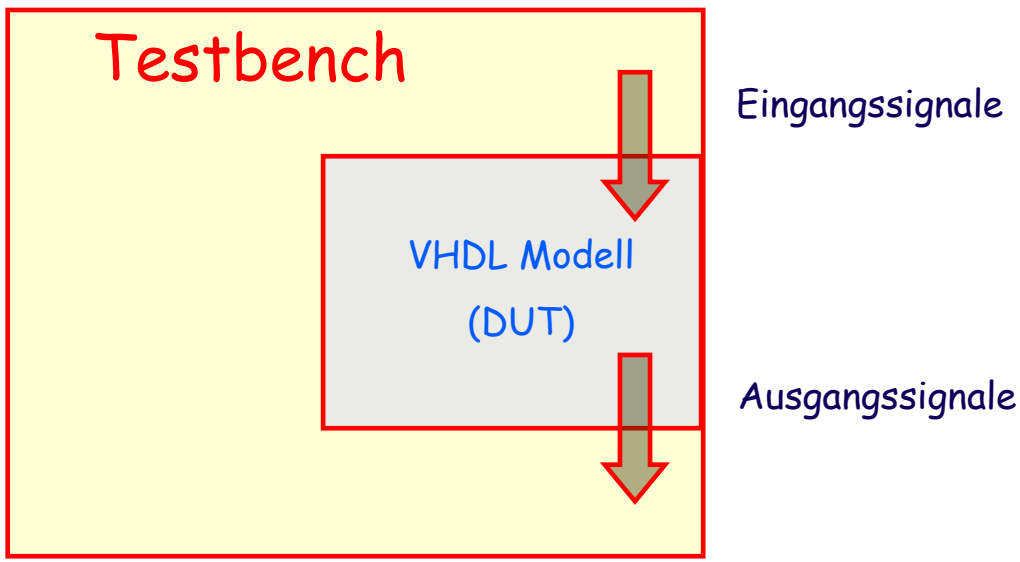


Testbench

Die Simulation dient i.a. der Verifikation von Entwurfsschritten. In VHDL erfolgt die Beschreibung der zu entwickelnden Systeme vorwiegend als Verhaltensmodell auf den oberen Entwurfsebenen (System-, Algorithmische und Register-Transfer-Ebene).

Ziel einer Simulation von Verhaltensmodellen ist

- Die Prüfung auf syntaktische Korrektheit
- Vergleich mit der vorgegebenen Spezifikation (Prüfung der Funktionalität)
- Bestimmung der Eigenschaften des Modells



Eine Testbench ist ein VHDL Programm, welches Stimuli für die zu testende Komponente erzeugt und in zeitlicher sinnvoller Abfolge die von den Komponente erzeugten Ausgangssignale prüft.

Die zu Testente Komponente wird als „DUT“ Device under test bezeichnet.

Die Entity einer Testbench ist immer leer, da es nach außen hin zum Simulator keine Ein- und Ausgangssignale gibt.

Simulationen

⇨ Verhaltenssimulation

Verifikation des Designs.
VHDL-Code muss nicht synthetisierbar sein.

⇨ Funktionale Simulation

Verifikation nach der Synthese

⇨ Pre Layout Gattersimulation

Nach dem Technologiemapping

⇨ Post Layout Gattersimulation

Nach Place&Route mit echtem Timing

Synthese

Aus einer Beschreibung auf der Register-Transfer-Ebene (Verhalten, Struktur) ist heute problemlos die Synthese einer elektronischen Schaltung möglich. Das System ist auf dieser Ebene ja bereits auf konkrete Baublöcke (Register, Operatoren) abgebildet, auf denen die Verarbeitung der Daten nach einem vorgegebenen Algorithmus und einem bestimmten Taktschema erfolgt. Die einzelnen Baublöcke und ihr Zusammenwirken kann dabei auf Verhaltensebene oder struktureller Ebene formuliert sein. Auf Grund der eindeutigen Abbildung erfüllen die Ergebnisse der Synthese die geforderten Entwurfsziele recht gut.

Die synthetisierte Struktur besteht aus zwei Komplexen:

- *Datenpfad :* Speicherelemente und Funktionseinheiten
- *Steuerwerk:* Steuerung des zeitlichen Ablaufs der Zustände mit den auszuführenden Datentransfers

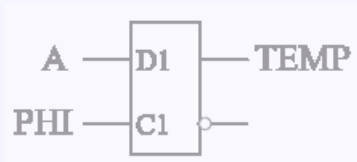
Synthese von Speicherelementen

Art der Schaltung

sequentielle Schaltung (Latch oder Register)

einem Signal oder einer Variablen nicht in allen Fällen ein Wert zugewiesen wird.

```
if PHI = '1' then
TEMP <= A;
end if;
```



Diese Beschreibung liefert ein Latch (taktzustandsgesteuertes FF). Das Signal PHI wirkt als Taktsignal. Auf Grund der fehlenden Formulierung für PHI = '0' behält die Schaltung den alten Wert bei.

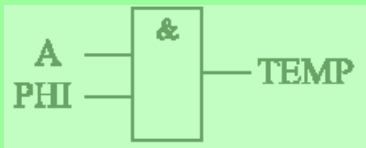
Taktflankenbezogene Signalabfragen liefern ein Register (taktflankengesteuertes FF).

if CLK'event and CLK = '1' **then**….

kombinatorische Schaltung

Damit eine kombinatorische Schaltung entsteht, muß dem Signal bzw. der Variablen in allen Fällen ein Wertzugewiesen werden.

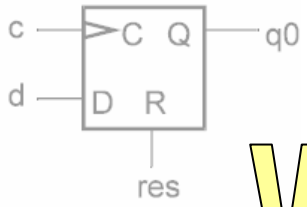
```
if PHI = '1' then
TEMP <= A;
else
TEMP <= '0';
end if;
```



Die Synthese erzeugt ein Logikgatter. Für jeden Wert von PHI ist eine Reaktion der Schaltung formuliert. Es kann eindeutig eine Logik synthetisiert werden.

ABEL vs. VHDL

Aufgabe:
Es ist ein D-FF zu entwerfen.



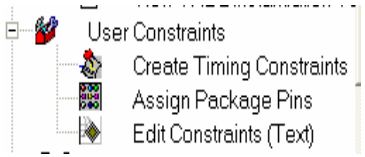
ABEL

```
MODULE dff
TITLE 'D-Flipflop'

DECLARATIONS
c,res,d PIN 81,80,79;
q0 PIN 74 ISTYPE 'REG';

EQUATIONS
q0 := d;
q0.CLK = c;
q0.RE = res;
END
```

VHDL

```
entity dff
  port ( res_i  : in std_logic;
         d_i    : in std_logic;
         c_i    : in std_logic;
         q0_o   : out std_logi);
end dff;
```

```
architecture RTL of dff is
  begin
     REG: process (res, clk_i)
        begin
           if (res_i = '1') then
                 d_o <=  '0';
           elsif (clk_i'event and clk_i = '1') then
                  d_o <= d_i;
           end if;
        end process REG;
end RTL;
```



Achtung: Pin`s werden in WebPACK bei:
User Constrains➔
Assign Package Pins definiert!

DANKE

und viel Spass mit VHDL