Einführung in die technische Informatik

Christopher Kruegel chris@auto.tuwien.ac.at

http://www.auto.tuwien.ac.at/~chris

VHDL

- VHDL
  - Akronym für

    Very High-Speed Integrated Circuit Hardware Description Language
  - Hardwarebeschreibungssprache
  - unterstützt das Hardware Design

- Konzepte
  - Partitionierung
    - Unterteilung des Entwurfs in unterschiedliche Komponenten
  - Abstraktion
    - Beschreibung der Komponenten auf unterschiedlicher Ebene

Y-Modell

- Y-Modell
  - unterstutzt systematischen Entwurf von Schaltungen
  - selben Konzepte
    - Partitionierung
    - Abstraktion
- 3 Komponenten (Sichtweisen)
  - Verhalten
  - Struktur
  - Geometrie

Y-Modell

- Verhalten
  - was macht die Komponente?
  - Spezifikation, Algorithmen, Boolsche Gleichungen
- Struktur
  - woraus ist die Komponente aufgebaut?
  - Module, Gatter, Leitungen
- Geometrie
  - wie sieht die Komponente aus?
  - Flurplan, Masken

VHDL - Modell

- Architektur
  - was macht die Komponente?
- Strukturbeschreibung
  - Schnittstellenbeschreibung
  - welche Interfaces bietet die Komponente nach aussen?
- Konfiguration
  - welche Architektur wird verwendet?

VHDL - Modell

- Konfiguration
  - verknüpft Architektur mit Strukturbeschreibung
  - selben Schnittstellen können unterschiedliche Funktionalität erfüllen
  - ein Interface kann unterschiedliche Implementierungen besitzen
  - Beispiel vom letzten Mal
    - die selbe Komponente mit 4 Eingängen und 1 Ausgang kann völlig unterschiedliche Funktionalität besitzen

f_1 -- f(x) = 1 ↔ x < 7 (x ist 4-bit Zahl)
f_2 -- f(x,y) = 1 ↔ x < y (x und y sind 2-bit Zahlen)

VHDL - Packages

- Package
  - Bibliothek von häufig verwendeten Teilen
    - Funktionen
    - Komponenten
    - Konstanten
  - Einbinden in die VHDL Beschreibung

```
library IEEE;
use     IEEE.std_logic_1164.all;
```


VHDL - Entity

- VHDL Entity
  - zentraler Baustein bei VHDL
  - Grundobjekt im VHDL Design Prozess
  - besteht aus
    - Schnittstellenbeschreibung (Interface)
    - Deklarieren eines Objekts mit Anschlüssen
    - Keyword **`entity`**
    - Architektur (Implementierung)
    - Realisieren der Funktion(en)
    - Keyword **`architecture`**

VHDL - Entity

- Syntax der Interface Beschreibung

```
entity entity_name is
     port ( in/out connections )
end entity_name;
```

- Beispiel
  - 2-zu-1 Multiplexer

```
entity 2_to_1_mux is
     port    ( i1, i2, select: IN bit;
               o1: OUT bit;            )
end 2_to_1_mux;
```


VHDL - Architektur

- Syntax der Architektur Beschreibung

```
architecture architecture_name of entity_name is
     [ local variables ]
begin
     [ process ]
end architecture_name;
```

- Verhaltensbeschreibung durch *Prozesse*

VHDL - Prozess

- Prozess
  - mit herkömmlichem Programm vergleichbar
    - sequentielle Abarbeitung der Anweisungen
    - Kontrollstrukturen
  - Eingabe und Ausgabe erfolgt über Signale
    - Einlesen von Werten über IN Verbindungen
    - Setzen der OUT Verbindungen auf bestimmte Werte
  - alle Prozesse laufen parallel ab
  - Aktivierung erfolgt durch *sensitivity-list*

    sensitivity-list steht an der Stelle von Funktionsargumenten

VHDL - Prozess

- Syntax der Prozess Beschreibung

```
process_name : process (sensitivity_list)
begin
   [ Anweisungsblock ]
end process process_name;
```

- Beispiel

```
compare : process (input1)
begin
   if (input1 == input2)
      output <= 1;
   end if;
end process compare;
```

VHDL - Prozess

- Syntax der Prozess Beschreibung

```
process_name : process (sensitivity_list)
begin
   [ Anweisungsblock ]
end process process_name;
```

- Beispiel

```
compare : process (input1)
begin
   if (input1 == input2)
      output <= 1;
   end if;
end process compare;
```

Prozess startet nur, wenn sich der Wert von `input1` ändert

VHDL - Prozess

Automation Systems Group

- Syntax der Prozess Beschreibung

```
process_name : process (sensitivity_list)
begin
   [ Anweisungsblock ]
end process process_name;
```

- Beispiel

```
compare : process (input1)
begin
   if (input1 == input2)
      output <= 1;
   end if;
end process compare;
```

Prozess startet nur, wenn sich der Wert von `input1` ändert

trotzdem kann auf `input2` zugegriffen werden

VHDL - Prozess

- Syntax der Prozess Beschreibung

```
process_name : process (sensitivity_list)
begin
   [ Anweisungsblock ]
end process process_name;
```

- Beispiel

```
compare : process (input1)
begin
   if (input1 == input2)
      output <= 1;
   end if;
end process compare;
```

Prozess startet nur, wenn sich der Wert von `input1` ändert

trotzdem kann auf `input2` zugegriffen werden

⇒ sensitivity-list != Funktionsargumente

VHDL - Prozess

- Anweisungsblock
  - Normalen Operationen
    - **`if X, elsif X, else, end if`**
    - **`case X is, when, end case`**
  - Zuweisung von Werten
    - an normale Variable mit **`:=`**

      `x := 0;`
    - an Signale mit **`<=`**

      `out_signal <= 0;`

VHDL - Prozess

- Anweisungsblock
  - Felder (Arrays)
    - Deklaration mittels
      oder

```
vname : vector(a to b)
vname : vector(a downto b)
```

    - Zugriff auf Element
      oder auf ganzes Feld

```
vname(index) := 0;
vname := "<Werte-Liste>";
bitfield := "0110";
```


VHDL - Beispiel

Angabe

Ein 1-bit Volladdierer soll in VHDL beschrieben werden. Dieser Addierer addiert die zwei Bit `x` und `y`, sowie den Übertrag der vorigen Stufe (`c_in`) und gibt das Ergebnis (`sum`) sowie den Übertrag für die nächste Stufe (`c_out`) aus.

VHDL - Lösung

1.Schritt -- Entity definieren

```
entity adder is
  port (x, y, c_in: IN bit;
        c_out, sum: OUT bit; )
end adder;
```


VHDL - Lösung

2. Schritt -- Funktionalität überlegen

X	Y	C_IN	SUM	C_OUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VHDL - Lösung

3. Schritt -- Funktionalität implementieren

```
architecture add of adder is
begin
    add_values : process (x, y, c_in)
        if  (x = '1' AND y = '1' AND c_in = '1') then
            sum <= '1'; c_out <= '1';
        elsif (x = '1' AND (y = '1' OR c_in = '1')) then
            sum <= '0'; c_out <= '1';
        elsif (y = '1' AND c_in = '1') then
            sum <= '0'; c_out <= '1';
        elsif (x = '1' OR y = '1' OR c_in = '1' ) then
            sum <= '1'; c_out <= '0';
        else
            sum <= '0'; c_out <= '0';
        end if;
   end process add_values;
end add;
```


VHDL - Lösung

Automation Systems Group

3. Schritt -- Funktionalität implementieren

```
architecture add of adder is
begin
    add_values : process (x, y, c_in)
        if  (x = '1' AND y = '1' AND c_in = '1') then
            sum <= '1'; c_out <= '1';
        elsif (x = '1' AND (y = '1' OR c_in = '1')) then
            sum <= '0'; c_out <= '1';
        elsif (y = '1' AND c_in = '1') then
            sum <= '0'; c_out <= '1';
        elsif (x = '1' OR y = '1' OR c_in = '1' ) then
            sum <= '1'; c_out <= '0';
        else
            sum <= '0'; c_out <= '0';
        end if;
   end process add_values;
end add;
```

VHDL - Lösung


```
if  (x = '1' AND y = '1' AND c_in = '1') then
    sum <= '1'; c_out <= '1';
elsif (x = '1' AND (y = '1' OR c_in = '1')) then
    sum <= '0'; c_out <= '1';
elsif (y = '1' AND c_in = '1') then
    sum <= '0'; c_out <= '1';
elsif (x = '1' OR y = '1' OR c_in = '1' ) then
    sum <= '1'; c_out <= '0';
else
    sum <= '0'; c_out <= '0';
```

X	Y	C_IN	SUM	C_OUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VHDL - Lösung


```
if  (x = '1' AND y = '1' AND c_in = '1') then
    sum <= '1'; c_out <= '1';
elsif (x = '1' AND (y = '1' OR c_in = '1')) then
    sum <= '0'; c_out <= '1';
elsif (y = '1' AND c_in = '1') then
    sum <= '0'; c_out <= '1';
elsif (x = '1' OR y = '1' OR c_in = '1' ) then
    sum <= '1'; c_out <= '0';
else
    sum <= '0'; c_out <= '0';
```

X	Y	C_IN	SUM	C_OUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VHDL - Lösung


```
if  (x = '1' AND y = '1' AND c_in = '1') then
    sum <= '1'; c_out <= '1';
elsif (x = '1' AND (y = '1' OR c_in = '1')) then
    sum <= '0'; c_out <= '1';
elsif (y = '1' AND c_in = '1') then
    sum <= '0'; c_out <= '1';
elsif (x = '1' OR y = '1' OR c_in = '1' ) then
    sum <= '1'; c_out <= '0';
else
    sum <= '0'; c_out <= '0';
```

X	Y	C_IN	SUM	C_OUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VHDL - Lösung


```
if  (x = '1' AND y = '1' AND c_in = '1') then
    sum <= '1'; c_out <= '1';
elsif (x = '1' AND (y = '1' OR c_in = '1')) then
    sum <= '0'; c_out <= '1';
elsif (y = '1' AND c_in = '1') then
    sum <= '0'; c_out <= '1';
elsif (x = '1' OR y = '1' OR c_in = '1' ) then
    sum <= '1'; c_out <= '0';
else
    sum <= '0'; c_out <= '0';
```

X	Y	C_IN	SUM	C_OUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VHDL - Lösung


```
if  (x = '1' AND y = '1' AND c_in = '1') then
    sum <= '1'; c_out <= '1';
elsif (x = '1' AND (y = '1' OR c_in = '1')) then
    sum <= '0'; c_out <= '1';
elsif (y = '1' AND c_in = '1') then
    sum <= '0'; c_out <= '1';
elsif (x = '1' OR y = '1' OR c_in = '1' ) then
    sum <= '1'; c_out <= '0';
else
    sum <= '0'; c_out <= '0';
```

X	Y	C_IN	SUM	C_OUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VHDL - Beispiel

Angabe

Realisieren Sie einen Zähler, der von 0 bis 3 zählt und dann wieder auf 0 zurückspringt. Der Zähler wird asynchron mit einem `reset` Eingang auf 0 gesetzt und zählt bei jedem Clock Tick (`clk`) um eins hoch.

VHDL - Lösung

1.Schritt -- Entity definieren

```
entity counter is
  port (reset, clk: IN bit;
        counter: OUT vector(1 downto 0); )
end counter;
```


VHDL - Lösung

2. Schritt -- Funktionalität überlegen

- Zähler benötigt 2-bit internen Zustand (`internal_cnt`)
- am Ausgang wird einfach interner Zählerstand ausgegeben

- Übergangsfunktion

CLK	IC_1	IC_0	IC_1	IC_0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

VHDL - Lösung

2. Schritt -- Funktionalität überlegen

- Zähler benötigt 2-bit internen Zustand (`internal_cnt`)
- am Ausgang wird einfach interner Zählerstand ausgegeben

- Übergangsfunktion

CLK	IC_1	IC_0	IC_1	IC_0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Überlauf - Zähler springt zurück auf Null

VHDL - Lösung

3. Schritt -- Funktionalität implementieren

```
architecture count of counter is
   internal_cnt : vector(1 downto 0);
begin
   cnt_proc : process (reset, clk)
        [ Prozess Ablauf ]
   end process cnt_proc;
end count;
```

VHDL - Lösung

3. Schritt -- Funktionalität implementieren

```
architecture count of counter is
    internal_cnt : vector(1 downto 0);
begin
    cnt_proc : process (reset, clk)
        [ Prozess Ablauf ]
    end process cnt_proc;
end count;
```

Interner Zustand als 2-bit Vektor

VHDL - Lösung

3. Schritt -- Funktionalität implementieren

```
architecture count of counter is
    internal_cnt : vector(1 downto 0);
begin
    cnt_proc : process (reset, clk)
        [ Prozess Ablauf ]
    end process cnt_proc;
end count;
```

Prozessablauf

VHDL - Lösung

3. Schritt -- Prozessablauf

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```


VHDL - Lösung

3. Schritt -- Prozessablauf

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```

Asynchroner Teil -
Initialisieren des Zählers

VHDL - Lösung

3. Schritt -- Prozessablauf

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```

Synchroner Teil -
Reaktion auf Clock Tick

VHDL - Lösung

3. Schritt -- Prozessablauf

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```

Update des internen Zustands
laut Übergangsfunktion

VHDL - Lösung

3. Schritt -- Prozessablauf

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```

Übernahme des internen Zählers am Ausgang

VHDL - Lösung

3. Schritt -- Prozessablauf

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```

Zuweisung eines Werts an lokale Variable

VHDL - Lösung

3. Schritt -- Prozessablauf

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```

Zuweisung eines Werts
an Ausgangsvariable (Signal)

VHDL - Lösung

3. Schritt -- Prozessablauf

`internal_cnt` ist 2-bit Vektor

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```

Zuweisung an 2-bit Vektor

VHDL - Lösung

3. Schritt -- Prozessablauf

internal_cnt ist 2-bit Vektor

```
cnt_proc : process (reset, clk)
    if reset = '1' then
        internal_cnt := '00';
    elsif clk = '1' then
        case internal_cnt is
        when '00' => internal_cnt := '01';
        when '01' => internal_cnt := '10';
        when '10' => internal_cnt := '11';
        when '11' => internal_cnt := '00';
        end case;
        counter <= internal_cnt;
    end if;
end process cnt_proc;
```

Vergleich mit 2-bit Vektor

VHDL - Synthese

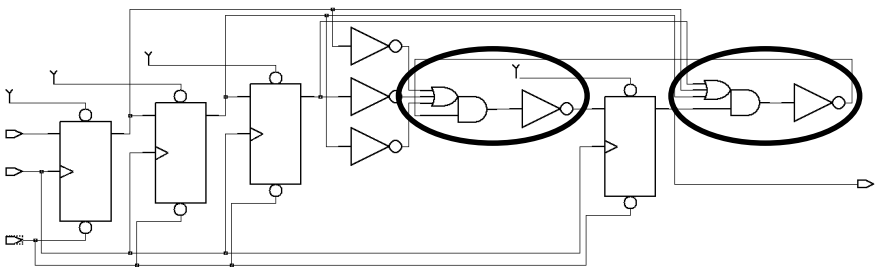
- Synthese
  - nachdem VHDL Spezifikation vorliegt
  - umsetzen der Spezifikation in logische Schaltungen
  - unter Verwendung von Standardelementen
- Standardelemente
  - Anweisungen eines Prozesses werden systematisch in logische Bauteile (Gatter, Latches) umgesetzt
  - nicht jede VHDL Spezifikation kann synthetisiert werden

VHDL - Synthese

- Standardelemente
  - Standardverbindungen
    - OAI (Or-And-Inverter)
    - AOI (And-Or-Inverter)
  - Kontrollstrukturen (`if`)
    - Multiplexer
  - lokale Variablen
    - Latches
  - Entkopplung
    - Operationsverstärker (OAmp)

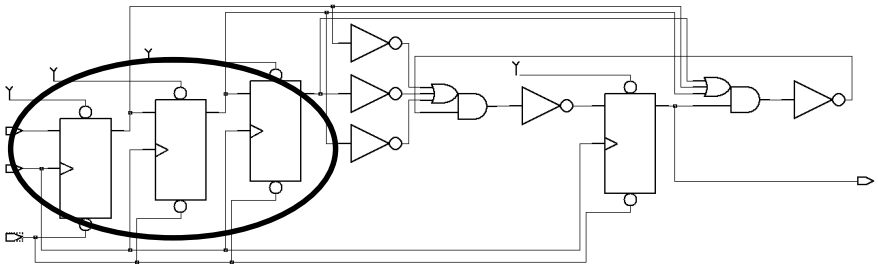
VHDL - Synthese

- Tastenentpreller
  - OAI (Or-And-Inverter)



VHDL - Synthese

- Tastenentpreller
  - Latches (für lokales 3-bit Schieberegister)



Zusammenfassung

- VHDL
  - Hardwarebeschreibungssprache
  - Idee
    - Hierarchie und Abstraktion
    - Y-Modell (Verhalten, Struktur, Geometrie)
  - 3 Komponenten
    - Schnittstellen - **`entity`**
    - Funktionalität - **`architecture`**
    - Konfiguration
  - Design und Synthese

Zusammenfassung

- Anforderungen
  - theoretisches Wissen und Konzepte
    - Buch
  - Verstehen von VHDL Spezifikationen
    - Folien von der VU-Webpage

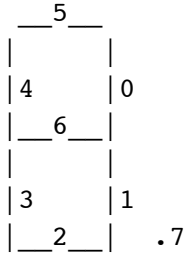
    `http://www.auto.tuwien.ac.at/~schi/LVAs/vhdl_beispiele1a.doc`
  - Erstellen eigener, simpler Spezifikationen
    - Übungsvortrag

VHDL - Folien

Angabe

Es soll ein Siebensegment Decoder realisiert werden. Die Aufgabe dieses Decoders ist die Darstellen einer Binärzahl (0-7) auf einem Display mit sieben Segmenten.

Zuordnung der Bussignale zu den LEDs (LEDs sind low-aktiv)



VHDL - Lösung

Interface

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity binaertosiebenseg is
  port (data  :in  std_logic_vector(2 downto 0);
        digit :out std_logic_vector(7 downto 0)
       );
end binaertosiebenseg;
```


VHDL - Lösung

Interface

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity binaertosiebenseg is
  port (data  :in  std_logic_vector(2 downto 0);
        digit :out std_logic_vector(7 downto 0)
       );
end binaertosiebenseg;
```

3 bit Eingabe
(Zahlen von 0 bis 7)

VHDL - Lösung

Interface

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity binaertosiebenseg is
  port (data  :in  std_logic_vector(2 downto 0);
        digit :out std_logic_vector(7 downto 0)
       );
end binaertosiebenseg;
```

8 bit Ausgabe
(7 Segmente und Punkt)

VHDL - Lösung

Funktionalität

```
architecture behaviour of binaertosiebenseg is
begin
  CONVERT : process(data)
  begin
   case data is
     when "000" =>  digit <= "01000000";  --Ausgabe "0"
     when "001" =>  digit <= "01111100";  --Ausgabe "1"
     when "010" =>  digit <= "00010010";  --Ausgabe "2"
     when "011" =>  digit <= "00011000";  --Ausgabe "3"
     when "100" =>  digit <= "00101100";  --Ausgabe "4"
     when "101" =>  digit <= "00001001";  --Ausgabe "5"
     when "110" =>  digit <= "00000001";  --Ausgabe "6"
     when OTHERS => digit <= "11011100";  --Ausgabe "7"
    end case;
  end process CONVERT;
end behaviour;
```

VHDL - Lösung

Funktionalität

```
architecture behaviour of binaertosiebenseg is
begin
  CONVERT : process(data)
  begin
   case data is
     when "000" =>  digit <= "01000000";  --Ausgabe "0"
     when "001" =>  digit <= "01111100";  --Ausgabe "1"
     when "010" =>  digit <= "00010010";  --Ausgabe "2"
     when "011" =>  digit <= "00011000";  --Ausgabe "3"
     when "100" =>  digit <= "00101100";  --Ausgabe "4"
     when "101" =>  digit <= "00001001";  --Ausgabe "5"
     when "110" =>  digit <= "00000001";  --Ausgabe "6"
     when OTHERS => digit <= "11011100";  --Ausgabe "7"
    end case;
  end process CONVERT;
end behaviour;
```

Abfrage auf Inputdaten (data)

VHDL - Lösung

Automation Systems Group

Funktionalität

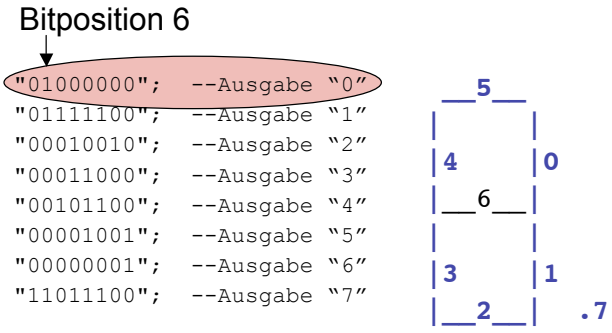
```
architecture behaviour of binaertosiebenseg is
begin
  CONVERT : process(data)
  begin
   case data is
     when "000" =>  digit <= "01000000";  --Ausgabe "0"
     when "001" =>  digit <= "01111100";  --Ausgabe "1"
     when "010" =>  digit <= "00010010";  --Ausgabe "2"
     when "011" =>  digit <= "00011000";  --Ausgabe "3"
     when "100" =>  digit <= "00101100";  --Ausgabe "4"
     when "101" =>  digit <= "00001001";  --Ausgabe "5"
     when "110" =>  digit <= "00000001";  --Ausgabe "6"
     when OTHERS => digit <= "11011100";  --Ausgabe "7"
    end case;
  end process CONVERT;
end behaviour;
```

Einfaches `switch` statement

VHDL - Lösung

Funktionalität

```
architecture behaviour of binaertosiebenseg is
begin
  CONVERT : process(data)
  begin
   case data is
     when "000" =>  digit <= "01000000";  --Ausgabe "0"
     when "001" =>  digit <= "01111100";  --Ausgabe "1"
     when "010" =>  digit <= "00010010";  --Ausgabe "2"
     when "011" =>  digit <= "00011000";  --Ausgabe "3"
     when "100" =>  digit <= "00101100";  --Ausgabe "4"
     when "101" =>  digit <= "00001001";  --Ausgabe "5"
     when "110" =>  digit <= "00000001";  --Ausgabe "6"
     when OTHERS => digit <= "11011100";  --Ausgabe "7"
    end case;
  end process CONVERT;
end behaviour;
```

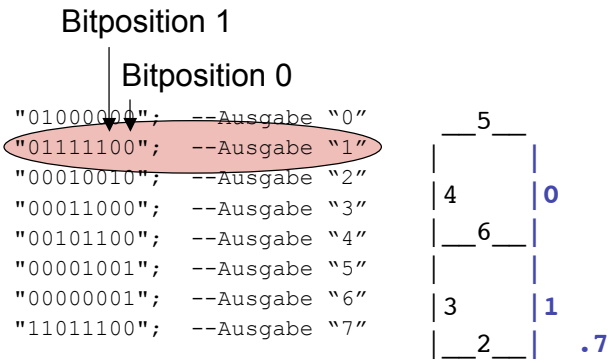


VHDL - Lösung

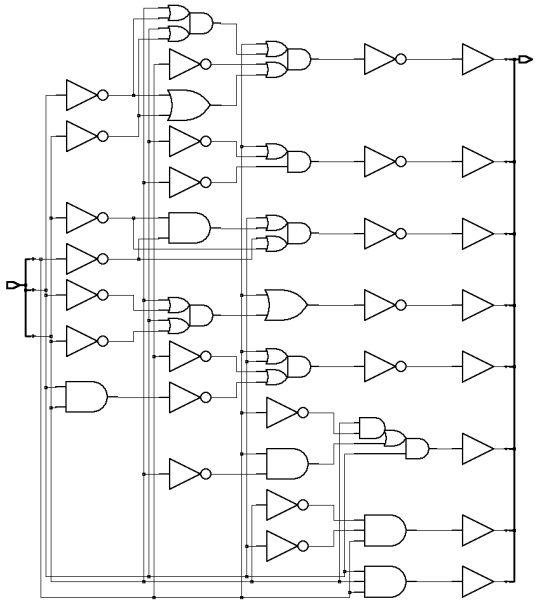
Automation Systems Group

Funktionalität

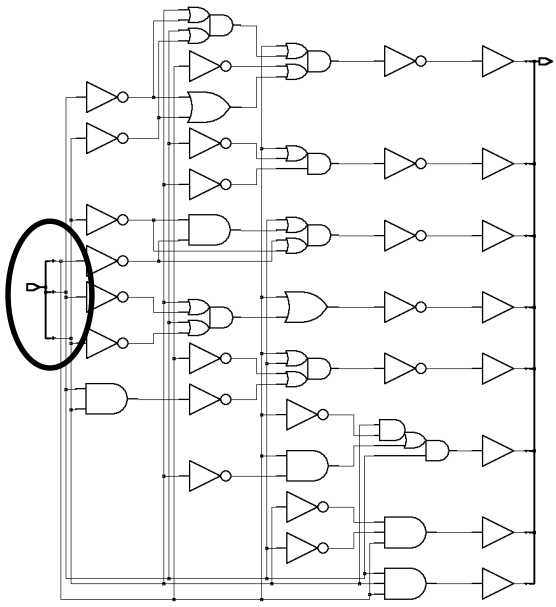
```
architecture behaviour of binaertosiebenseg is
begin
  CONVERT : process(data)
  begin
   case data is
     when "000" =>  digit <= "01000000";  --Ausgabe "0"
     when "001" =>  digit <= "01111100";  --Ausgabe "1"
     when "010" =>  digit <= "00010010";  --Ausgabe "2"
     when "011" =>  digit <= "00011000";  --Ausgabe "3"
     when "100" =>  digit <= "00101100";  --Ausgabe "4"
     when "101" =>  digit <= "00001001";  --Ausgabe "5"
     when "110" =>  digit <= "00000001";  --Ausgabe "6"
     when OTHERS => digit <= "11011100";  --Ausgabe "7"
    end case;
  end process CONVERT;
end behaviour;
```



VHDL - Synthese



VHDL - Synthese



VHDL - Synthese