Protokoll zum Praktikum Programmierbare Schaltkreise

Fakultät Informatik
TU Dresden

Christian Kroh
Matrikelnummer: 3755154
Studiengang: Informatik (Diplom)
Jahrgang: 2010/2011

Patrick Russell
Matrikelnummer: s0970860
Studiengang: Informatik (Diplom)
Jahrgang: 2010/2011

28. November 2013, Dresden

Aufgabe 1 - Binär-Dekoder

1.1 Entwurf

Input 4-Bit Binärzahl durch Schieberegister SW3 ... SW0

Output 7-Segmente Darstellung einer Hexadezimalziffer (8 Einzelsignale = 7 Segmente + 1 Punkt)

Input				Output								
SW3	SW2	SW1	SW0	Hex	A	B	C	D	E	F	G	dot
0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	1	1	1	0	0	1	1	1	1	1
0	0	1	0	2	0	0	1	0	0	1	0	1
0	0	1	1	3	0	0	0	0	1	1	0	1
0	1	0	0	4	1	0	0	1	1	0	0	1
0	1	0	1	5	0	1	0	0	1	0	0	1
0	1	1	0	6	0	1	0	0	0	0	0	1
0	1	1	1	7	0	0	0	1	1	1	1	1
1	0	0	0	8	0	0	0	0	0	0	0	1
1	0	0	1	9	0	0	0	0	1	0	0	1
1	0	1	0	A	0	0	0	1	0	0	0	1
1	0	1	1	b	1	1	0	0	0	0	0	1
1	1	0	0	C	0	1	1	0	0	0	1	1
1	1	0	1	d	1	0	0	0	0	1	0	1
1	1	1	0	E	0	1	1	0	0	0	0	1
1	1	1	1	F	0	1	1	1	0	0	0	1

7.1 Decoder.vhdl Code

1.2 Auswertung

Ressourcenbedarf

- 7 Logik-Elemente
- 12 Pins

Aufgabe 2 - Hamming-Distanz

2.1 Entwurf

Input 2 4-Bit Werte

- 1.Wert: 4-Bit Binärzahl durch Schieberegister SW3 ... SW0
- 2.Wert: 4-Bit Binärzahl durch Schieberegister SW7 ... SW4

Output 7-Segmente Darstellung einer Hexadezimalziffer (8 Einzelsignale = 7 Segmente + 1 Punkt)

Ansatz SW3 ... SW0 und SW7 ... SW4 logisch xor verknüpfen und Ergebnis direkt auf 7-Segmente Anzeige mappen

(7.2 Hamming.vhdl Code)

2.2 Auswertung

Ressourcenbedarf

- 9 Logik-Elemente
- 16 Pins

Aufgabe 3 - Modulo-n-Zähler

3.1 Entwurf

a) Der Zähler ist nach 50 Millionen Schritten zurückzusetzen (50 MHz Takt entspricht 50 Millionen Taktperioden pro Sekunde)

b) Für das Schieberegister ist der Zählerzustand ein Enable-Signal

c)

Input

- 50MHz Takt
- Reset (Schiebeschalter SW0)

Output LED-Zeile

Ansatz 2 Komponenten: Schieber und Zähler

Zähler gibt alle 50-Millionen Taktperioden (50MHz Takt ergibt $50 \cdot 10^6$ Taktschritte pro Sekunde) einen Takt lang ein enable-Signal aus. (7.3 Zaehler.vhdl-Code)

Schieber beinhaltet den Zähler als Komponente und verschiebt bei dessen enable-Signal die LED-Anzeige um eine Stelle pro Takt. (7.3 Schieber.vhdl-Code)

3.2 Auswertung

Ressourcenbedarf

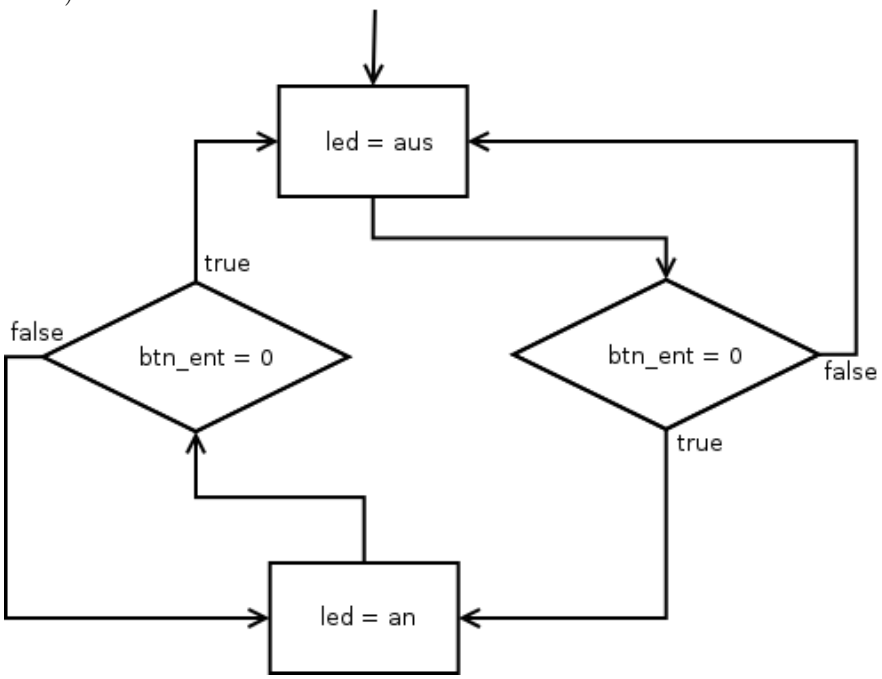
- 60 Logik-Elemente
- davon 38 dedizierte Logik-Elemente
- 12 Pins
- maximale Taktfrequenz von 250 MHz

Aufgabe 4 - Entprell-Automat

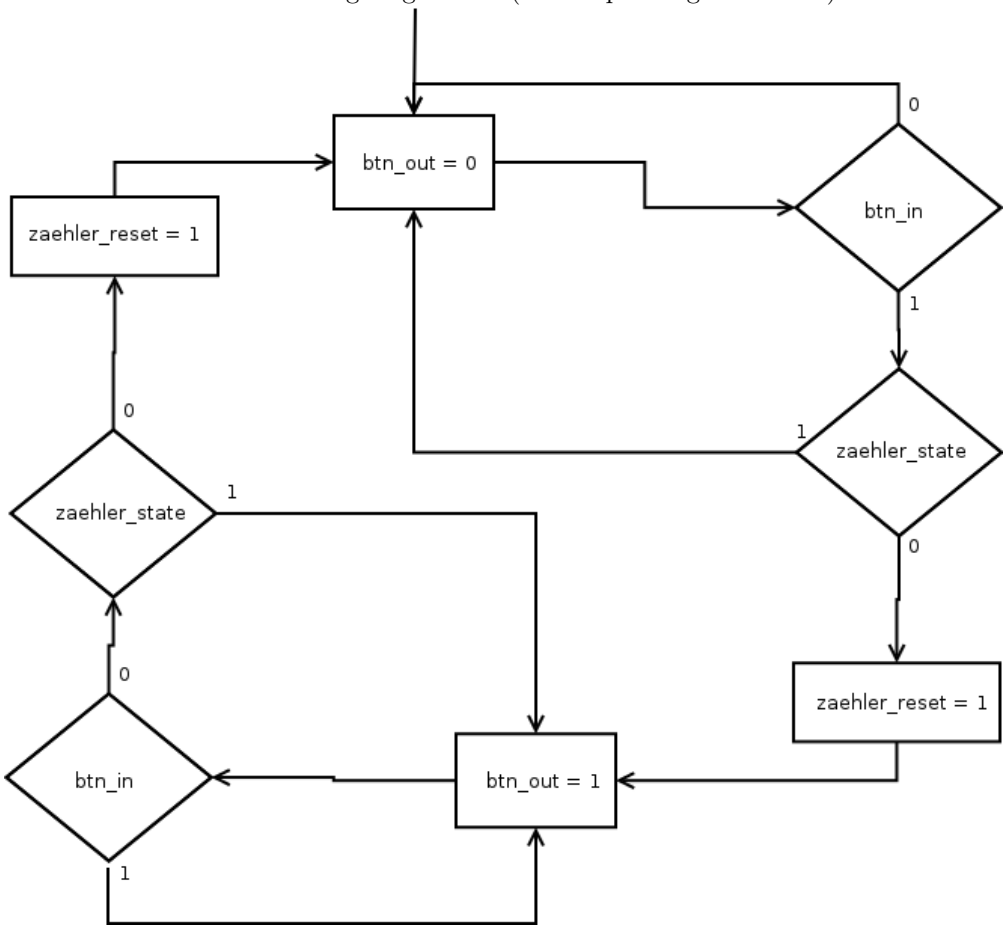
4.1 Entwurf

State-Machine-Charts

LED enthält die Komponente Entprellung und verbindet die Ein- und Ausgangssignale. (7.4 LED.vhdl-Code)



Entprellung enthält die Komponente Zaehler, der bei der Veränderung des Eingangsignals gestartet wird und für 3ms weitere Änderungen ignoriert. (7.4 Entprellung.vhdl-Code)



Zaehler implementiert einen Zähler, der durch ein Signal definierte Schritte zählt. Ausgegeben wird der aktuelle Zustand des Zählers. Eingegeben ein Reset-Signal. (7.4 Zaehler.vhdl-Code)

Kopplung

Es wird eine synchrone Automatenkopplung mit einem Taktsignal verwendet.

4.2 Auswertung

Ressourcenbedarf

- 86 Logik-Elemente
- davon 79 dedizierte Logik-Elemente
- 44 Register
- 3 Pins
- maximale Taktfrequenz von 178 MHz

Aufgabe 5 - HALLO-Anzeige

5.1 Entwurf

zu a) Es müssen 5 Zeichen kodiert werden (H, A, L, O, Leerzeichen).

$ld\,5 = 3$

Daher werden für eine Binärkodierung mindestens 3 Bits benötigt.

Input			Output								
BIT2	BIT1	BIT0	CHAR	A	B	C	D	E	F	G	DOT
0	0	0		1	1	1	1	1	1	1	1
0	0	1	H	1	0	0	1	0	0	0	1
0	1	0	A	0	0	0	1	0	0	0	1
0	1	1	L	1	1	1	0	0	0	1	1
1	0	0	O	0	0	0	0	0	0	1	1

b) Für das Schieberegister ist der Zählerzustand ein Enable-Signal

c)

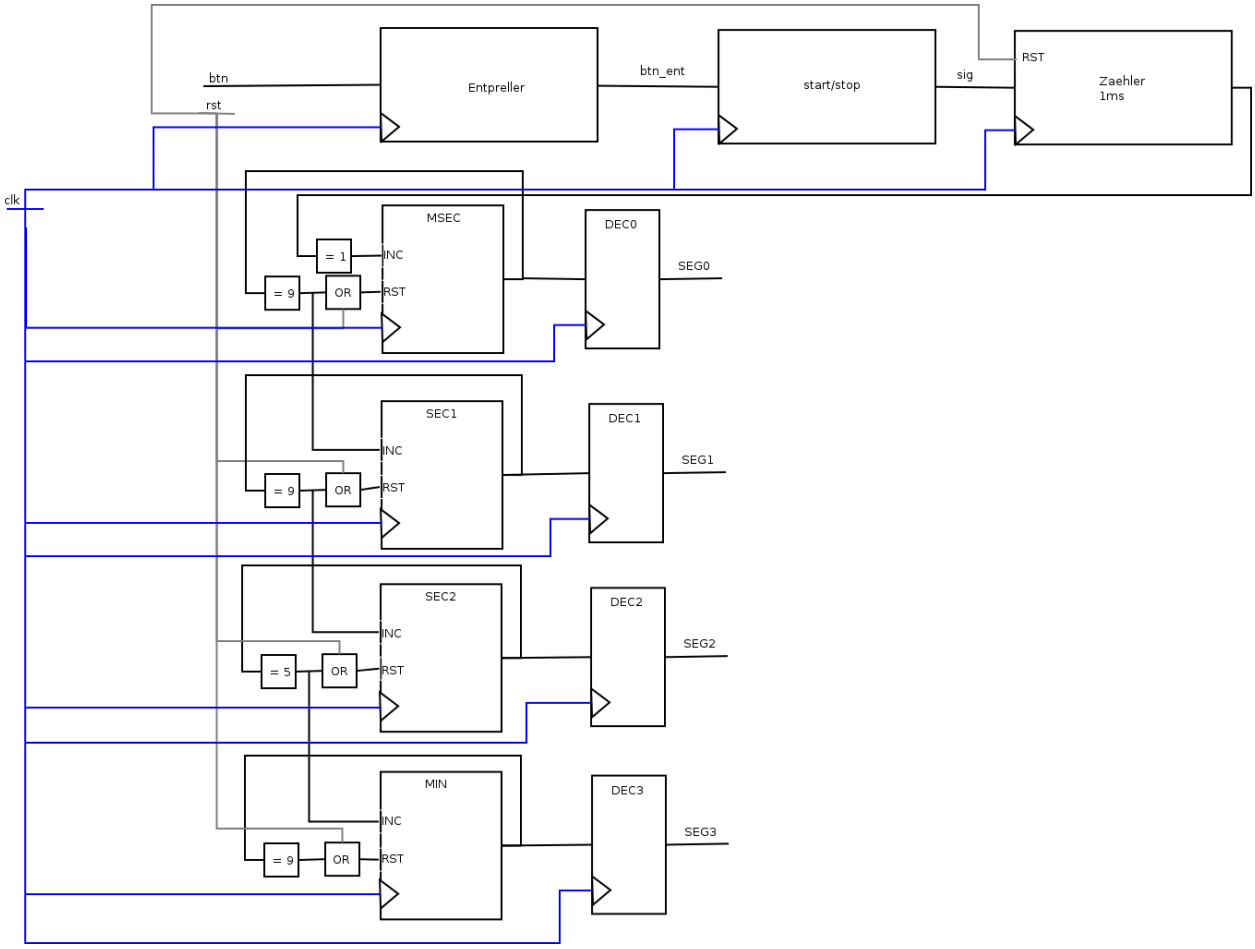
5.2 Auswertung

Ressourcenbedarf

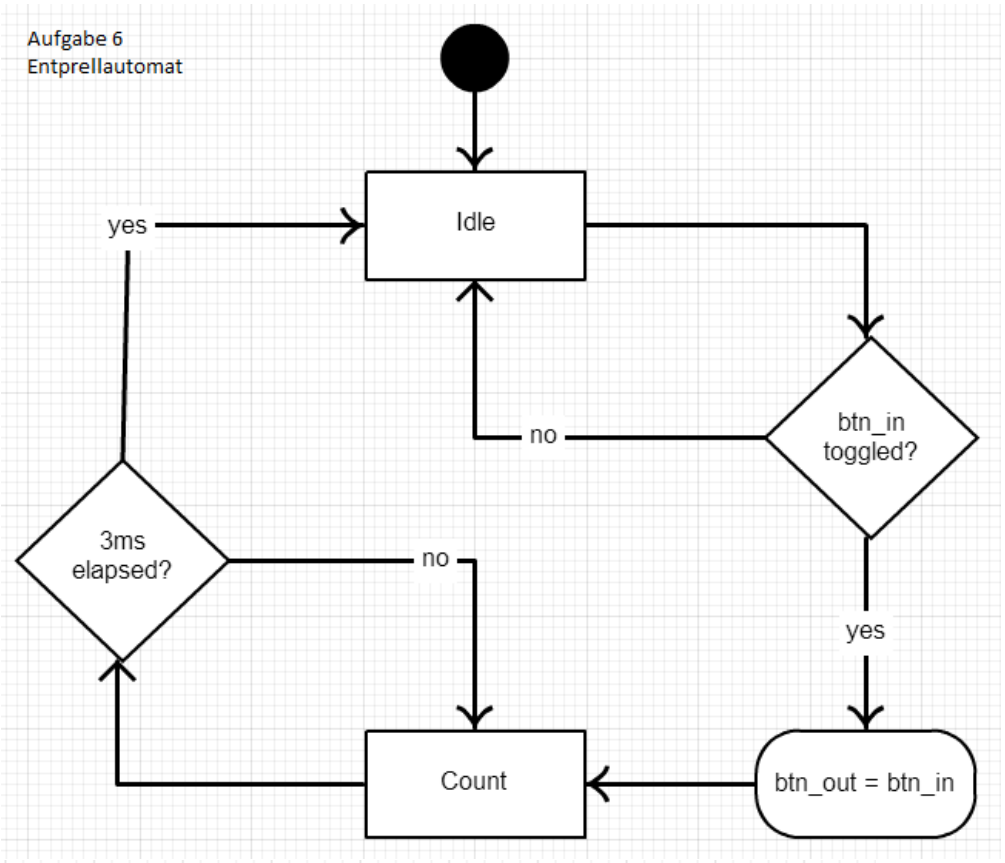
- 73 Logik-Elemente
- 61 Register
- 34 Pins
- maximale Taktfrequenz von 262 MHz

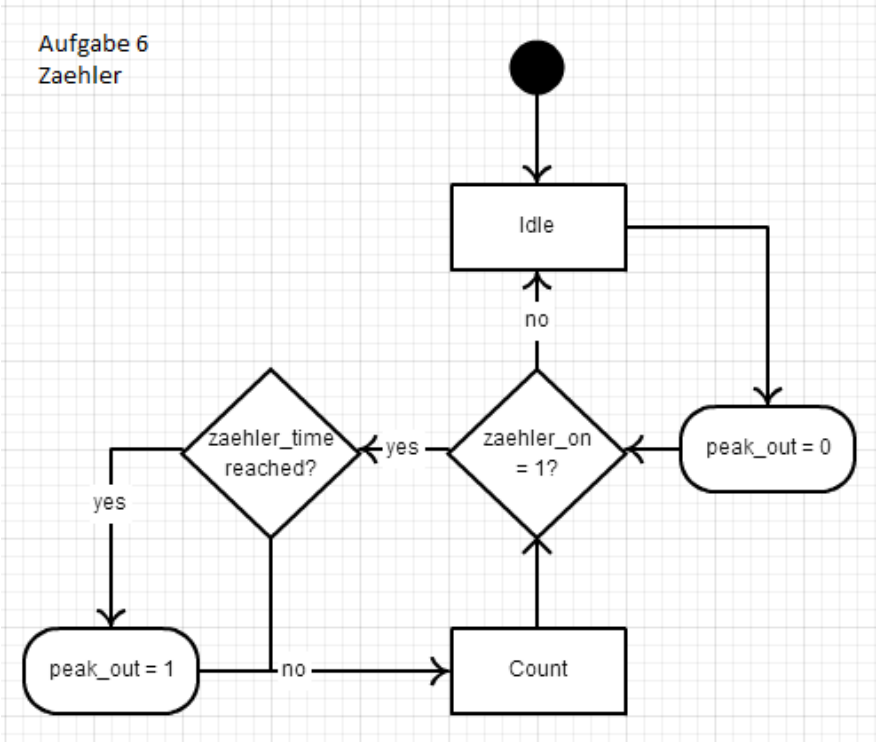
Aufgabe 6 - Stoppuhr

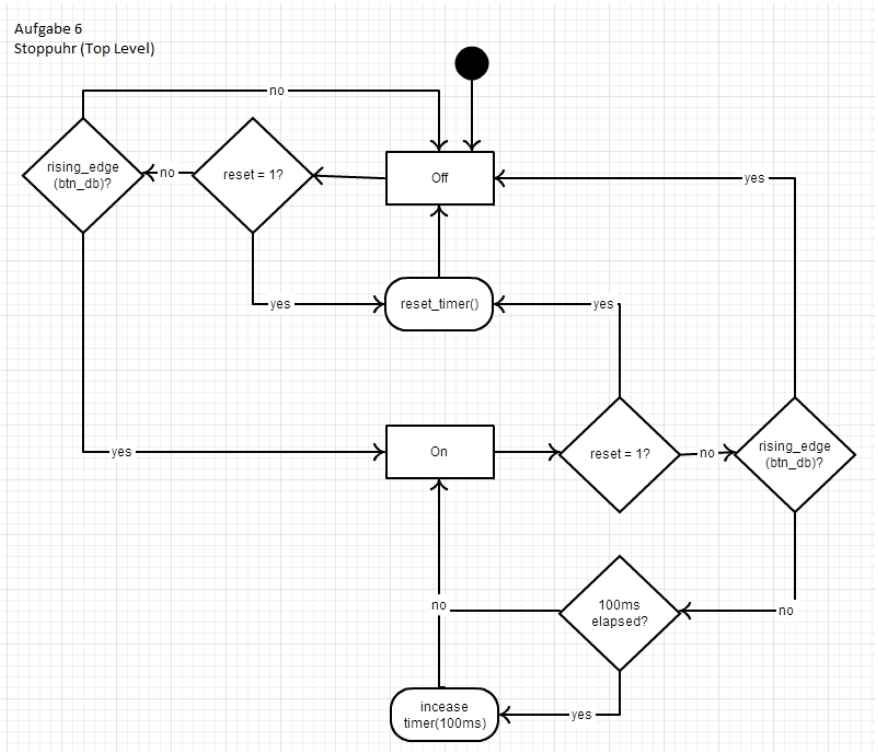
6.1 Entwurf



State-Machine-Charts







Kopplung

Es wird eine synchrone Automatenkopplung mit einem Taktsignal verwendet.

6.2 Auswertung

Ressourcenbedarf

- 163 Logik-Elemente
- davon 121 dedizierte Logik-Elemente
- 35 Pins
- maximale Taktfrequenz von 225 MHz

Anhang

7.1 01-Aufgabe Code

Listing 1: VHDL-Code Decoder.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Decoder is
6
7    port (
8      sw : in std_logic_vector(3 downto 0);
9      cc : out std_logic_vector(7 downto 0));
10
11 end Decoder;
12
13 architecture Dec1 of Decoder is
14 begin
15
16   ----------------------------------------------------------------
17   -- Outputs: 4 bit breites Wort in Hexadezimale 7-Segment Anzeige
18   ----------------------------------------------------------------
19   with sw select
20     cc <= "00000011" when "0000",
21         "10011111" when "0001",
22         "00100101" when "0010",
23         "00001101" when "0011",
24         "10011001" when "0100",
25         "01001001" when "0101",
26         "01000001" when "0110",
27         "00011111" when "0111",
28         "00000001" when "1000",
29         "00001001" when "1001",
30         "00010001" when "1010",
31         "11000001" when "1011",
32         "01100011" when "1100",
33         "10000101" when "1101",
34         "01100001" when "1110",
35         "01110001" when "1111";
36 end dec1;
```

7.2 02-Aufgabe Code

Listing 2: VHDL-Code Hamming.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Hamming is
6
7    port (
8      sw1 : in std_logic_vector(3 downto 0); -- Erstes 4bit Wort
9      sw2 : in std_logic_vector(3 downto 0); -- Zweites 4bit Wort
10     cc : out std_logic_vector(7 downto 0)); -- 7 Segment Ausgabe
11
12 end Hamming;
13
14 architecture ham1 of Hamming is
15   signal xo : std_logic_vector(3 downto 0);
16 begin
17
18   xo <= sw1 xor sw2; -- Jede Stelle nur 1, wenn sich die Woerter an der Stelle unterscheiden
19
20   -- Je nach Anzahl der Einsen im Signal "xo" wird die Ausgabe 0-4 ausgegeben.
21   with xo select
22     cc <= "00000011" when "0000",
23         "10011111" when "0001",
24         "10011111" when "0010",
25         "00100101" when "0011",
26         "10011111" when "0100",
27         "00100101" when "0101",
28         "00100101" when "0110",
29         "00001101" when "0111",
30         "10011111" when "1000",
31         "00100101" when "1001",
32         "00100101" when "1010",
33         "00001101" when "1011",
34         "00100101" when "1100",
35         "00001101" when "1101",
36         "00001101" when "1110",
37         "10011001" when "1111";
38 end ham1;
```

7.3 03-Aufgabe Code

Listing 3: VHDL-Code Schieber.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use Ieee.std_logic_unsigned.all;
5
6  entity Schieber is
7
8    port (
9      clk : in std_logic;
10     rst : in std_logic;
11     ld : out std_logic_vector(9 downto 0)
12     );
13
14
15 end Schieber;
16
17
18 architecture schieb1 of Schieber is
19   component zaehler
20     port (
21       clk : in std_logic;
22       clk_out : out std_logic
23       );
24   end component;
25   signal state : std_logic_vector(9 downto 0) := "0000000001";
26   signal shift : std_logic;
27
28 begin
29   custom_clk : zaehler PORT MAP (clk => clk, clk_out => shift);
30
31   process(clk)
32   begin
33
34     if rising_edge(clk) then
35       if rst = '1' then
36         state <= "0000000001";
37       elsif shift = '1' then
38         state <= state(8 downto 0)&state(9);
39       end if;
40     end if;
41   end process;
42
43   ld <= state;
44
45 end schieb1;
```

Listing 4: VHDL-Code Zaehler.vhdl

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  entity Zaehler is
7
8    port (
9      clk : in std_logic;
10     clk_out : out std_logic
11     );
12
13
14 end Zaehler;
15
16 architecture zae1 of Zaehler is
```

```
17
18
19      signal counter : unsigned(26 downto 0) := (others => '0'); -- Zaehler mod 50.000.000
20      signal state : std_logic := '1';
21  begin
22
23    process(clk, state, counter)
24    begin
25
26      if rising_edge(clk) then
27
28        state <= '0';
29        if counter = to_unsigned(50000000, counter'length) then
30          counter <= (others => '0');
31          state <= '1';
32
33        else
34          counter <= counter + 1;
35        end if;
36
37      end if;
38    end process;
39
40    clk_out <= state;
41  end zae1;
```

7.4 04-Aufgabe Code

Listing 5: VHDL-Code LED.vhdl

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5  use ieee.numeric_std.all;
6
7  entity LED is
8
9    port (
10     clk : in std_logic;
11     btn : in std_logic;
12     ld : out std_logic
13     );
14 end LED;
15
16 architecture led1 of LED is
17   -- Komponente Entprellung fuer das btn-Signal wird eingebunden
18   component Entprellung
19     port (
20       clk : in std_logic;
21       btn_in : in std_logic;
22       btn_out : out std_logic
23       );
24   end component;
25
26   signal led_sig : std_logic := '0';
27   signal btn_led : std_logic;
28   signal btn_out : std_logic;
29   signal btn_out_d : std_logic;
30   signal btn_in_d :std_logic := '0';
31   signal btn_in :std_logic := '0';
32
33   begin
34   custom_entpreller : Entprellung PORT MAP (
35                   clk => clk,
36                   btn_in => btn_in,
37                   btn_out => btn_out);
38   process (btn_led )
39   begin
40       if btn_led = '0' then -- aenderung der led bei uebergang des entprellten btn-signals
             in den aktiven zustand
41       led_sig <= not led_sig;
42       end if;
43   end process;
44
45   -- synchronisierung der signale
46   process (clk)
47   begin
48     if rising_edge(clk) then
49       btn_in <= btn_in_d;
50       btn_in_d <= btn;
51       btn_out_d <= btn_out;
52       btn_led <= btn_out_d;
53     end if;
54   end process;
55
56
57   ld <= not led_sig;
58 end led1;
```

Listing 6: VHDL-Code Entprellung.vhdl

```
1
2  library ieee;
```

```
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5  use ieee.numeric_std.all;
6
7
8  entity Entprellung is
9
10   port (
11     clk : in std_logic;
12     btn_in : in std_logic;
13     btn_out : out std_logic
14     );
15
16 end Entprellung;
17
18 architecture entprel1 of Entprellung is
19
20   signal btn_old : std_logic := '0';
21   signal state : std_logic := '0';
22
23   signal zaehler_state : std_logic :='0';
24   signal zaehler_state_d : std_logic :='0';
25   signal zaehler_reset : std_logic := '0';
26
27
28
29   -- Zaehler-Komponente wird eingebunden
30   component Zaehler
31     port (
32       clk : in std_logic;
33       count_steps : in unsigned(31 downto 0);
34       counter_reset : in std_logic;
35       counter_state : out std_logic
36       );
37   end component;
38
39 begin
40   custom_zaehler : Zaehler PORT MAP (
41               clk => clk,
42               count_steps => to_unsigned(150000, 32),
43               -- zu zaehlende Schritte, bis deaktivierung des counter_state signals
44               counter_state => zaehler_state,
45               counter_reset => zaehler_reset);
46
47   -- entprellung des eingangsignals btn_in
48   process(clk)
49   begin
50     if rising_edge(clk) then
51       if state = '0' then -- falls ausserhalb der prelldauer
52         if btn_in /= btn_old then -- falls das eingangssignal sich aendert, wird der
               entpreller gestartet
53           zaehler_reset <= '1';
54           btn_old <= btn_in;
55         end if;
56       else
57         -- zaehler_reset soll nur einen takt aktiv sein
58         if zaehler_reset = '1' then
59           zaehler_reset <= '0';
60         end if;
61       end if;
62     end if;
63   end process;
64
65   -- synchronisierung des zaehler-zustands
66   process (clk)
67   begin
68     if rising_edge(clk) then
```

```
69        state <= zaehler_state_d;
70        zaehler_state_d <= zaehler_state;
71      end if;
72    end process;
73
74    -- ausgabe des entprellten signals
75    btn_out <= btn_old;
76  end entprel1;
```

Listing 7: VHDL-Code Zaehler.vhdl

```
1
2   library ieee;
3   use ieee.std_logic_1164.all;
4   use Ieee.std_logic_unsigned.all;
5   use ieee.numeric_std.all;
6
7   entity Zaehler is
8
9     port (
10      clk : in std_logic;
11      count_steps : in unsigned(31 downto 0); -- zu zaehlende taktschritte, bis zur
            deaktivierung des counter_state signals
12      counter_reset : in std_logic;
13      counter_state : out std_logic);
14
15
16  end Zaehler;
17
18  architecture zae1 of Zaehler is
19
20    signal reset : std_logic := '0';
21    signal reset_d : std_logic := '0';
22    signal state : std_logic := '0';
23    signal counter : unsigned(31 downto 0) := (others => '0');
24  begin
25
26    process(clk)
27    begin
28
29      if rising_edge(clk) then
30        if reset = '1' then -- zuruecksetzen des zaehlers
31          counter <= (others => '0');
32        end if;
33
34        if counter < count_steps then -- zaehler aktiv
35          state <= '1';
36          counter <= counter + 1;
37        else
38          state <= '0';
39        end if;
40      end if;
41    end process;
42
43    -- synchronisierung des reset-signals
44    process(clk)
45    begin
46      if rising_edge(clk) then
47        reset_d <= counter_reset;
48        reset <= reset_d;
49      end if;
50    end process;
51
52
53    counter_state <= state;
54  end zae1;
```

7.5 05-Aufgabe Code

Listing 8: VHDL-Code hallo.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6  ------------------------------------------------------------------
7  -- Top Level:
8  -- Verbindet den Multiplexer mit 4 Decodern und legt den Ausgang
9  -- je eines Decoders an eine 7 Segment Anzeige
10 ------------------------------------------------------------------
11 entity Hallo is
12
13   port (
14     clk : in std_logic;
15     rst : in std_logic;
16     seg1 : out std_logic_vector(7 downto 0);  --------------
17     seg2 : out std_logic_vector(7 downto 0);  -- 7 Segment
18     seg3 : out std_logic_vector(7 downto 0); -- Ausgaenge
19     seg4 : out std_logic_vector(7 downto 0)); --------------
20
21 end Hallo;
22
23 architecture hello of hallo is
24
25   component Multiplex
26     port (
27       clk : in std_logic;
28       rst : in std_logic;
29       led_out : out std_logic_vector(11 downto 0)
30     );
31   end component;
32
33   component Decoder
34     port (
35       clk : in std_logic;
36       code : in std_logic_vector(2 downto 0);
37       decoded : out std_logic_vector(7 downto 0));
38
39     end component;
40
41     signal dig : std_logic_vector(11 downto 0); -- nimmt 12bit Wort aus dem Multiplexer
           entgegen
42     signal dig0 : std_logic_vector(2 downto 0); ----------------------------------------
43     signal dig1 : std_logic_vector(2 downto 0); -- 4*3bit die je ein Zeichen aus dem 12bit
44     signal dig2 : std_logic_vector(2 downto 0); -- Wort des Multiplexers abzweigen
45     signal dig3 : std_logic_vector(2 downto 0); ----------------------------------------
46
47 begin
48
49   ------------------------------------------------------------------------------
50   -- Multiplexer Ausgang wird an das Signal "dig" angelegt
51   -- Je ein Signal mit je einem Zeichen wird als Eingang eines Decoders angelegt
52   ------------------------------------------------------------------------------
53   mult : Multiplex PORT MAP( clk => clk, rst => rst, led_out => dig);
54   dec0 : Decoder PORT MAP(clk => clk, code => dig0, decoded => seg4);
55   dec1 : Decoder PORT MAP(clk => clk, code => dig1, decoded => seg3);
56   dec2 : Decoder PORT MAP(clk => clk, code => dig2, decoded => seg2);
57   dec3 : Decoder PORT MAP(clk => clk, code => dig3, decoded => seg1);
58
59   -- abzweigen von je 3bit (ein Zeichen) aus dem Multiplexer Ausgang
60   dig0 <= dig(11 downto 9);
61   dig1 <= dig(8 downto 6);
62   dig2 <= dig(5 downto 3);
63   dig3 <= dig(2 downto 0);
```

```
64
65 end hello;
```

Listing 9: VHDL-Code Decoder.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6  entity Decoder is
7
8    port (
9      clk : in std_logic;
10     code : in std_logic_vector(2 downto 0);
11     decoded : out std_logic_vector(7 downto 0));
12 end Decoder;
13
14 architecture decoder1 of Decoder is
15
16   signal decoded_out : std_logic_vector(7 downto 0) := (others => '0'); -- Signal, dass an
         den Ausgang "decoded" angelegt wird
17
18   begin
19
20   ----------------------------------------------------
21   -- Dekodieren eines 3bit breiten Wortes in ein 8bit
22   -- breites Wort fuer die 7 Segment Anzeige
23   ----------------------------------------------------
24   process (clk)
25   begin
26     if rising_edge(clk) then
27       case code is
28          when "000" => decoded_out <= "11111111"; -- _
29          when "001" => decoded_out <= "10010001"; -- H
30          when "010" => decoded_out <= "00010001"; -- A
31          when "011" => decoded_out <= "11100011"; -- L
32          when "100" => decoded_out <= "00000011"; -- O
33          when others => decoded_out <= "11111111";
34       end case;
35     end if;
36   end process;
37
38   decoded <= decoded_out;
39
40 end decoder1;
```

Listing 10: VHDL-Code Multiplex.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5
6  entity Multiplex is
7
8    port (
9      clk : in std_logic;
10     rst : in std_logic;
11     led_out : out std_logic_vector(11 downto 0) -- 12bit breiter Ausgang (3bit je Zeichen)
12     );
13
14 end Multiplex;
15
16 -- 000: _
17 -- 001: H
18 -- 010: A
```

```
19  -- 011: L
20  -- 100: O
21
22  architecture multi of Multiplex is
23  -- _ _ _ _ H A L L O _ _ _
24    signal tex : std_logic_vector(35 downto 0) := "000000000000001010011011100000000000"; --
          kompletter Schriftzug der einmal durchlaufen wird
25    signal counter : unsigned(24 downto 0) := (others => '0'); -- Zaehlersignal (mod
          25.000.000)
26    signal mul : unsigned(3 downto 0); -- Steuersignal Multiplexer (mod 10) : 9 moegliche
          12bit breite Teilworte des kompletten Schriftzugs
27
28  begin
29
30    ---------------------------------------------------------------------
31    -- Inkrementieren des Steuersignals "mul" alle 25.000.000 Takte
32    ---------------------------------------------------------------------
33    process(clk)
34    begin
35      if rising_edge(clk) then
36        if(rst = '1') then
37          counter <= (others => '0');
38          mul <= (others => '0');
39        elsif(counter = "101111101011100000111111") then
40          counter <= (others => '0');
41          mul <= mul + 1;
42          if(mul = "1000") then
43            mul <= "0000";
44          end if;
45        else
46          counter <= counter + 1;
47        end if;
48      end if;
49    end process;
50
51    ---------------------------------------------------------------------------------------------------------------
52    -- Je nach Steuersignal "mul" wird ein anderes 12bit breites Teilwort des kompletten
          Schriftzugs ausgegeben
53    ---------------------------------------------------------------------------------------------------------------
54    with mul select
55      led_out <=  tex(35 downto 24) when "0000",
56              tex(32 downto 21) when "0001",
57              tex(29 downto 18) when "0010",
58              tex(26 downto 15) when "0011",
59              tex(23 downto 12) when "0100",
60              tex(20 downto 9) when "0101",
61              tex(17 downto 6) when "0110",
62              tex(14 downto 3) when "0111",
63              tex(11 downto 0) when "1000",
64              (others => '0') when others;
65
66  end multi;
```

7.6 06-Aufgabe Code

Listing 11: VHDL-Code Stoppuhr.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6  entity Stoppuhr is
7
8    port (
9      clk : in std_logic;
10     rst : in std_logic;
11     onoff : in std_logic;
12     seg1 : out std_logic_vector(7 downto 0);
13     seg2 : out std_logic_vector(7 downto 0);
14     seg3 : out std_logic_vector(7 downto 0);
15     seg4 : out std_logic_vector(7 downto 0));
16
17 end Stoppuhr;
18
19 architecture uhr of Stoppuhr is
20
21   -- Zaehler: gibt jede Zehntelsekunde einen Peak aus
22   component Zaehler
23     port (
24       clk : in std_logic;
25       zaehler_time : in unsigned(31 downto 0);
26       zaehler_on : in std_logic;
27       peak_out : out std_logic
28       );
29   end component;
30
31   -- entprellt das Eingangssignal des (on/off) Buttons
32   component EntprellAutomat
33     port (
34       clk : in std_logic;
35       btn : in std_logic;
36       btnout : out std_logic
37       );
38   end component;
39
40   -- 4 Decoder: Je ein Decoder dekodiert eine Stelle der aktuellen Zeit fuer die 7-Segment
         Anzeige
41   component Decoder
42     port (
43       clk : in std_logic;
44       code : in std_logic_vector(3 downto 0);
45       decoded : out std_logic_vector(7 downto 0)
46       );
47   end component;
48
49   -- Steuersignal und Ausgabesignal des Zaehlers
50   signal timer_on, peak : std_logic := '0';
51
52   -- Je ein Signal fuer je eine Stelle der aktuellen Zeit
53   signal min, sec1, sec2, ms : unsigned(3 downto 0) := (others => '0');
54
55   -- Steuersignal fuer die Stoppuhr und Signale fuer den entprellten Button, sowie des alten
         Signalpegels des Buttons
56   signal running, onoff_db, onoff_old : std_logic := '0';
57
58   -- Signale fuer die 7-Segment Anzeige
59   signal segMin, segSec1, segSec2, segMs : std_logic_vector(7 downto 0) := (others => '0');
60
61
62 begin
```

```
63
64  zaehl : Zaehler PORT MAP (clk => clk,
65                  zaehler_time => to_unsigned(5000000, 32),
66                  zaehler_on => timer_on,
67                  peak_out => peak);
68
69  prell : EntprellAutomat PORT MAP (
70                  clk => clk,
71                  btn => onoff,
72                  btnout => onoff_db);
73
74  -- Jeder Decoder dekodiert eine Stelle der aktuellen Zeit
75  dec0 : Decoder PORT MAP(clk => clk, code => std_logic_vector(min), decoded => segMin);
76  dec1 : Decoder PORT MAP(clk => clk, code => std_logic_vector(sec1), decoded => segSec1);
77  dec2 : Decoder PORT MAP(clk => clk, code => std_logic_vector(sec2), decoded => segSec2);
78  dec3 : Decoder PORT MAP(clk => clk, code => std_logic_vector(ms), decoded => segMs);
79
80
81  process(clk)
82  begin
83    if rising_edge(clk) then
84      if(rst = '1') then           -- aktiver Reset setzt alles zurueck und stoppt die Uhr
85        running <= '0';
86        timer_on <= '0';
87        min <= (others => '0');
88        sec1 <= (others => '0');
89        sec2 <= (others => '0');
90        ms <= (others => '0');
91      else
92        onoff_old <= onoff_db;
93        if(onoff_db = '1' and onoff_db /= onoff_old) then
              ---------------------------------------
94          running <= not running;        -- on/off umschalten wenn btn gedrueckt
95          timer_on <= not timer_on;       ---------------------------------------
96        end if;
97
98        if(running = '1') then         -- Wenn die Uhr laeuft...
99          if(peak = '1') then         -- ...und der Zaehler einen Peak ausgibt...
100           if(ms = to_unsigned(9, 4)) then     ------------------------------------------
101             if(sec2 = to_unsigned(9, 4)) then   --
102               if(sec1 = to_unsigned(5, 4)) then  --
103                 if(min = to_unsigned(9, 4)) then --
104                   min <= (others => '0');   --
105                   sec1 <= (others => '0');  --
106                   sec2 <= (others => '0');  --
107                   ms <= (others => '0');   --
108                 else        --
109                   min <= min + 1;     -- ...erhoehe die aktuelle Zeit
110                   sec1 <= (others => '0');  -- um eine Zehntelsekunde (mod 10 Minuten)
111                   sec2 <= (others => '0');  --
112                   ms <= (others => '0');   --
113                 end if;        --
114               else         --
115                 sec1 <= sec1 + 1;     --
116                 sec2 <= (others => '0');   --
117                 ms <= (others => '0');    --
118               end if;         --
119             else          --
120               sec2 <= sec2 + 1;      --
121               ms <= (others => '0');     --
122             end if;          --
123           else           --
124             ms <= ms + 1;        --
125           end if;           --
126         end if;            --
127       end if;             ------------------------------------------
128
```

```
129       end if;
130     end if;
131   end process;
132 
133   seg4 <= segMin and "11111110"; -- Punkt der 7-Segment Anzeige aktivieren
134   seg3 <= segSec1;
135   seg2 <= segSec2 and "11111110"; -- Punkt der 7-Segment Anzeige aktivieren
136   seg1 <= segMS;
137 
138 end uhr;
```

Listing 12: VHDL-Code Decoder.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5 
6  entity Decoder is
7 
8    port (
9      clk : in std_logic;
10     code : in std_logic_vector(3 downto 0);
11     decoded : out std_logic_vector(7 downto 0));
12 end Decoder;
13 
14 architecture decoder1 of Decoder is
15 
16   signal decoded_out : std_logic_vector(7 downto 0) := (others => '0'); -- Signal, dass an
         den Ausgang "decoded" angelegt wird
17 
18   begin
19 
20   -------------------------------------------------
21   -- Dekodieren eines 4bit breiten Wortes in ein 8bit
22   -- breites Wort fuer die 7 Segment Anzeige
23   -------------------------------------------------
24   process (clk)
25   begin
26     if rising_edge(clk) then
27       case code is
28          when "0000" => decoded_out <= "00000011";  -- 0
29          when "0001" => decoded_out <= "10011111"; -- 1
30          when "0010" => decoded_out <= "00100101";  -- 2
31          when "0011" => decoded_out <= "00001101";  -- 3
32          when "0100" => decoded_out <= "10011001";  -- 4
33          when "0101" => decoded_out <= "01001001";  -- 5
34          when "0110" => decoded_out <= "01000001"; -- 6
35          when "0111" => decoded_out <= "00011111"; -- 7
36          when "1000" => decoded_out <= "00000001"; -- 8
37          when "1001" => decoded_out <= "00001001"; -- 9
38          when others => decoded_out <= "11111111"; -- error
39       end case;
40     end if;
41   end process;
42 
43   decoded <= decoded_out;
44 
45 end decoder1;
```

Listing 13: VHDL-Code EntprellAutomat.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4 
5 
```

```
6  entity EntprellAutomat is
7
8    port (
9      clk : in std_logic;
10     btn : in std_logic;
11     btnout : out std_logic
12     );
13
14 end EntprellAutomat;
15
16 architecture prell of EntprellAutomat is
17
18   component Zaehler
19     port (
20       clk : in std_logic;
21       zaehler_time : in unsigned(31 downto 0);
22       zaehler_on : in std_logic;
23       peak_out : out std_logic
24     );
25   end component;
26
27   type zustaende is (idle, count); -- 2 Zustaende, idle = button betaetigen moeglich, count
         = 3ms warten (bis 150.000 hochzaehlen bei 50Mhz)
28   attribute enum_encoding : string;
29   attribute enum_encoding of zustaende : type is "1 0";
30   signal z_alt, z_neu : zustaende := idle; -- alter und neuer Zustand des Automaten
31   signal btn_s, btn_output : std_logic := '0'; -- Synchronisiertes (Eingangs)Buttonsignal
         und Ausgangssignal des Automaten
32   signal btn_old : std_logic := '0'; -- Speichern des vorherigen "btn_s" Pegels
33   signal btn2 : std_logic := '1'; -- Synchronisierungssignal fuer den Button
34   signal timer_on : std_logic := '0'; -- Steuerung des externen Zaehler Moduls
35   signal peak : std_logic := '0'; -- Ausgabe des externen Zaehler Moduls
36
37 begin
38
39   -- enthaltener Zaehler, der (falls aktiviert) alle 3ms fuer einen Takt eine eins an das
         Signal "peak" ausgibt
40   timer : Zaehler PORT MAP (clk => clk, zaehler_time => to_unsigned(150000,32), zaehler_on
         => timer_on, peak_out => peak);
41
42   -- Synchronisieren des Eingabesignals (Button) und Speichern des alten "btn_s" Pegels
43   process(clk)
44   begin
45     if rising_edge(clk) then
46       btn2 <= btn;
47       btn_old <= btn_s;
48       btn_s <= not btn2;
49     end if;
50   end process;
51
52   -- Uebernehmen und berechnen des neuen Zustandes
53   process(clk)
54   begin
55     if rising_edge(clk) then
56       z_alt <= z_neu;
57       case z_alt is
58         when idle =>  if(btn_s /= btn_old) then  -- Wechseln in "count" und starten des
               Zaehlers, sobald sich btn_s aendert
59                 btn_output <= btn_s;
60                 z_neu <= count;
61                 timer_on <= '1';
62               end if;
63         when count => if(peak = '1') then    -- Wechseln zurueck in "idle", sobald der
               Zaehler eine eins an "peak" anlegt (3ms vergangen)
64                 z_neu <= idle;
65                 timer_on <= '0';
66               end if;
```

```
67        end case;
68      end if;
69    end process;
70
71    btnout <= btn_output;
72  end prell;
```

Listing 14: VHDL-Code Zaehler.vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5
6   entity Zaehler is
7
8     port (
9       clk : in std_logic;
10      zaehler_time : in unsigned(31 downto 0); -- steuert wieviele Takte der Zaehler zaehlen
            soll, bis er einen peak ausgeben soll. (Zaehler zaehlt mod zaehler_time)
11      zaehler_on : in std_logic; -- Steuersignal, bei 1 laeuft der Zaehler, bei 0 wird der
            Zaehler gestoppt und zurueckgesetzt
12      peak_out : out std_logic -- gibt fuer einen Takt eine eins aus, sobald der durch
            "zaehler_time" angelegte Wert erreicht ist
13      );
14
15  end Zaehler;
16
17  architecture timer of Zaehler is
18
19    signal counter : unsigned(31 downto 0) := (others => '0'); -- Zaehlsignal
20    signal peak : std_logic := '0'; -- Signal, dass an den Ausgang "peak_out" gegeben wird
21
22  begin
23
24    ---------------------------------------------------------------------------------
25    -- Liegt "zaehler_on" auf eins, wird das Signal "counter" inkrementiert.
26    -- Wird der durch "zaehler_time" angegebene Maximalwert erreicht wird fuer einen Takt
27    -- eine 1 an das Signal "peak" ausgegeben und "counter" auf 0 zurueckgesetzt
28    -- Liegt "zaehler_on" auf null, wird "counter" auf 0 gesetzt und nicht hochgezaehlt.
29    ---------------------------------------------------------------------------------
30    process(clk)
31    begin
32      if rising_edge(clk) then
33        peak <= '0';
34        if(zaehler_on = '1') then
35          if(counter = zaehler_time - 1) then
36            counter <= (others => '0');
37            peak <= '1';
38          else
39            counter <= counter + 1;
40          end if;
41        else
42          counter <= (others => '0');
43        end if;
44      end if;
45    end process;
46
47    peak_out <= peak;
48
49  end timer;
```