Protokoll zum Praktikum Programmierbare Schaltkreise

Fakultät Informatik
TU Dresden

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

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

24. 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

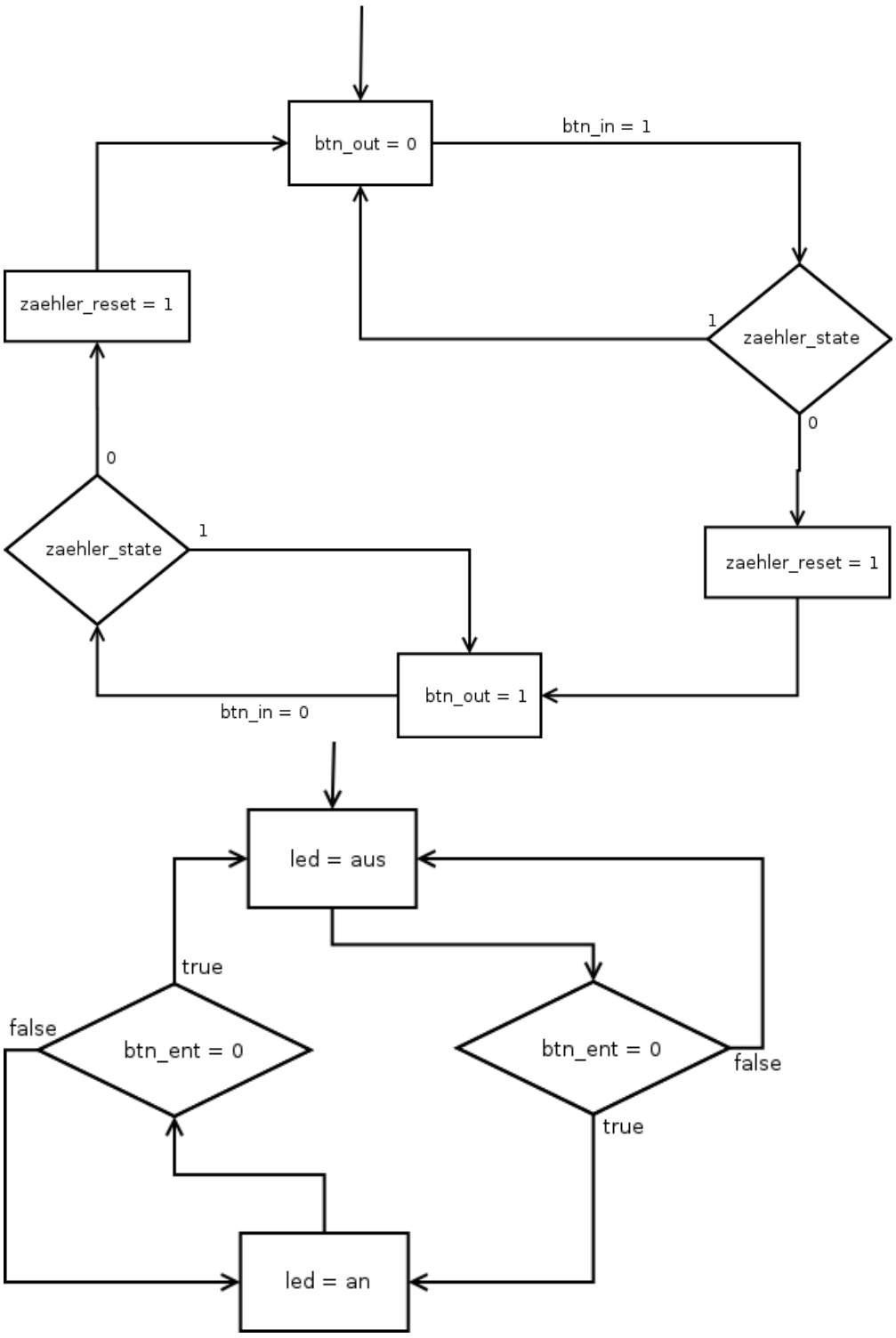
Ressourcenbedarf

- 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)

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)

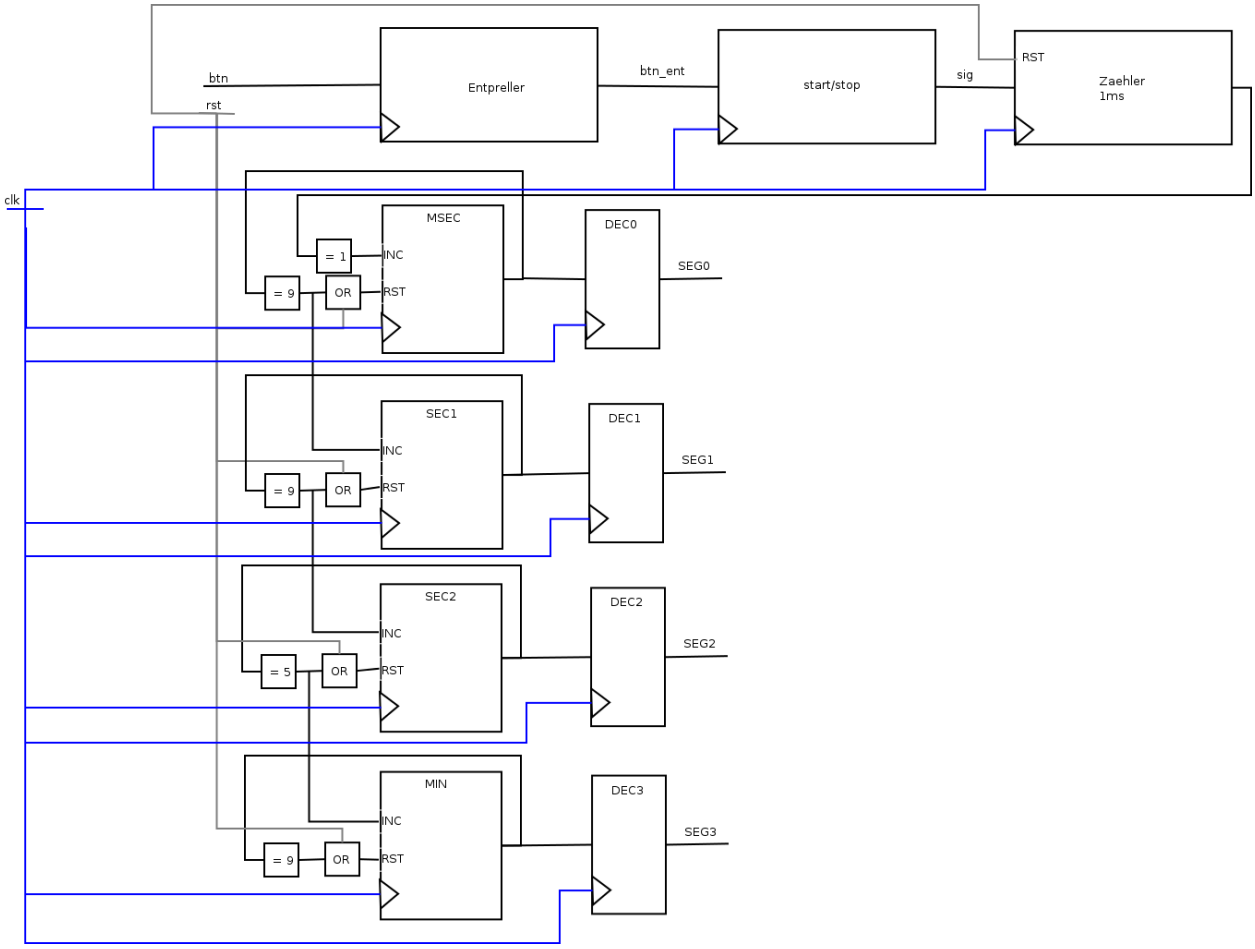
5.2 Auswertung

Ressourcenbedarf

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

Aufgabe 6 - Stoppuhr

6.1 Entwurf



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

7.2 02-Aufgabe Code

Listing 2: VHDL-Code Hamming.vhdl

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  entity Hamming is
7
8    port (
9      sw1 : in std_logic_vector(3 downto 0);
10     sw2 : in std_logic_vector(3 downto 0);
11     cc : out std_logic_vector(7 downto 0));
12
13 end Hamming;
14
15 architecture ham1 of Hamming is
16   signal xo : std_logic_vector(3 downto 0);
17 begin
18
19   xo <= sw1 xor sw2;
20
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  entity Hallo is
7
8    port (
9      clk : in std_logic;
10     rst : in std_logic;
11     seg1 : out std_logic_vector(7 downto 0);
12     seg2 : out std_logic_vector(7 downto 0);
13     seg3 : out std_logic_vector(7 downto 0);
14     seg4 : out std_logic_vector(7 downto 0));
15
16 end Hallo;
17
18 architecture hello of hallo is
19
20   component Multiplex
21     port (
22       clk : in std_logic;
23       rst : in std_logic;
24       led_out : out std_logic_vector(11 downto 0)
25     );
26   end component;
27
28   component Decoder
29     port (
30       clk : in std_logic;
31       code : in std_logic_vector(2 downto 0);
32       decoded : out std_logic_vector(7 downto 0));
33
34     end component;
35
36     signal dig : std_logic_vector(11 downto 0);
37     signal dig0 : std_logic_vector(2 downto 0);
38     signal dig1 : std_logic_vector(2 downto 0);
39     signal dig2 : std_logic_vector(2 downto 0);
40     signal dig3 : std_logic_vector(2 downto 0);
41
42 begin
43
44   mult : Multiplex PORT MAP( clk => clk, rst => rst, led_out => dig);
45   dec0 : Decoder PORT MAP(clk => clk, code => dig0, decoded => seg4);
46   dec1 : Decoder PORT MAP(clk => clk, code => dig1, decoded => seg3);
47   dec2 : Decoder PORT MAP(clk => clk, code => dig2, decoded => seg2);
48   dec3 : Decoder PORT MAP(clk => clk, code => dig3, decoded => seg1);
49
50
51   dig0 <= dig(11 downto 9);
52   dig1 <= dig(8 downto 6);
53   dig2 <= dig(5 downto 3);
54   dig3 <= dig(2 downto 0);
55
56 end hello;
```

Listing 9: VHDL-Code Decoder.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 Decoder is
8
9    port (
10     clk : in std_logic;
11     code : in std_logic_vector(2 downto 0);
12     decoded : out std_logic_vector(7 downto 0));
13 end Decoder;
14
15 architecture decoder1 of Decoder is
16
17   signal decoded_out : std_logic_vector(7 downto 0) := (others => '0');
18
19   begin
20
21   process (clk)
22   begin
23     if rising_edge(clk) then
24       case code is
25          when "000" => decoded_out <= "11111111";
26          when "001" => decoded_out <= "10010001";
27          when "010" => decoded_out <= "00010001";
28          when "011" => decoded_out <= "11100011";
29          when "100" => decoded_out <= "00000011";
30          when others => decoded_out <= "11111111";
31       end case;
32     end if;
33   end process;
34
35   decoded <= decoded_out;
36
37 end decoder1;
```

Listing 10: VHDL-Code Multiplex.vhdl

```
1  --
2  -- Copyright (c) 20013
3  -- Technische Universitaet Dresden, Dresden, Germany
4  -- Faculty of Computer Science
5  -- Institute for Computer Engineering
6  -- Chair for VLSI-Design, Diagnostics and Architecture
7  --
8  -- For internal educational use only.
9  -- The distribution of source code or generated files
10 -- is prohibited.
11 --
12
13 --
14 -- Entity: Example
15 -- Author(s): Martin Zabel, Matthias Haesing
16 --
17 -- Simple example for the Terasic DE0 board.
18 --
19 -- Revision: $Revision: 1.1 $
20 -- Last change: $Date: 2013-10-09 12:49:38 $
21 --
22
23 library ieee;
24 use ieee.std_logic_1164.all;
25 use ieee.numeric_std.all;
26
27
28 entity Multiplex is
29
30   port (
31     clk : in std_logic;
32     rst : in std_logic;
```

```
33      led_out : out std_logic_vector(11 downto 0)
34      );
35
36  end Multiplex;
37
38  -- 000: _
39  -- 001: H
40  -- 010: A
41  -- 011: L
42  -- 100: O
43
44  architecture multi of Multiplex is
45  -- _ _ _ _ H A L L O _ _ _
46    signal tex : std_logic_vector(35 downto 0) := "000000000000001010011011100000000000";
47    signal counter : unsigned(24 downto 0) := (others => '0');
48    signal mul : unsigned(3 downto 0);
49
50  begin
51
52    process(clk)
53    begin
54      if rising_edge(clk) then
55        if(rst = '1') then
56          counter <= (others => '0');
57          mul <= (others => '0');
58        elsif(counter = "10111110101111100000111111") then
59      -- elsif(counter = "00000000000011000000111111") then
60          counter <= (others => '0');
61          mul <= mul + 1;
62          if(mul = "1000") then
63            mul <= "0000";
64          end if;
65        else
66          counter <= counter + 1;
67        end if;
68      end if;
69    end process;
70
71    with mul select
72      led_out <=  tex(35 downto 24) when "0000",
73              tex(32 downto 21) when "0001",
74              tex(29 downto 18) when "0010",
75              tex(26 downto 15) when "0011",
76              tex(23 downto 12) when "0100",
77              tex(20 downto 9) when "0101",
78              tex(17 downto 6) when "0110",
79              tex(14 downto 3) when "0111",
80              tex(11 downto 0) when "1000",
81              (others => '0') when others;
82
83  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   component Zaehler
22     port (
23       clk : in std_logic;
24       zaehler_time : in unsigned(31 downto 0);
25       zaehler_on : in std_logic;
26       peak_out : out std_logic
27       );
28   end component;
29
30   component EntprellAutomat
31     port (
32       clk : in std_logic;
33       btn : in std_logic;
34       btnout : out std_logic
35       );
36   end component;
37
38   component Decoder
39     port (
40       clk : in std_logic;
41       code : in std_logic_vector(3 downto 0);
42       decoded : out std_logic_vector(7 downto 0)
43       );
44   end component;
45
46   signal timer_on, peak : std_logic := '0';
47   signal min, sec1, sec2, ms : unsigned(3 downto 0) := (others => '0');
48   signal running, onoff_db, onoff_old : std_logic := '0';
49   signal segMin, segSec1, segSec2, segMs : std_logic_vector(7 downto 0) := (others => '0');
50
51
52 begin
53
54   zaehl : Zaehler PORT MAP (clk => clk,
55                   zaehler_time => to_unsigned(5000000, 32),
56                   zaehler_on => timer_on,
57                   peak_out => peak);
58
59   prell : EntprellAutomat PORT MAP (
60                   clk => clk,
61                   btn => onoff,
62                   btnout => onoff_db);
63
64   dec0 : Decoder PORT MAP(clk => clk, code => std_logic_vector(min), decoded => segMin);
```

```
65    dec1 : Decoder PORT MAP(clk => clk, code => std_logic_vector(sec1), decoded => segSec1);
66    dec2 : Decoder PORT MAP(clk => clk, code => std_logic_vector(sec2), decoded => segSec2);
67    dec3 : Decoder PORT MAP(clk => clk, code => std_logic_vector(ms), decoded => segMs);
68
69
70    process(clk)
71    begin
72      if rising_edge(clk) then
73        if(rst = '1') then
74          running <= '0';
75          timer_on <= '0';
76          min <= (others => '0');
77          sec1 <= (others => '0');
78          sec2 <= (others => '0');
79          ms <= (others => '0');
80        else
81          onoff_old <= onoff_db;
82          -------------------------
83          -- on/off umschalten wenn btn gedrueckt
84          -------------------------
85          if(onoff_db = '1' and onoff_db /= onoff_old) then
86            running <= not running;
87            timer_on <= not timer_on;
88          end if;
89
90          if(running = '1') then
91            if(peak = '1') then
92              if(ms = to_unsigned(9, 4)) then
93                if(sec2 = to_unsigned(9, 4)) then
94                  if(sec1 = to_unsigned(5, 4)) then
95                    if(min = to_unsigned(9, 4)) then
96                      min <= (others => '0');
97                      sec1 <= (others => '0');
98                      sec2 <= (others => '0');
99                      ms <= (others => '0');
100                   else
101                     min <= min + 1;
102                     sec1 <= (others => '0');
103                     sec2 <= (others => '0');
104                     ms <= (others => '0');
105                   end if;
106                 else
107                   sec1 <= sec1 + 1;
108                   sec2 <= (others => '0');
109                   ms <= (others => '0');
110                 end if;
111               else
112                 sec2 <= sec2 + 1;
113                 ms <= (others => '0');
114               end if;
115             else
116               ms <= ms + 1;
117             end if;
118           end if;
119         end if;
120
121       end if;
122     end if;
123   end process;
124
125   seg4 <= segMin and "11111110";
126   seg3 <= segSec1;
127   seg2 <= segSec2 and "11111110";
128   seg1 <= segMS;
129
130 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  -- Decoder bekommt einen 3bit breiten Code fuer ein Zeichen und gibt den 8bit breiten Code
       fuer die 7-Segment Anzeige des Zeichens aus
7
8  entity Decoder is
9
10   port (
11     clk : in std_logic;
12     code : in std_logic_vector(3 downto 0);
13     decoded : out std_logic_vector(7 downto 0));
14 end Decoder;
15
16 architecture decoder1 of Decoder is
17
18   signal decoded_out : std_logic_vector(7 downto 0) := (others => '0');
19
20   begin
21
22   process (clk)
23   begin
24     if rising_edge(clk) then
25       case code is
26          when "0000" => decoded_out <= "00000011";  -- 0
27          when "0001" => decoded_out <= "10011111"; -- 1
28          when "0010" => decoded_out <= "00100101";  -- 2
29          when "0011" => decoded_out <= "00001101";  -- 3
30          when "0100" => decoded_out <= "10011001";  -- 4
31          when "0101" => decoded_out <= "01001001";  -- 5
32          when "0110" => decoded_out <= "01000001"; -- 6
33          when "0111" => decoded_out <= "00011111"; -- 7
34          when "1000" => decoded_out <= "00000001"; -- 8
35          when "1001" => decoded_out <= "00001001"; -- 9
36          when others => decoded_out <= "11111111"; -- error
37       end case;
38     end if;
39   end process;
40
41   decoded <= decoded_out;
42
43 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;
31  signal btn_s, btn_old, timer_on, btn_output : std_logic := '0';
32  signal btn2 : std_logic := '1';
33  signal peak : std_logic := '0';
34
35  begin
36
37  timer : Zaehler PORT MAP (clk => clk, zaehler_time => to_unsigned(150000,32), zaehler_on
        => timer_on, peak_out => peak);
38
39  process(clk)
40  begin
41    if rising_edge(clk) then
42      btn2 <= btn;
43      btn_old <= btn_s;
44      btn_s <= not btn2;
45    end if;
46  end process;
47
48
49  process(clk)
50  begin
51    if rising_edge(clk) then
52      z_alt <= z_neu;
53      case z_alt is
54        when idle =>  if(btn_s /= btn_old) then
55                    btn_output <= btn_s;
56                    z_neu <= count;
57                    timer_on <= '1';
58                  end if;
59        when count => if(peak = '1') then
60                    z_neu <= idle;
61                    timer_on <= '0';
62                  end if;
63      end case;
64    end if;
65  end process;
66  btnout <= btn_output;
67  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);
11      zaehler_on : in std_logic;
12      peak_out : out std_logic
13      );
14
```

```
15 end Zaehler;
16
17 architecture timer of Zaehler is
18
19   signal counter : unsigned(31 downto 0) := (others => '0');
20   signal peak : std_logic := '0';
21
22 begin
23
24   process(clk)
25   begin
26     if rising_edge(clk) then
27       peak <= '0';
28       if(zaehler_on = '1') then
29         if(counter = zaehler_time - 1) then
30           counter <= (others => '0');
31           peak <= '1';
32         else
33           counter <= counter + 1;
34         end if;
35       else
36         counter <= (others => '0');
37       end if;
38     end if;
39   end process;
40
41   peak_out <= peak;
42
43 end timer;
```