

GTI – ÜBUNG 13

ARITHMETIK UND VHDL

Aufgabe 1 – Arithmetik

2

Beschreibung

- ▶ Multiplizieren Sie die beiden Binärzahlen $a = 0100110$ und $b = 0101$ durch Anwendung der Methode, die bei der Implementierung eines sequentiellen Multiplizierers zum Einsatz kommt.
- ▶ Geben Sie die einzelnen Schritte und das Ergebnis explizit an

Hinweis: Hier hilft das Schulprinzip der schriftlichen Multiplikation

Aufgabe 1 – Arithmetik

► Multiplikation

Bitweise konjunktive Verknüpfung des Faktors A mit jeder Stelle des Faktors B

Schaubild:

$$\begin{array}{r}
 \begin{array}{cccc}
 a_3 & a_2 & a_1 & a_0
 \end{array} \times \begin{array}{cccc}
 b_3 & b_2 & b_1 & b_0
 \end{array} \\
 \hline
 \begin{array}{cccc}
 & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 + & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 \\
 + & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 \\
 + & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & \\
 \hline
 p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
 \end{array}
 \end{array}$$

Array-Multiplizierer:

Für jede Bitmultiplikation eine Baueinheit, d.h. insgesamt $|A| * |B|$

Sequentieller Multiplizierer:

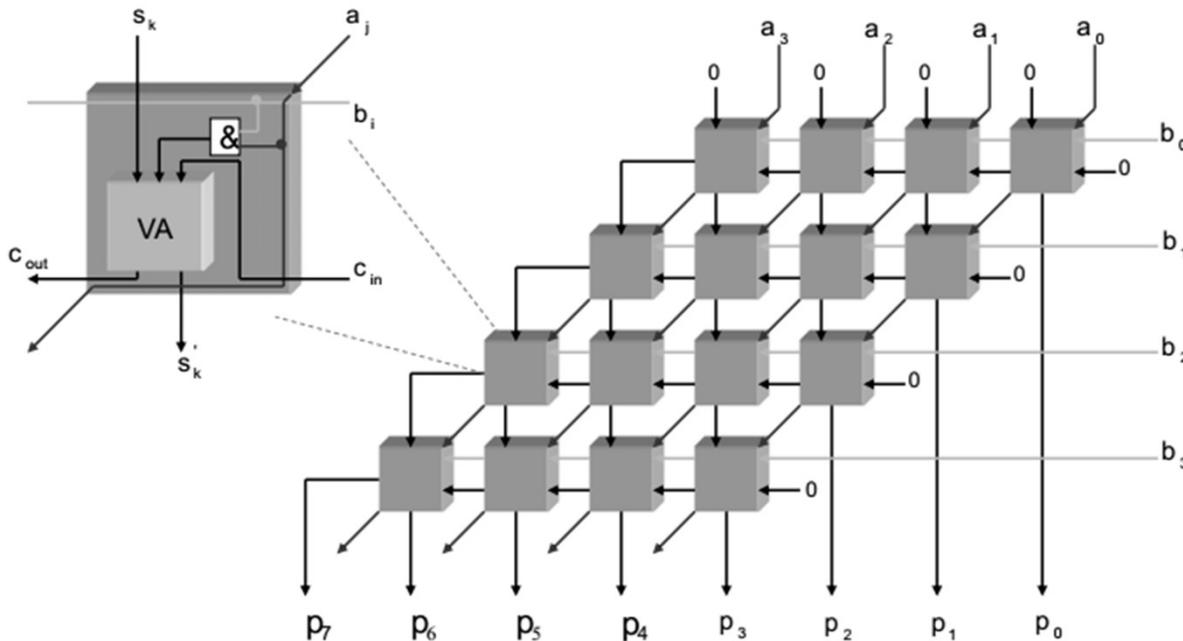
Ist b_i ($i = 0, \dots, |B|-1$) 0, so wird nichts addiert, sonst A mit Länge n geschiftet um i nach links (Realisierung mit Hilfe eines Schieberegisters)

Aufgabe 1 – Arithmetik

Array-Multiplizierer:

Problem: lange Laufzeit
hoher Hardwareaufwand

$$\begin{array}{r}
 a_3 a_2 a_1 a_0 \times b_3 b_2 b_1 b_0 \\
 \hline
 a_3 b_0 \ a_2 b_0 \ a_1 b_0 \ a_0 b_0 \\
 + \quad \quad \quad a_3 b_1 \ a_2 b_1 \ a_1 b_1 \ a_0 b_1 \\
 + \quad \quad \quad \quad a_3 b_2 \ a_2 b_2 \ a_1 b_2 \ a_0 b_2 \\
 + \quad \quad \quad \quad \quad a_3 b_3 \ a_2 b_3 \ a_1 b_3 \ a_0 b_3 \\
 \hline
 p_7 \ p_6 \ p_5 \ p_4 \ p_3 \ p_2 \ p_1 \ p_0
 \end{array}$$



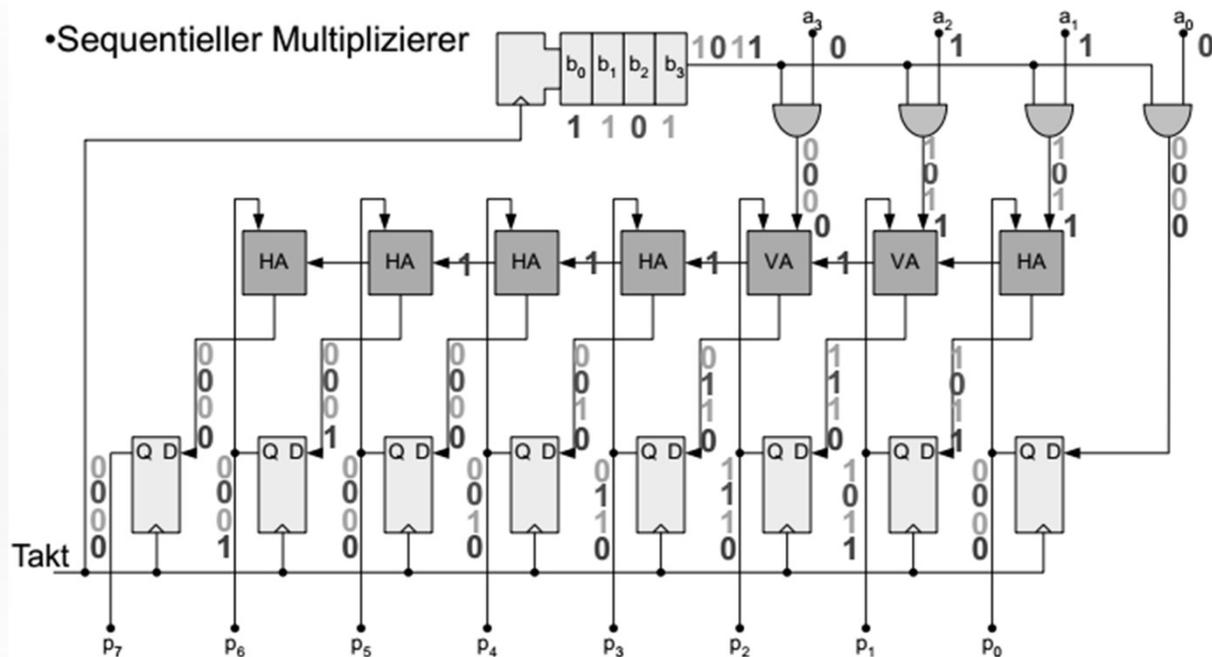
Aufgabe 1 – Arithmetik

5

Sequentieller Multiplizierer:

Verringerung des Hardwareaufwands.

Umsetzung mit Hilfe eines Schieberegisters:



Aufgabe 1 – Arithmetik

- ▶ Multiplikation der beiden Binärzahlen $a = 0100110$ und $b = 0101$

Sequentielles Multiplizieren:

Wir addieren das um i geshiftete Produkt von $A * b_i$ ($i = 0, \dots, |B|-1$)

$$\begin{array}{r} 0100110 * 0101 = \\ 1: \quad 0100110 \quad * 0 \\ 2: \quad + 0100110 \quad * 1 \\ 3: \quad + 0100110 \quad * 0 \\ 4: \quad \quad + 0100110 \quad * 1 \\ \hline = 0010111110 \end{array}$$

Aufgabe 1 – Arithmetik

- Explizite Angabe der einzelnen Schritte (a = 0100110 und b = 0101)

```
0100110 * 0101 =  
  
  0000000 | Reset  
+0000000 | Add a * b3  
  0000000 | Shift Left  
  
  0000000  
+0100110 | Add a * b2  
  0100110 | Shift Left  
  
  01001100  
+0000000 | Add a * b1  
  01001100 | Shift Left  
  
  10011000  
+0100110 | Add a * b0  
  10111110 | Ergebnis
```

Aufgabe 1 – Arithmetik

Beschreibung

- ▶ Dividieren Sie die Binärzahl $a = 011\ 1101$ durch die Binärzahl $b = 0110$ anhand des NonRestoring-Divisionsverfahrens.
- ▶ Geben Sie die einzelnen Schritte, sowie den Quotienten und den Partialrest explizit an.

Hinweis: Es gibt mehrere Divisionsverfahren.

Aufgabe 1 – Arithmetik

► Divisionsverfahren:

Allgemeines Prinzip:

$$\begin{array}{r}
 022697 : 124 = 183 \\
 \underline{124} \\
 1029 \\
 \underline{992} \\
 377 \\
 \underline{372} \\
 5
 \end{array}$$

Restoring Division:

- Versuchsweise Subtraktion in jedem Schritt
- Rückgängigmachen (Restoring), wenn ein negativer Rest auftritt

0 1 1 0 1 1 0 1 0 0	- - - - -	$R^{(0)}, Q^{(0)}$
0 1 1 0 1 1 0 1 0 0	- - - - -	linksschieben, subtrahieren
0 0 1 1 0 0 0 1 0 0	- - - - 1	positiver Rest
0 1 1 0 0 0 1 0 0	- - - 1 -	linksschieben, subtrahieren
0 0 1 0 0 1 1 0 0	- - - 1 1	positiver Rest
0 1 0 0 1 1 0 0	- - 1 1 -	linksschieben, subtrahieren
0 0 0 1 0 0 0 0	- - 1 1 1	positiver Rest
0 0 1 0 0 0 0	- 1 1 1 -	linksschieben, subtrahieren
1 1 1 0 0 1 0	- 1 1 1 0	negativer Rest addieren
0 0 1 0 0 0 0	- 1 1 1 0	
0 1 0 0 0 0	1 1 1 0 -	linksschieben, subtrahieren
R 0 0 0 0 1	Q 1 1 1 0 1	positiver Rest

Aufgabe 1 – Arithmetik

Non-restoring Division:

- Mit einem negativen Rest wird weitergearbeitet
- Negativer Rest wird nach der Entstehung, solange durch Additionen korrigiert, bis er wieder positiv geworden ist
- Ein negativer Rest im letzten Schritt wird durch einen Korrekturschritt ausgeglichen

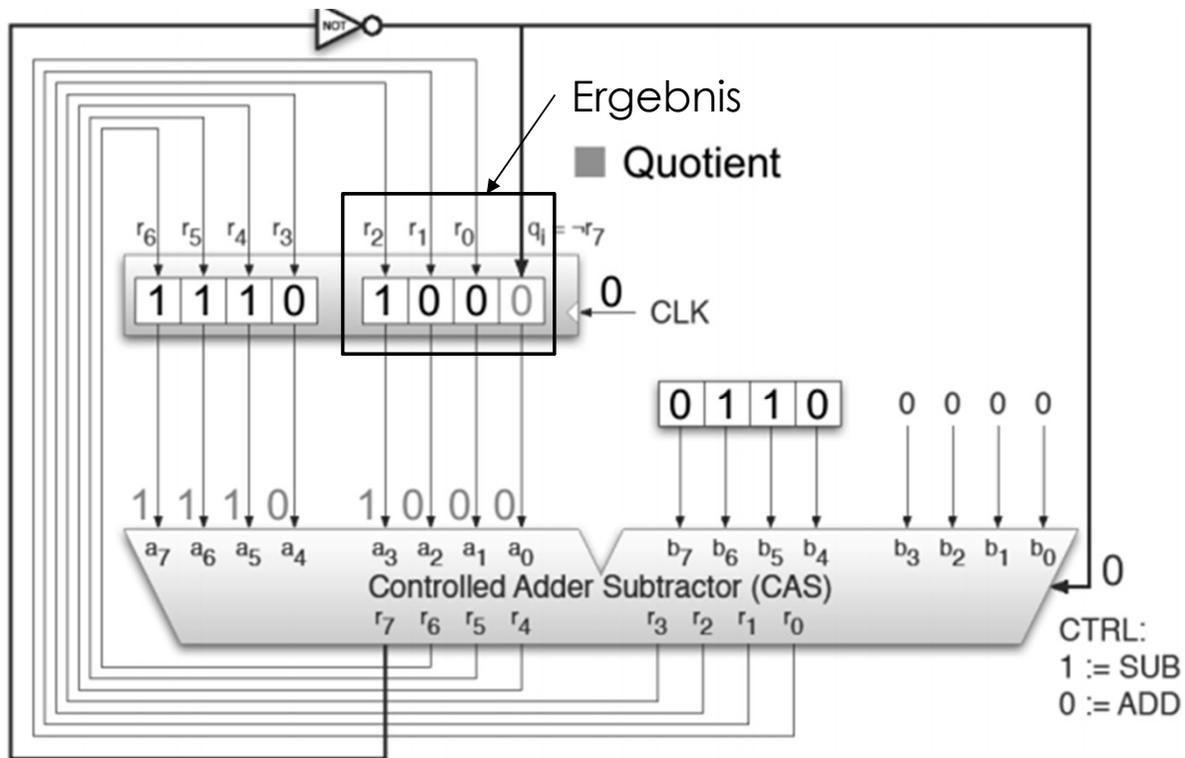
Schritt	Partialrest	Quotient	Beschreibung
0	0 1 0 1 1 0 1 0	- - - -	$R^{(0)} >= 0, Q^{(0)}$
	1 0 1 1 0 1 0 0	- - - -	$2 * R^{(0)}, Q^{(0)}$
	0 1 1 1 0 0 0 0	- - - -	$B' \text{ subtrahieren} \Rightarrow q_0 = 1$
1	0 1 0 0 0 1 0 0	- - - 1	$R^{(1)} >= 0, Q^{(1)} = 2 * Q^{(0)} + q_0$
	1 0 0 0 1 0 0 0	- - - 1	$2 * R^{(1)}, Q^{(1)}$
	0 1 1 1 0 0 0 0	- - - 1	$B' \text{ subtrahieren} \Rightarrow q_1 = 1$
2	0 0 0 1 1 0 0 0	- - 1 1	$R^{(2)} >= 0, Q^{(2)} = 2 * Q^{(1)} + q_1$
	0 0 1 1 0 0 0 0	- - 1 1	$2 * R^{(2)}, Q^{(2)}$
	0 1 1 1 0 0 0 0	- - 1 1	$B' \text{ subtrahieren} \Rightarrow q_2 = 0$
3	1 1 0 0 0 0 0 0	- 1 1 0	$R^{(3)} < 0, Q^{(3)} = 2 * Q^{(2)} + q_2$
	1 0 0 0 0 0 0 0	- 1 1 0	$2 * R^{(3)}, Q^{(3)}$
	0 1 1 1 0 0 0 0	- 1 1 0	$B' \text{ addieren} \Rightarrow q_3 = 0$
4 = 1	1 1 1 1 0 0 0 0	1 1 0 0	$R^{(4)} < 0, Q^{(4)} = 2 * Q^{(3)} + q_3$
	1 1 1 1 0 0 0 0	1 1 0 0	Nicht schieben: Nur
	0 1 1 1 0 0 0 0	1 1 0 0	$B' \text{ zur Korrektur addieren}$
	0 1 1 0 0 0 0 0	1 1 0 0	$R^{(4+1)}, Q^{(4)}$

Aufgabe 1 – Arithmetik

12

Gatterumsetzung:

Prinzip am Beispiel in den Vorlesungsfolien nachvollziehen (F. 73 - 78):



Aufgabe 1 – Arithmetik

13

Algorithmus $a/b = c$:

- Erweitere a um ein Vorzeichenbit
- Schiebe a eins nach links
- Bilde das Komplement von b
- Schiebe $-b$ soweit nach links, dass es mit a beginnt
- Wiederhole für jede Stelle von b
 - Rechne:
 - $a + (-b)$, wenn das carry = 1 | Normale Subtraktion
 - $a + b$, wenn das carry = 0 | Korrekturaddition
 - Speichere das carry als niedrigstes Bit (LSB) im Quotienten
 - Shifte a und q um 1 nach links
- Ist das letzte carry = 0, addiere Rest + b | Korrekturschritt

Aufgabe 1 – Arithmetik

14

Algorithmus $a/b = c$:

- Erweitere a um ein Vorzeichenbit
- Schiebe a eins nach links
- Bilde das Komplement von b
- Schiebe $-b$ soweit nach links, dass es mit a beginnt
- Wiederhole für jede Stelle von b
 - Rechne:
 - $a + (-b)$, wenn das carry = 1 | Normale Subtraktion
 - $a + b$, wenn das carry = 0 | Korrekturaddition
 - Speichere das carry als niedrigstes Bit (LSB) im Quotienten (q)
 - Shifte a und q um 1 nach links
- Ist das letzte carry = 0, addiere Rest + b | Korrekturschritt

Aufgabe 1 – Arithmetik

15

- Division $a = 011\ 1101$ durch $b = 0110$ mittels des NonRestoring-Verfahrens.

Schritt	Partialrest	Quotient	
0	0111101	----	a
	00111101	----	Vorzeichenbit
	001111010	----	Linksshift
	0110	----	b
	00110	----	Vorzeichenbit
	11001	----	Komplement
	11010	----	Komplement + 1
	110100000	----	Shift zu a

Aufgabe 1 – Arithmetik

16

Algorithmus $a/b = c$:

- Erweitere a um ein Vorzeichenbit
- Schiebe a eins nach links
- Bilde das Komplement von b
- Schiebe $-b$ soweit nach links, dass es mit a beginnt
- Wiederhole für jede Stelle von b
 - Rechne:
 - $a + (-b)$, wenn das carry = 1 | Normale Subtraktion
 - $a + b$, wenn das carry = 0 | Korrekturaddition
 - Speichere das carry als niedrigstes Bit (LSB) im Quotienten (q)
 - Shifte a und q um 1 nach links
- Ist das letzte carry = 0, addiere Rest + b | Korrekturschritt

Aufgabe 1 – Arithmetik

- Division $a = 011\ 1101$ durch $b = 0110$ mittels des NonRestoring-Verfahrens.

Schritt	Partialrest	Quotient	
1	0 01111010	----	a
	1 10100000	----	-b
	10 00011010	---1	a + (-b)
	0 00110100	--1-	Linksshift
2	0 00110100	--1-	a
	1 10100000	--1-	-b
	01 11010100	--10	a + (-b)
	1 10101000	-10-	Linksshift

Aufgabe 1 – Arithmetik

- Division $a = 011\ 1101$ durch $b = 0110$ mittels des NonRestoring-Verfahrens.

Unser Partialrest ist kleiner als 0, weswegen wir einen Korrekturschritt machen müssen. Wir müssen also b addieren (nicht subtrahieren!)

Schritt	Partialrest	Quotient	
3 KORREKTUR	1 10101000	-10-	a
	0 01100000	-10-	b
	10 00001000	-101	a+b
	0 00010000	101-	Linksshift

Aufgabe 1 – Arithmetik

- Division $a = 011\ 1101$ durch $b = 0110$ mittels des NonRestoring-Verfahrens.

Schritt	Partialrest	Quotient	
4	0 00010000	101-	a
	1 10100000	101-	-b
	01 10110000	1010	a + (-b)
5 KORREKTUR	+0 01100000		b Korrektur
	10 00010000		Rest

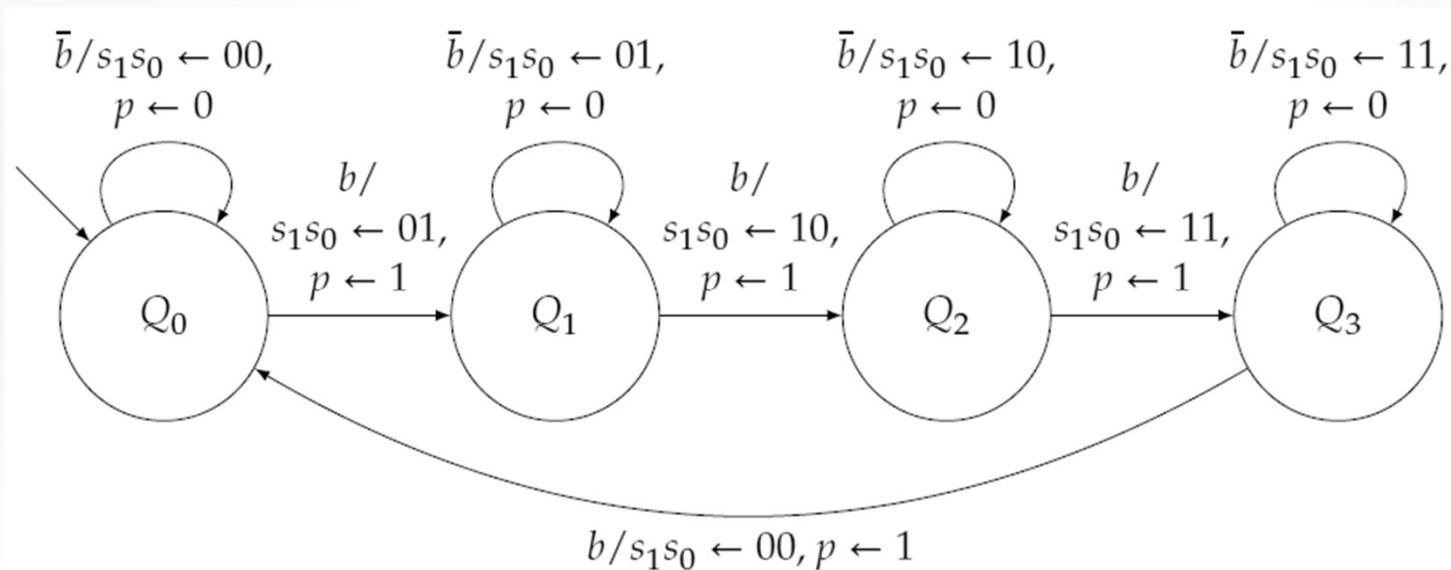
Das Ergebnis lautet: $a/b = 1010$ mit Rest 0001

Aufgabe 2 - VHDL

20

Beschreibung

- Implementieren Sie den folgenden Mealy-Automaten, der die Armbanduhr aus Übung 11 beschreibt, in VHDL. Der Automat soll mit einem synchronen Reset-Signal in den Anfangszustand zurückgesetzt werden können.

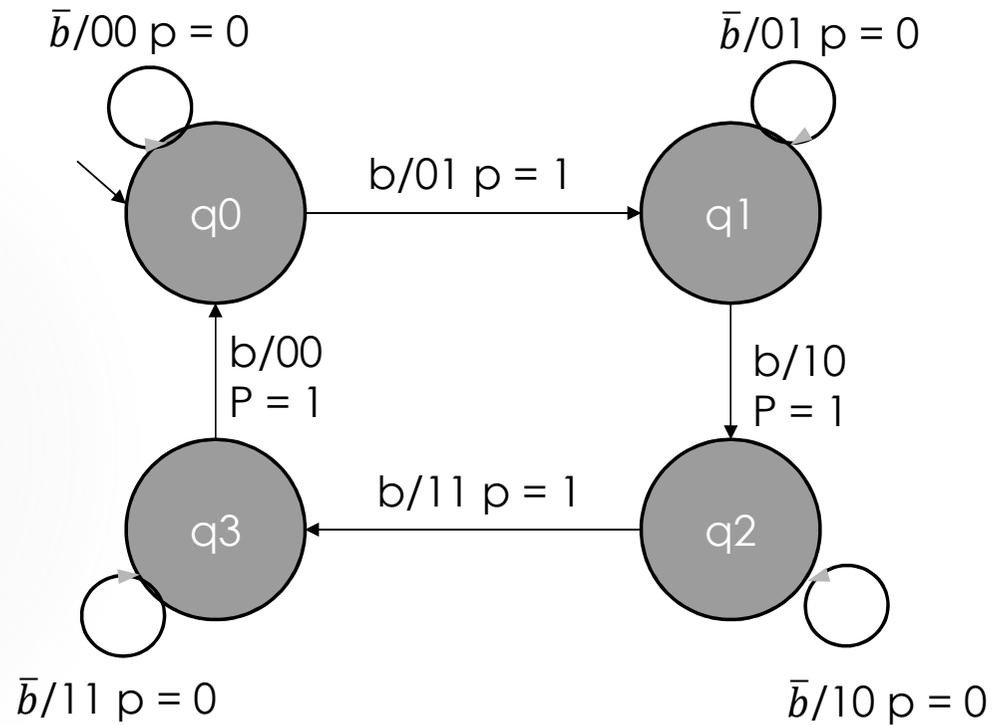


Aufgabe 2 - VHDL

21

Beschreibung

► Mealy-Automat:



Aufgabe 2 - VHDL

22

VHDL:

- Hardwarebeschreibungssprache, d.h. Signalleitungen und Logikgatter anstatt Variablen und Schleifen
- noch hardwarenäher als Assembler
- Pro:
 - Sehr gute Performance
 - Geeignet für zeitkritische Anwendungen
 - Spezielle Hardware-Anforderungen können realisiert werden
 - Perfekt für die Parallelisierung gleichzeitiger Aufgaben
- Contra:
 - Herausfordernde (umständliche) Programmierung
 - Nicht unbedingt für komplexe Algorithmen geeignet

Aufgabe 2 - VHDL

23

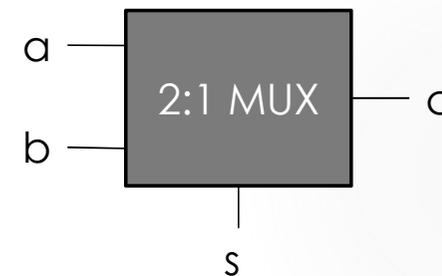
VHDL-Basics:

Entity:

- Beschreibt eine Blackbox einer Komponente
- Ein/Ausgänge, d.h. die Schnittstelle nach außen
- Beispiel: Eingang a, b; Steuersignal s; Ausgang c

Architecture:

- Beschreibt das Verhalten/Funktionsweise einer Komponente
- Mehrere Architekturen einer Entity möglich
- Beispiel: Wenn $s = 0$, dann $c = a$. Wenn $s = 1$, dann $c = b$.



Aufgabe 2 - VHDL

24

Entity:

```
entity MUX is
    port(
        i_a, i_b, i_s      : in      std_logic;
        o_c                : out    std_logic
    );
end entity MUX;
```

Name der Entity

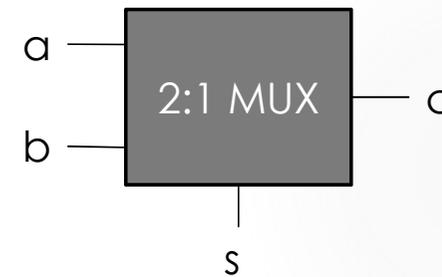
Liste der Eingangssignale

Signaltyp

Kennzeichnung der Richtung

Alle Zeilen enden mit ';' bis auf die letzte

Jede Entity wird so beendet



Konvention: Eingangssignale beginnen mit einem i_*, Ausgangssignale mit o_*

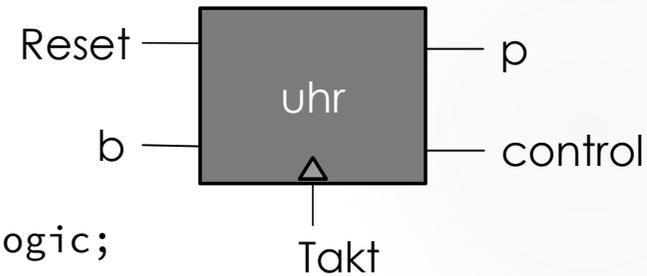
Aufgabe 2 - VHDL

25

- Implementieren Sie den oben entworfenen Mealy-Automaten

Entity:

```
entity uhr is
port(
    i_clk, i_rst      : in    std_logic;
    i_b               : in    std_logic;
    control           : out   std_logic_vector(1 downto 0);
    o_p               : out   std_logic
);
end entity uhr;
```



Bitvektor (d.h. mehrere Bits)
Vgl. mit einem Array (control[0] & control[1])
Also zwei Bit breites Wort

Aufgabe 2 - VHDL

26

Architecture:

architecture muxA of MUX is

Annotations: "Name der Architektur" points to muxA, "Name der Entity" points to MUX.

-- Ich bin ein Kommentar ☺

[interne Signale] ← Hilfssignale

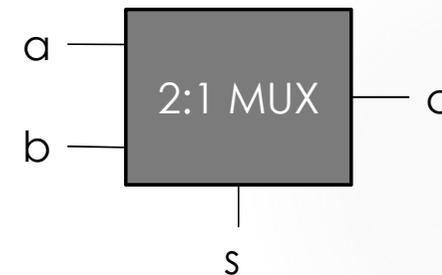
[Einbindungen von anderen Entitys] ← Falls diese gebraucht werden

begin ← Kennzeichnung des auszuführenden Teils

[eigentlicher Code]

end architecture muxA;

Annotation: "Jede Architektur wird so beendet" points to end architecture muxA;



Aufgabe 2 - VHDL

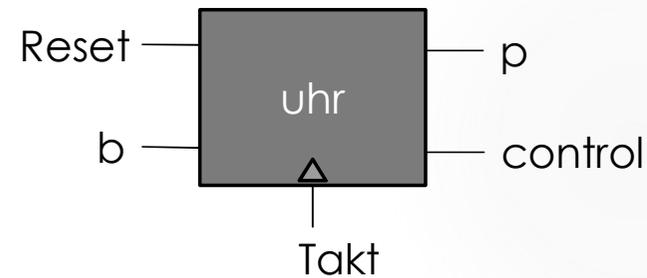
27

- Implementieren Sie den oben entworfenen Mealy-Automaten

Architecture:

```
architecture mealy of uhr is
--interne Signale

--Einbindungen von anderen Entitys
begin
--eigentlicher Code
end architecture mealy;
```



Aufgabe 2 - VHDL

28

- Implementieren Sie den oben entworfenen Mealy-Automaten

```
architecture mealy of uhr is
```

```
--interne Signale
```

```
type states is (q0, q1, q2, q3);
```

```
signal state : states;
```

Definition eines Signaltyps

Nimmt neuen Typ an

```
begin
```

```
end architecture mealy;
```

WICHTIG:
Eine Möglichkeit in der
Klausur seine Zustände
darzustellen!

Aufgabe 2 - VHDL

29

```
architecture mealy of uhr is
```

```
[...]
```

```
begin
```

Optional Name Sensitivitätsliste

```
FSM : process(i_clk)
```

```
begin
```

Bei jeder steigenden Flanke von i_clk

```
  if rising_edge(i_clk) then
```

```
    [sequentieller Code]
```

```
  end if;
```

```
end process FSM;
```

```
end architecture mealy;
```

process: Immer wenn sich ein Signal der Sensitivitätsliste ändert, wird der process durchlaufen

Aufgabe 2 - VHDL

30

- ▶ Mittels eines taktsynchronen Reset-Signals soll der Automat in den Anfangszustand versetzt werden können.

[...]

```
if rising_edge(i_clk)      then
    if i_rst = '1' then
        state <= q0;
        control <= „00“;
        o_p <= '0';
    else
        [...]
    end if;
end if;
[...]
```

Wertzuweisung bei Signalen mit <=;
eigene Typen ohne Anführungszeichen

Bitvektorzuweisung mit „...“

Std_logic-Zuweisung mit '...'

Wäre das reset-Signal asynchron, stände es außerhalb des rising_edge-Blocks und in der Sensitivitätsliste

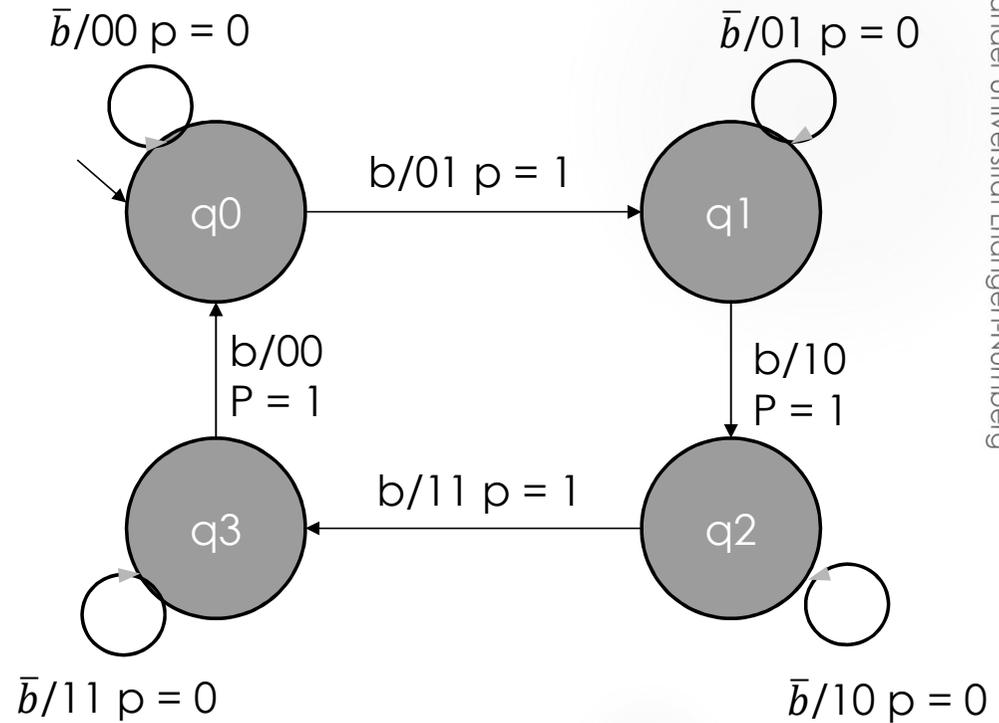
Aufgabe 2 - VHDL

31

```
[...]
else
  case state is
    when q0 =>
      if b = '1' then
        state <= q1;
        control <= „01“;
        p <= '1';
      else
        p <= '0';
      end if;
    [...]
  end case;
end if;
end if;
```

Typ nach dem unterschieden werden soll

Werteabfrage



Aufgabe 2 - VHDL

32

- Zustandsübergang für die ersten beiden Zustände

[...]

else

case state is

when q0 => [...]

when q1 =>

if b = '1' then

state <= q2;

control <= „10“;

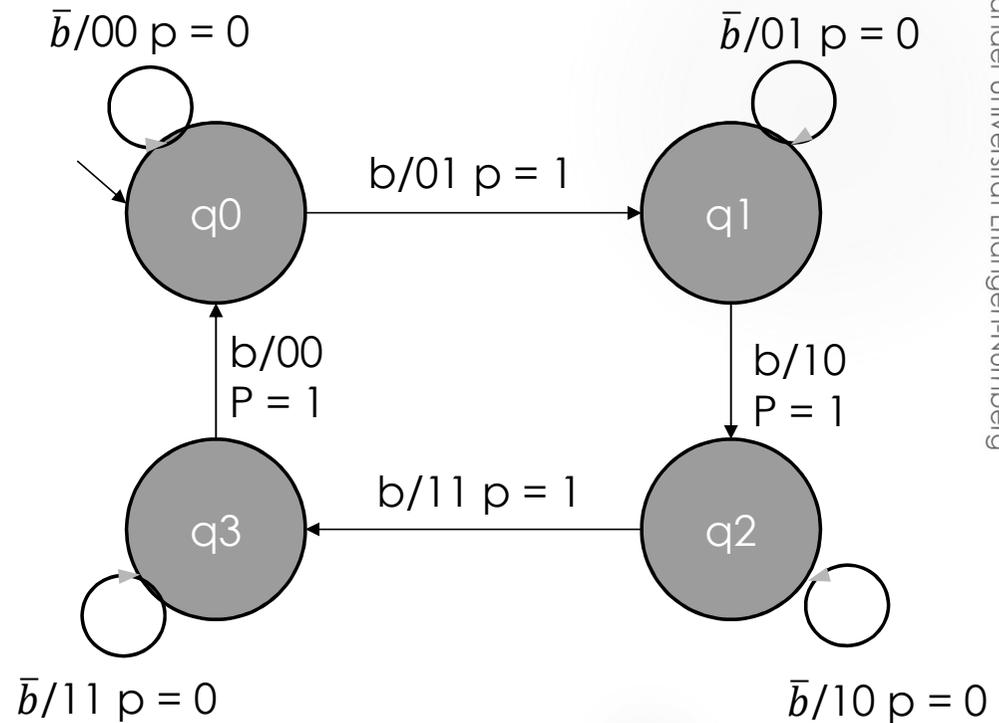
p <= '1';

else

p <= '0';

end if;

[...]



Aufgabe 2 - VHDL

33

- Zustandsübergang für die ersten beiden Zustände

[...]

else

case state is

[...]

when q2 =>

if b = '1' then

state <= q3;

control <= „11“;

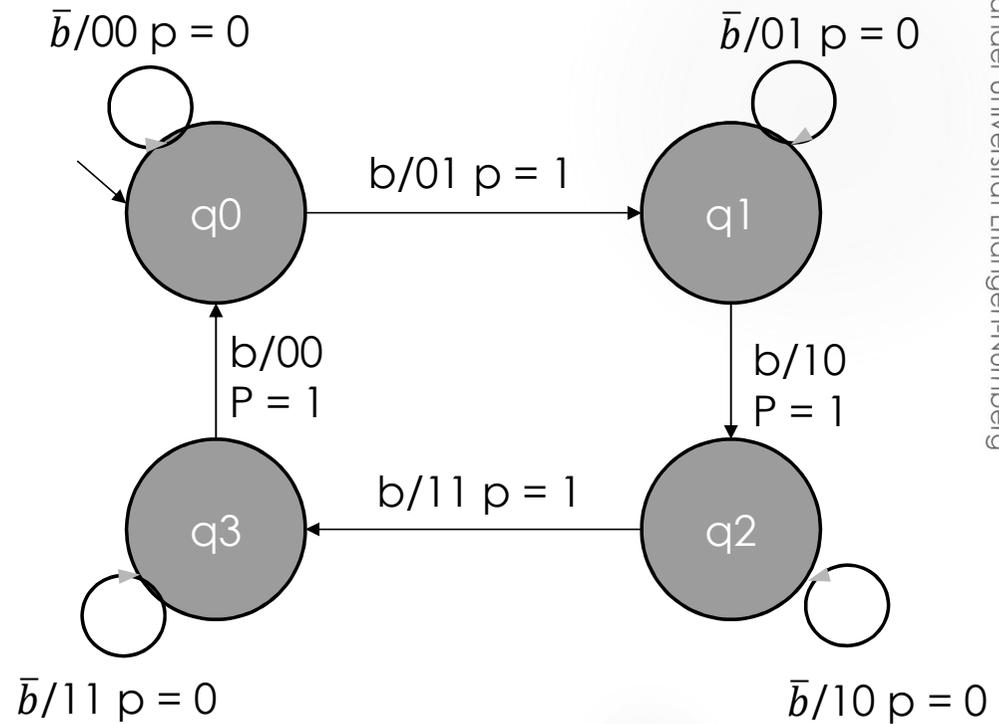
p <= '1';

else

p <= '0';

end if;

[...]



Aufgabe 2 - VHDL

34

- Zustandsübergang für die ersten beiden Zustände

[...]

else

case state is

[...]

when q3 =>

if b = '1' then

state <= q0;

control <= „00“;

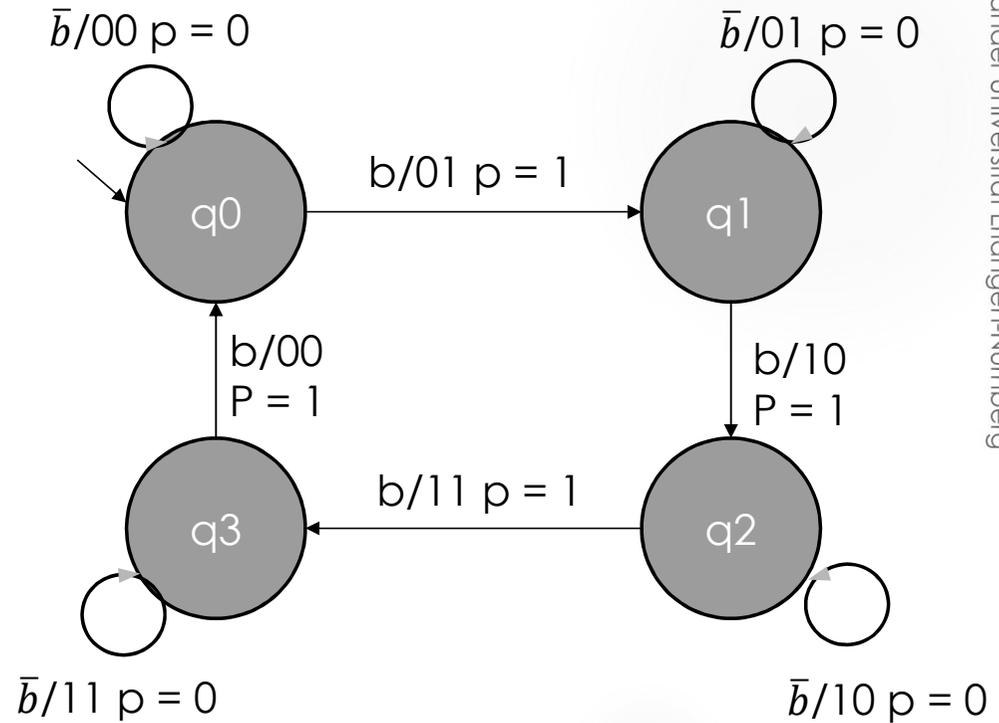
p <= '1';

else

p <= '0';

end if;

[...]



Aufgabe 2 - VHDL

35

- ▶ Zustandsübergang für ungültige Zustände als guter Stil

[...]

```
else
```

```
    case state is
```

```
        [...]
```

```
        when others =>      --sollte nie eintreten
```

```
    end case;
```

```
end if;
```

```
end if;
```

Aufgabe 2 - VHDL

36

```
architecture mealy of uhr is
[...]
begin
FSM : process(i_clk)
begin
    if rising_edge(i_clk) then
        [sequentieller Code]
    end if;
end process FSM;
end architecture mealy;
```

Aufgabe 3 - VHDL

37

Beschreibung

- ▶ Entwerfen Sie eine Funktion `OR_REDUCE` welche die Elemente eines beliebig langen Eingabe-Vektors vom Typ `std_logic_vector` mittels sukzessivem ODER-Vergleich auf eine 1 bit lange Ausgabe vom Typ `std_logic` reduziert.

Aufgabe 3 - VHDL

► Variablen – Signale

Syntax:

Variable:

- `variable beispiel : std_logic;` | Deklaration
- `beispiel := '0';` | Wertzuweisung

Signal:

- `signal beispiel : std_logic;` | Deklaration
- `beispiel <= '0';` | Wertzuweisung

Aufgabe 3 - VHDL

39

► Variablen – Signale

Semantik:

Variable:

- Sequentiell, d.h. die Reihenfolge im Code ist wichtig
- Überschreibbar (d.h. in Schleife nutzbar)

Signal:

- Nebenläufig (Concurrent), d.h. Reihenfolge egal
- Damit keine mehrmaligen Wertzuweisungen innerhalb eines Abschnitts

Aufgabe 3 - VHDL

40

► Funktionen/Procedures

- Lokale Variablen sind nur innerhalb der Funktion/Prozedur gültig
- Deklaration innerhalb der Architektur/Entity/Package

Funktionen:

- Unterprogramm mit Rückgabewert (können auch rekursiv aufgerufen werden)
- Können innerhalb von Architekturen aufgerufen werden

Prozedur:

- Unterprogramm ohne Rückgabewert
- Können innerhalb von Architekturen aufgerufen werden
- In Parameterliste kann Richtung des Parameters definiert werden (in/ out/ inout)

Aufgabe 3 - VHDL

42

- ▶ OR_REDUCE: n-Bit-std_logic_vector -> 1-Bit-std_logic

```
function OR_REDUCE(arg: std_logic_vector) return std_logic is
variable result : std_logic;
begin
    result := '0';
    [...]
end;
```

← Wertzuweisung mit :=

Aufgabe 3 - VHDL

- ▶ OR_REDUCE: n-Bit-std_logic_vector -> 1-Bit-std_logic

Function OR_REDUCE (arg: std_logic_vector) return std_logic is

```
variable result      : std_logic;
```

```
begin
```

```
    result := '0';
```

```
    for i in arg'range loop
```

```
        result := result or arg(i);
```

```
    end loop;
```

```
    return result;
```

```
end;
```

Länge des Bitvektors arg

i-ter Wert des Vektors



Logisches Oder

Schleifen:

- Loop
- for ... loop
- while ... loop

Achtung: kein sequentieller Code,
loops werden bei der Synthese
entrollt und weiterverarbeitet!

Aufgabe 4 - VHDL

Beschreibung

- ▶ Entwerfen Sie eine ALU (Rechenwerk), die, abhängig von einem Steuersignal `op` zwei 8-bit breite Eingangsworte `a` und `b` miteinander verarbeitet und auf dem 8-bit breiten Ausgangswort `result` ausgibt
 - ▶ $\text{result} \leftarrow a + b$ falls `op = 00`
 - ▶ $\text{result} \leftarrow a - b$ falls `op = 01`
 - ▶ $\text{result} \leftarrow a \bmod b$ falls `op = 10`
 - ▶ $\text{result} \leftarrow a \text{ lshift } b$ falls `op = 11`
- ▶ Verwenden Sie `std_logic_vector` für die Schnittstelle und die IEEE-Bibliothek `numeric_std` für die Berechnungen.

Aufgabe 4 - VHDL

► Case/If/When

If:

- Prioritär, d.h. die Reihenfolge der Auswertung ist durch die Verschachtelung festgelegt (deswegen auch in Hardware teurer, priority encoding)
- Syntax: `if [Condition] then ... elsif ... else ... endif;`

Case:

- Alle when-Fälle werden gleichzeitig ausgewertet (als Multiplexer inferiert)
- Syntax: `Case [Attribut] is when [Cond1] => ... when [Cond2] => ... end case;`

When:

- Für Unterscheidung bei einer Signalzuweisung (nicht prioritär)
- Syntax: `sig1 <= [Wert1] when [Cond1] else [Wert2] when [Code2] else [Wert3];`

Aufgabe 4 - VHDL

- ▶ Verwenden Sie `std_logic_vector` für die Schnittstelle und die IEEE-Bibliothek `numeric_std` für die Berechnungen.

```
library ieee;
use ieee.std_logic_1164.all; --std_logic_vector
use ieee.numeric_std.all; --Arithmetik
```

Zu importierende Bibliothek

Stellen Verknüpfungen und Typen zur Verfügung

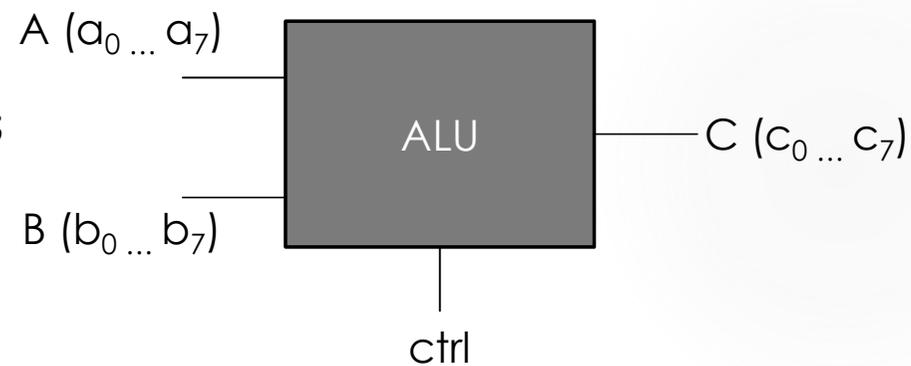
Aufgabe 4 - VHDL

47

► Entity:

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

entity alu is
port(
    a, b    : in  std_logic_vetor(7 downto 0);
    op     : in  std_logic_vetor(1 downto 0);
    result  : out std_logic_vetor(7 downto 0)
);
end entity alu;
```



Aufgabe 4 - VHDL

48

► Architecture:

```
architecture rechnen of alu is
signal signed_result : signed(7 downto 0);
begin
[...]
```

end architecture rechnen;

Spezieller Typ aus Bibliothek:
Vorzeichenbehaftet



Aufgabe 4 - VHDL

49

► Architecture: Variante 1

```
[...]
signed_result <= signed(a) + signed(b)   when op = „00“ else
                signed(a) - signed(b)   when op = „01“ else
                signed(a and b)         when op = „10“ else
                signed(a sll 1);

result      <= std_logic_vector(signed_result );
end architecture;
```

Importierten Operanden

Fallunterscheidung mit when [Cond] else
Alternative: if (aber länger)

Casten in das vorzeichenbehaftete
Format

Casten der vorzeichenbehafteten Ergebnisses
In das vorzeichenlose Format

Aufgabe 4 - VHDL

50

► Architecture: Variante 2

[...]

```
with op select signed_result <=
    signed(a) + signed(b)      when „00“,
    signed(a) - signed(b)      when „01“,
    signed(a and b)            when „10“,
    signed(a sll 1)            when „11“,
    “00000000”                when others;
result <= std_logic_vector(signed_result);
end architecture;
```

Fallunterscheidung mit with signal select

Das war die letzte GTI-Übung ☹️

51

Ein Paar letzte Worte:

- ▶ Verzeihung für die (vielen) Fehler, die ich gemacht habe
- ▶ Dazu die letzten Worte aus dem Film „Some Like It Hot“ (1959)
 - ▶ Well, nobody's perfect
- ▶ Falls Fragen bezüglich der Klausur aufkommen sollten:
 - ▶ E-Mail an: jspnbg@web.de
 - ▶ Tolles Kontaktformular 😊
- ▶ Euch noch ein angenehmes, sowie erfolgreiches Reststudium!
- ▶ Und natürlich (ansonsten wärt ihr ja schwer enttäuscht 😊) gibt es noch (!) ein tolles Zitat zum Abschluss!

Ein letztes Mal: vielen Dank für eure wunderbare Aufmerksamkeit!

„सर सलामत, तो पगड़ी हज़ार“

Aussprache: Sar salamat, to pagri hazar.

Übersetzung:

Wenn dein Kopf heil ist, kannst du tausend Turbane haben.