Programable Logic Circuts
Lab 05

Sølve Kjelseth

September 5, 2025

0. Introduction

This is the fifth report in this course, detailing the completion of the fifth lab exercise. With the purpose is to investigate generics and counters.

Note: As always, the LaTeX code is open source, see my GitHub 

0.0. Table of Contents

0.1. List of Figures

1. Part 1

This Part is about making a generic counter with a desired rollover value. Then instancing it to check the function.

1.0. Solving

Solving this task was done by first making a generic counter and then add a clause to reset the counter when the rollover value was reached and output a signal. Then the counter was instanced in the TLE with a generic map in addition to the port map to show the function.

1.1. Code

Code 1.0: TLE for testing counter and rollover

VHDL

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

entity L05P01 is
    generic(output_size : natural := 5);
    port(KEY  : in  std_logic_vector(3 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end L05P01;


architecture structural of L05P01 is

    component modulo_k_counter
        generic(n : natural := 4;
                k : integer := 10);
        port(Clk : in  std_logic;
             Rst : in  std_logic;
             Hop : out std_logic;
             Num : out std_logic_vector(n-1 downto 0));
    end component;

    signal display  : std_logic_vector(output_size-1 downto 0);
    signal rollover : std_logic;
    signal Clk, Rst : std_logic;

begin

    LEDR(output_size-1 downto 0) <= display;
    LEDR(8 downto output_size) <= (others => '0');
    LEDR(9) <= rollover;

    Clk <= KEY(3);
    Rst <= KEY(2);

    K20: modulo_k_counter
            generic map
                (n => output_size,
                 k => 20)
            port map
                (Clk => Clk,
                 Rst => Rst,
                 Hop => rollover,
                 Num => display);

end structural;
```

Code 1.1: Generic counter component used in the TLE

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity modulo_k_counter is
    generic(n : natural := 4;
            k : natural := 10);
    port(Clk : in  std_logic;
         Rst : in  std_logic;
         Hop : out std_logic;
         Num : out std_logic_vector(n-1 downto 0));
end modulo_k_counter;

architecture behavioural of modulo_k_counter is

    signal Val : std_logic_vector(n-1 downto 0);

begin
    process(Clk, Rst)
    begin
        if rising_edge(Clk) then
            if Val = k-1 then
                Val <= (others => '0');
                Hop <= '1';
            else
                Val <= Val + 1;
                Hop <= '0';
            end if;
        end if;
        if Rst = '0' then
            Val <= (others => '0');
            Hop <= '0';
        end if;
        Num <= Val;
    end process;
end behavioural;
```

1.2. RTL

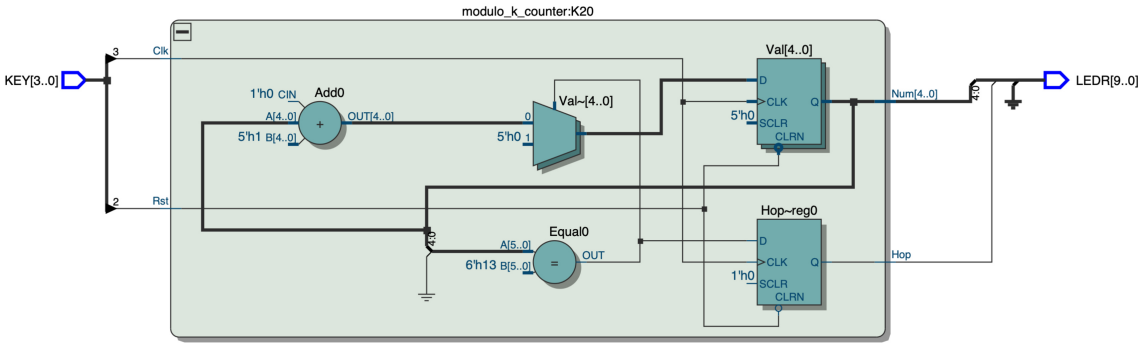
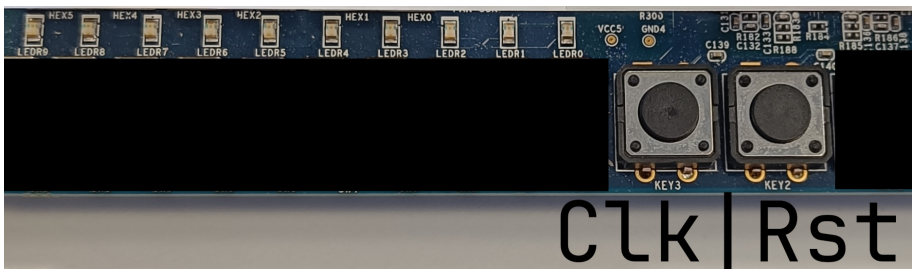


Figure 1.0: RTL of the TLE with counter

1.3. Results



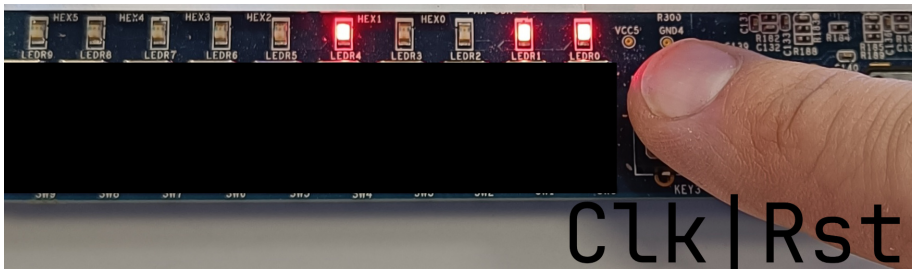
(a) Default state

(b) Clock cycle 1, counts up

(c) Clock cycle 2, counts up

(d) Clock cycle 3, counts up

Figure 1.1: Function test, works as expected



(e) Clock cycle 19, counts up

(f) Clock cycle 20, resets to 0 and sets rollover flag

(g) Clock cycle 21, counts up and clears rollover flag

(h) Asynchronous reset pressed, sets counter to 0 and clears rollover flag

Figure 1.1: Function test, works as expected

2. Part 2

This is part is about making a clock using generic counters and making the minutes assignable with the switches.

2.0. Solving

Solving this task was done by first modifying the generic counter from part 1 to add additional I/O. A set enable and set value input, and a count number output. Then the 7 segment decoder, as used in multiple previous lab tasks was used inside a new BCD display module. This essentially splits up the decimal 10's place and 1's place to two different numbers and shows them on each respective display. Then a TLE was created for running the instances of a timer for each hundredths, seconds and minutes and making each rollover value the clock input for the next counter. Instances for the BCD display was made with signals to transfer each of the three to the HEX display.

2.1. Code

Code 2.0: TLE of the clock

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

entity L05P02 is
    port(CLOCK_50   : in  std_logic;
         SW         : in  std_logic_vector(7 downto 0);
         KEY        : in  std_logic_vector(3 downto 0);
         LEDR       : out std_logic_vector(9 downto 0);
         HEX0, HEX1 : out std_logic_vector(6 downto 0);
         HEX2, HEX3 : out std_logic_vector(6 downto 0);
         HEX4, HEX5 : out std_logic_vector(6 downto 0));
end L05P02;


architecture behavioural of L05P02 is

    component modulo_k_counter
        generic(n : natural := 4;
                k : integer := 10);
        port(Clk : in  std_logic;
             Set : in  std_logic;
             Svl : in  std_logic_vector(n-1 downto 0);
             Rst : in  std_logic;
             Hop : out std_logic;
             Num : out std_logic_vector(n-1 downto 0));
    end component;

    component display_BCD
        port(Num    : in  std_logic_vector(6 downto 0);
             HEXlsd : out std_logic_vector(6 downto 0);
             HEXmsd : out std_logic_vector(6 downto 0));
    end component;

    signal Clk, Clk_h, Clk_s, Clk_m, overflow : std_logic := '0';
    signal Set, Rst, disabled, startup        : std_logic := '1';
    signal unused_num : std_logic_vector(18 downto 0) := (others => '0');
    signal hundredths : std_logic_vector(6  downto 0) := (others => '0');
    signal seconds    : std_logic_vector(6  downto 0) := (others => '0');
    signal minutes    : std_logic_vector(6  downto 0) := (others => '0');
    signal minutes_in : std_logic_vector(6  downto 0) := (others => '0');

begin
    process(CLOCK_50, startup)
    begin
        if rising_edge(CLOCK_50) then
            if startup = '1' then
                Rst <= '0';
                startup <= '0';
            else
                Rst <= KEY(1);
            end if;
        end if;
    end process;

    Clk <= CLOCK_50 and KEY(2);
    Set <= KEY(3);
    minutes_in <= SW(6 downto 0);
```

```
58      LEDR(9 downto 1) <= (others => '0');
59      LEDR(0) <= overflow;
60
61      K500000_divider: modulo_k_counter
62          generic map
63              (n => 19,
64               k => 500000)
65          port map
66              (Clk => Clk,
67               Set => disabled,
68               Svl => unused_num,
69               Rst => Rst,
70               Hop => Clk_h,
71               Num => unused_num);
72
73      K100_hundredths: modulo_k_counter
74          generic map
75              (n => 7,
76               k => 100)
77          port map
78              (Clk => Clk_h,
79               Set => disabled,
80               Svl => hundredths,
81               Rst => Rst,
82               Hop => Clk_s,
83               Num => hundredths);
84
85      K60_seconds: modulo_k_counter
86          generic map
87              (n => 7,
88               k => 60)
89          port map
90              (Clk => Clk_s,
91               Set => disabled,
92               Svl => seconds,
93               Rst => Rst,
94               Hop => Clk_m,
95               Num => seconds);
96
97      K60_minutes: modulo_k_counter
98          generic map
99              (n => 7,
100              k => 60)
101         port map
102             (Clk => Clk_m,
103              Set => Set,
104              Svl => minutes_in,
105              Rst => Rst,
106              Hop => Overflow,
107              Num => minutes);
108
109     D_hundredths: display_BCD
110         port map
111             (Num    => hundredths,
112              HEXlsd => HEX0,
113              HEXmsd => HEX1);
114
115     D_seconds: display_BCD
116         port map
117             (Num    => seconds,
118              HEXlsd => HEX2,
```

```
             HEXmsd => HEX3);

    D_minutes: display_BCD
        port map
            (Num    => minutes,
             HEXlsd => HEX4,
             HEXmsd => HEX5);

end behavioural;
```

Code 2.1: Generic counter component, modified from part 1

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity modulo_k_counter is
    generic(n : natural := 4;
            k : natural := 10);
    port(Clk : in  std_logic;
         Set : in  std_logic;
         Svl : in  std_logic_vector(n-1 downto 0);
         Rst : in  std_logic;
         Hop : out std_logic;
         Num : out std_logic_vector(n-1 downto 0));
end modulo_k_counter;

architecture behavioural of modulo_k_counter is

    signal Val : std_logic_vector(n-1 downto 0) := (others => '0');

begin
    process(Clk, Rst, Set)
    begin
        if rising_edge(Clk) then
            if Val >= k-1 then
                Val <= (others => '0');
                Hop <= '1';
            else
                Val <= Val + 1;
                Hop <= '0';
            end if;
        end if;
        if Set = '0' then
            if Svl >= k then
                Val <= (others => '0');
                Hop <= '1';
            else
                Val <= Svl;
                Hop <= '0';
            end if;
        end if ;
        if Rst = '0' then
            Val <= (others => '0');
            Hop <= '0';
        end if;
        Num <= Val;
    end process;
end behavioural;
```

Code 2.2: Binary Coded Decimal Display decoder component

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

entity display_BCD is
    port(Num    : in  std_logic_vector(6 downto 0);
         HEXlsd : out std_logic_vector(6 downto 0);
         HEXmsd : out std_logic_vector(6 downto 0));
end display_BCD;

architecture behavioural of display_BCD is

    component decoder7seg
        port(N   : in  std_logic_vector(3 downto 0);
             HEX : out std_logic_vector(6 downto 0));
    end component;

    signal lsd, msd : std_logic_vector(3 downto 0);
    signal Num_int, lsd_int, msb_int : integer;


begin
    process(Num)
    begin
        Num_int <= to_integer(unsigned(Num));

        if (Num_int >= 0 and Num_int < 10) then
            msd <= "0000";
        elsif (Num_int >= 10 and Num_int < 20) then
            msd <= "0001";
        elsif (Num_int >= 20 and Num_int < 30) then
            msd <= "0010";
        elsif (Num_int >= 30 and Num_int < 40) then
            msd <= "0011";
        elsif (Num_int >= 40 and Num_int < 50) then
            msd <= "0100";
        elsif (Num_int >= 50 and Num_int < 60) then
            msd <= "0101";
        elsif (Num_int >= 60 and Num_int < 70) then
            msd <= "0110";
        elsif (Num_int >= 70 and Num_int < 80) then
            msd <= "0111";
        elsif (Num_int >= 80 and Num_int < 90) then
            msd <= "1000";
        elsif (Num_int >= 90 and Num_int < 100) then
            msd <= "1001";
        end if;

        msb_int <= to_integer(unsigned(msd));
        lsd_int <= Num_int - (10 * msb_int);
        lsd <= std_logic_vector(to_unsigned(lsd_int, lsd'length));
    end process;

    HEX_lsd: decoder7seg port map
            (N   => lsd,
             HEX => HEXlsd);

    HEX_msd: decoder7seg port map
            (N   => msd,
```

```
60              HEX => HEXmsd);
61
62  end behavioural;
```

Code 2.3: 7 segment display decoder as used previously

```vhdl
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity decoder7seg is
5      port(N   : in  std_logic_vector(3 downto 0);
6           HEX : out std_logic_vector(6 downto 0));
7  end decoder7seg;
8
9  architecture behavioural of decoder7seg is
10
11 begin
12     process(N)
13     begin
14         case N is
15             when "0000" =>
16                 HEX <= "1000000";
17             when "0001" =>
18                 HEX <= "1111001";
19             when "0010" =>
20                 HEX <= "0100100";
21             when "0011" =>
22                 HEX <= "0110000";
23             when "0100" =>
24                 HEX <= "0011001";
25             when "0101" =>
26                 HEX <= "0010010";
27             when "0110" =>
28                 HEX <= "0000011";
29             when "0111" =>
30                 HEX <= "1111000";
31             when "1000" =>
32                 HEX <= "0000000";
33             when "1001" =>
34                 HEX <= "0011000";
35             when "1010" =>
36                 HEX <= "0001000";
37             when "1011" =>
38                 HEX <= "0000011";
39             when "1100" =>
40                 HEX <= "1000110";
41             when "1101" =>
42                 HEX <= "0100001";
43             when "1110" =>
44                 HEX <= "0000110";
45             when "1111" =>
46                 HEX <= "0001110";
47         end case;
48     end process;
49 end behavioural;
```

2.2. RTL

Unfortunately the RTL is very limited in this document as insane resolution is not feasible. There is a possibility to get a rough overview using the following figures, but a fully no-loss resizeable, pdf exports, of all the the RTL views is found on the GitHub inside the following path: `PLK_lab/Lab05/Extra/RTL_exports`.

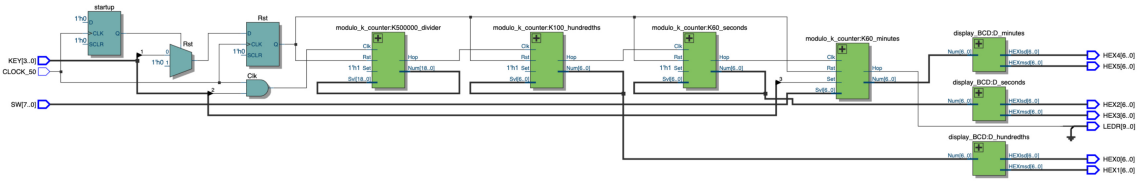


Figure 2.0: RTL of the TLE

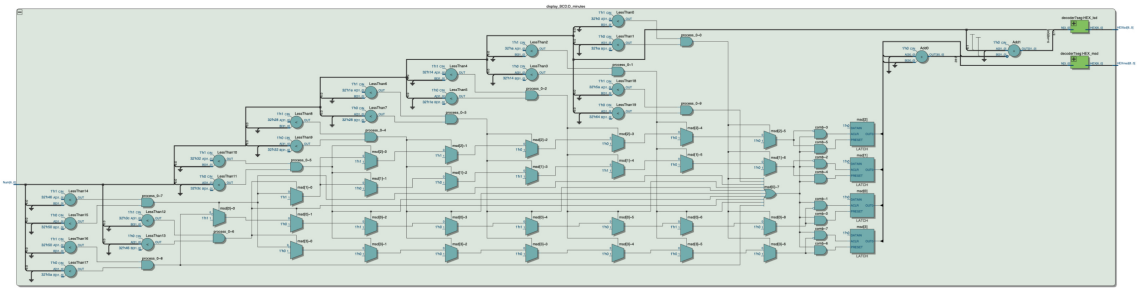


Figure 2.1: RTL of the BCD

RTL of the 7 segment display decoders is not included as this is from previous labs and not important for this lab. As always available on the GitHub.

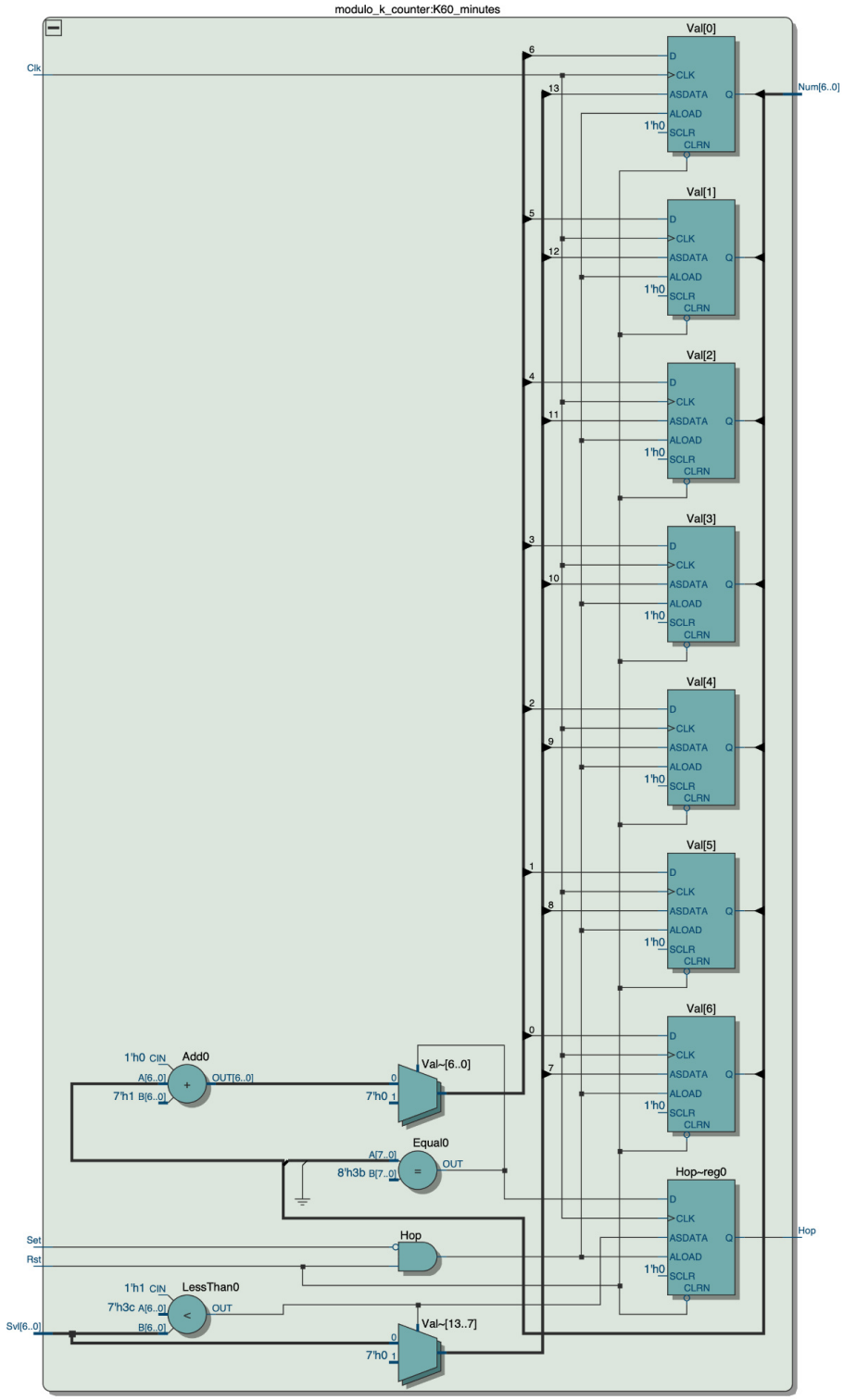


Figure 2.2: RTL of the generic counter

2.3. Results

It proved to be inefficient to take still pictures of a clock and therefore is a video uploaded instead to YouTube. You can watch it here.

Full url: https://youtu.be/4qAZzqxC_Sc?si=TP4QpGwuQ4AIIG2A

3. Part 3

The purpose of this task was to make a morse code generator for the first 8 letters of the alphabet, using the provided codes morse and the structure as described by the diagram.

3.0. Solving

Solving this was hard with my current knowledge on VHDL, therefore I set on a path to re-learn VHDL to really understand how everything works as it is different to the higher level programming. I had many issues with either trying to sync to different clock events, or trying to drive logic from multiple processes. I also found the hint codes very hard to follow and decided to make everything from scratch myself. The result was using a FSM (Finite State Machine) and making different states based on the current state and other conditions. A sketch of how these states should transfer to each other was made and after compiling, the state exported diagram shown if figure 3.0 ended up virtually identical.

The logic behind each module is that everything runs on the fast clock and that enable signals is sent during one clock cycle between all modules. There is enable signals for shifting the selected letter by one spot and at the same time decreasing the size. There is also a enable signal to Load new letter in from the selector. The reset is asynchronous but that is probably not visibly measurable as the clock is running at 50 MHz. The only thing from the diagram that is not implemented is the 2 bit counter as the functionality from implementing it separate instead of just counting and using the main clock seems to serve no additional purpose and was therefore omitted as readability was a higher priority.

3.1. Code

Code 3.0: TLE Connecting the components and I/O

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

entity L05P03 is
    port(CLOCK_50   : in  std_logic;
         SW         : in  std_logic_vector(2 downto 0);
         KEY        : in  std_logic_vector(3 downto 0);
         LEDR       : out std_logic_vector(9 downto 0));
end entity;


architecture behavioural of L05P03 is

    signal startup : std_logic := '1';
    signal Rst, LoadLetter, ShiftLetter : std_logic;
    signal LetterNumber : integer range 0 to 8;
    signal LetterData, Letter : std_logic_vector(3 downto 0);
    signal LetterSize : integer range 0 to 4;

begin

    LEDR(9 downto 1) <= (others => '0');

```

```
24      process(CLOCK_50) is
25      begin
26          if rising_edge(CLOCK_50) and startup = '1' then
27              startup <= '0';
28          end if;
29      end process;
30
31      Rst <= KEY(2) and not(startup);
32
33      Selector: entity work.LetterSelection(behavioural)
34          port map(Clk          => CLOCK_50,
35                   nRst         => Rst,
36                   Selection    => SW(2 downto 0),
37                   LetterNumber => LetterNumber,
38                   LetterData   => LetterData);
39
40      SizeRegister: entity work.LetterSizeRegister(behavioural)
41          port map(Clk          => CLOCK_50,
42                   nRst         => Rst,
43                   Enable       => ShiftLetter,
44                   Load         => LoadLetter,
45                   LetterNew    => LetterNumber,
46                   LetterSize   => LetterSize);
47
48      ShiftRegister: entity work.LetterShiftRegister(behavioural)
49          port map(Clk          => CLOCK_50,
50                   nRst         => Rst,
51                   Enable       => ShiftLetter,
52                   Load         => LoadLetter,
53                   LetterNew    => LetterData,
54                   LetterOut    => Letter);
55
56      LogicUnit: entity work.MorseLogic(behavioural)
57          port map(Clk          => CLOCK_50,
58                   nRst         => Rst,
59                   nEnable      => KEY(3),
60                   Letter       => Letter,
61                   LetterSize   => LetterSize,
62                   LoadLetter   => LoadLetter,
63                   ShiftLetter  => ShiftLetter,
64                   OutLED       => LEDR(0));
65
66  end architecture;
```

Code 3.1: Logic component controlling the state

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

entity MorseLogic is
    port(Clk         : in  std_logic;
         nRst        : in  std_logic;
         nEnable     : in  std_logic;
         Letter      : in  std_logic_vector(3 downto 0);
         LetterSize  : in  integer range 0 to 4;
         LoadLetter  : out std_logic;
         ShiftLetter : out std_logic;
         OutLED      : out std_logic);
end entity;


architecture behavioural of MorseLogic is

    type MorseState is (Sleep, Running, Dot, Dash, Between);
    signal State : MorseState;
    signal Count : integer range 0 to 75e6;

begin

    process(Clk, nRst) is
    begin
        if rising_edge(Clk) then
            Count <= Count + 1;
            case State is

                when Sleep =>
                OutLED <= '0';
                    if nEnable = '0' then
                        LoadLetter <= '1';
                        State <= Running;
                    end if;

                when Running =>
                    if not (LetterSize = 0) then
                        LoadLetter <= '0';
                        if Letter(3) = '0' then
                            State <= Dot;
                            ShiftLetter <= '1';
                            Count <= 0;
                        elsif Letter(3) = '1' then
                            State <= Dash;
                            ShiftLetter <= '1';
                            Count <= 0;
                        end if;
                    end if;

                when Dot =>
                    if Count = 25e6 - 1 then
                        Count <= 0;
                        if LetterSize = 0 then
                            State <= Sleep;
                        else
                            State <= Between;
                        end if;
```

```
                    else
                        OutLED <= '1';
                        ShiftLetter <= '0';
                    end if;

                when Dash =>
                    if Count = 75e6 - 1 then
                        Count <= 0;
                        if LetterSize = 0 then
                            State <= Sleep;
                        else
                            State <= Between;
                        end if;
                    else
                        OutLED <= '1';
                        ShiftLetter <= '0';
                    end if;

                when Between =>
                    OutLED <= '0';
                    if Count = 25e6 - 1 then
                        Count <= 0;
                        State <= Running;
                    end if;

            end case;
        end if;
        if nRst = '0' then
            State <= Sleep;
            Count <= 0;
            LoadLetter <= '0';
            ShiftLetter <= '0';
            OutLED <= '0';
        end if;
    end process;

end architecture;
```

Code 3.2: Letter selection component

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

entity LetterSelection is
    port(Clk          : in  std_logic;
         nRst         : in  std_logic;
         Selection    : in  std_logic_vector(2 downto 0);
         LetterNumber : out integer range 0 to 8;
         LetterData   : out std_logic_vector(3 downto 0));
end entity;


architecture behavioural of LetterSelection is

    signal LetterNr : integer range 0 to 8 := 0;

begin

    process(Clk, nRst)
    begin
        if rising_edge(Clk) then
            LetterNr <= to_integer(unsigned(Selection)) + 1;
            case LetterNr is
                when 1 =>      -- A ._
                    LetterData <= "01--";
                when 2 =>      -- B _...
                    LetterData <= "1000";
                when 3 =>      -- C _._.
                    LetterData <= "1010";
                when 4 =>      -- D _..
                    LetterData <= "100-";
                when 5 =>      -- E .
                    LetterData <= "0---";
                when 6 =>      -- F .._.
                    LetterData <= "0010";
                when 7 =>      -- G __.
                    LetterData <= "110-";
                when 8 =>      -- H ....
                    LetterData <= "0000";
                when others =>
                    LetterData <= (others => '-');
            end case;
        end if;
        LetterNumber <= LetterNr;
        if nRst = '0' then
            LetterNumber <= 0;
            LetterData   <= "----";
        end if;
    end process;

end architecture;
```

Code 3.3: Letter size component

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

entity LetterSizeRegister is
    port(Clk        : in  std_logic;
         nRst       : in  std_logic;
         Enable     : in  std_logic;
         Load       : in  std_logic;
         LetterNew  : in  integer range 0 to 8;
         LetterSize : out integer range 0 to 4);
end entity;


architecture behavioural of LetterSizeRegister is

    signal Size : integer range 0 to 4 := 0;
begin

    LetterSize <= Size;

    process(Clk, nRst) is
    begin
        if rising_edge(Clk) then
            if Enable = '1' then
                Size <= Size - 1;
            end if;
            if Load = '1' then
                case  LetterNew is
                    when 1 =>       -- A ._
                        Size <= 2;
                    when 2 =>       -- B _...
                        Size <= 4;
                    when 3 =>       -- C _._.
                        Size <= 4;
                    when 4 =>       -- D _..
                        Size <= 3;
                    when 5 =>       -- E .
                        Size <= 1;
                    when 6 =>       -- F .._.
                        Size <= 4;
                    when 7 =>       -- G __.
                        Size <= 3;
                    when 8 =>       -- H ....
                        Size <= 4;
                    when others =>
                        Size <= 0;
                end case;
            end if;
        end if;
        if nRst = '0' then
            Size <= 0;
        end if;
    end process;

end architecture;
```

Code 3.4: Letter shift component

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

entity LetterShiftRegister is
    port(Clk       : in  std_logic;
         nRst      : in  std_logic;
         Enable    : in  std_logic;
         Load      : in  std_logic;
         LetterNew : in  std_logic_vector(3 downto 0);
         LetterOut : out std_logic_vector(3 downto 0));
end entity;


architecture behavioural of LetterShiftRegister is

    signal Letter : std_logic_vector(3 downto 0) := (others => '0');

begin

    LetterOut <= Letter;

    process(Clk, nRst) is
    begin
        if rising_edge(Clk) then
            if Load = '1' then
                Letter <= LetterNew;
            else
                if Enable = '1' then
                    Letter <= Letter (2 downto 0) & '-';
                end if;
            end if;
        end if;
        if nRst = '0' then
            Letter <= (others => '0');
        end if;
    end process;

end architecture;
```

3.2. State and RTL

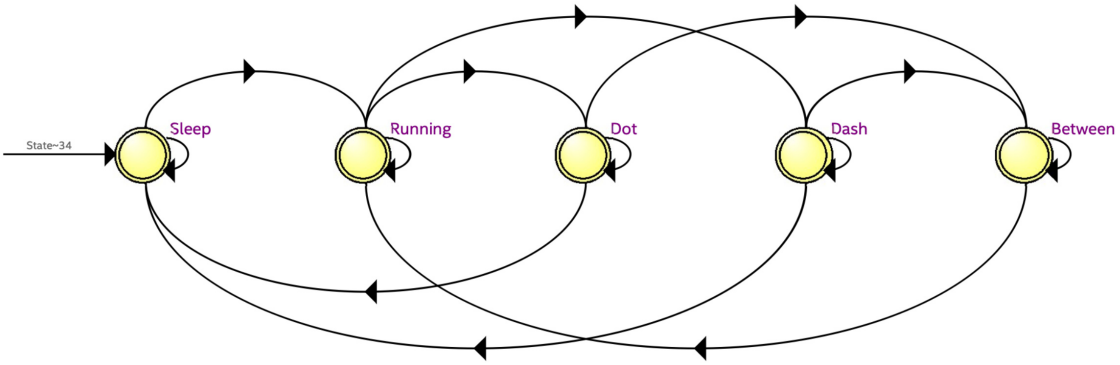


Figure 3.0: FSM diagram from Quartus

Unfortunately the RTL is very limited in this document as insane resolution is not feasible. There is a possibility to get a rough overview using the following figures, but a fully no-loss resizeable, pdf exports, of all the the RTL views is found on the GitHub inside the following path: `PLK_lab/Lab05/Extra/RTL_exports`.

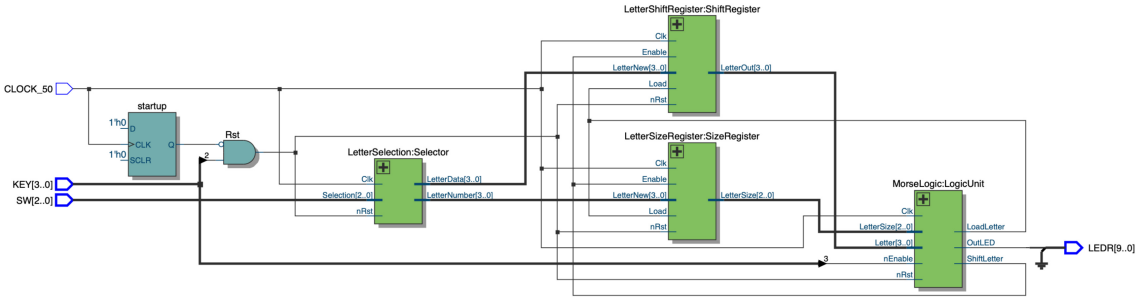


Figure 3.1: RTL of the TLE

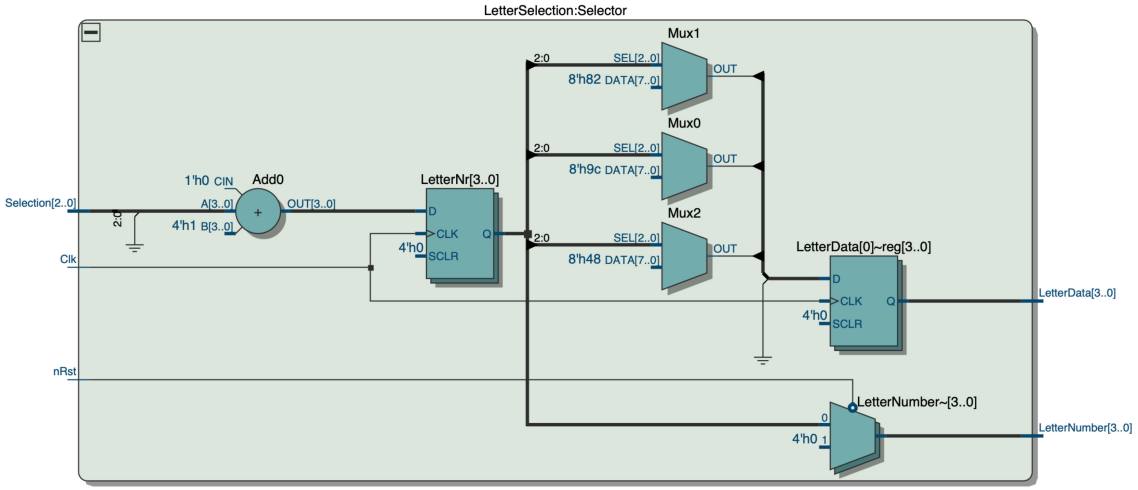


Figure 3.2: RTL of the letter selector

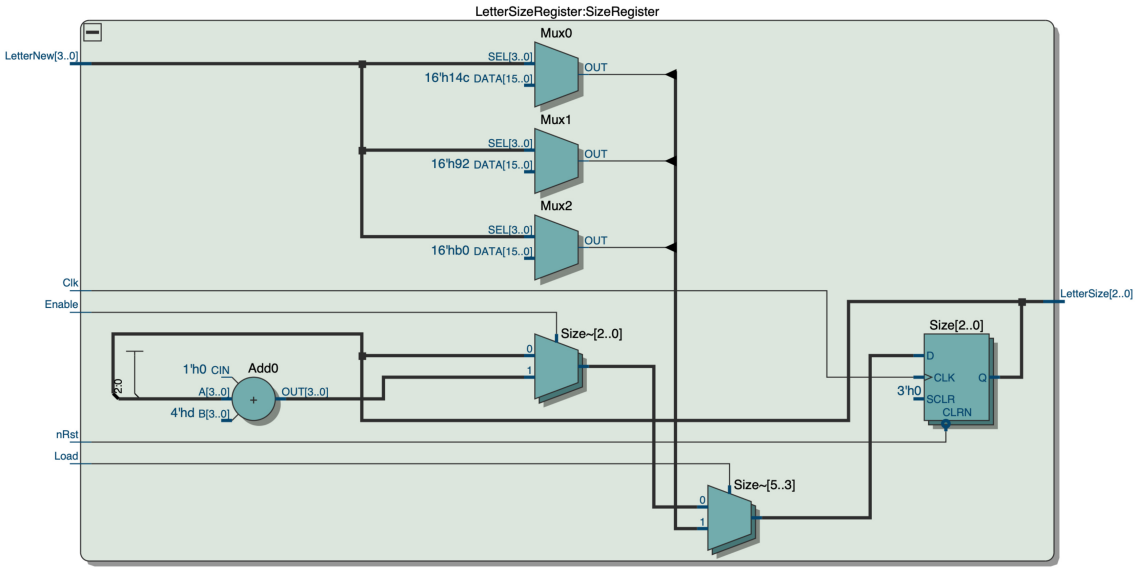


Figure 3.3: RTL of the letter size register

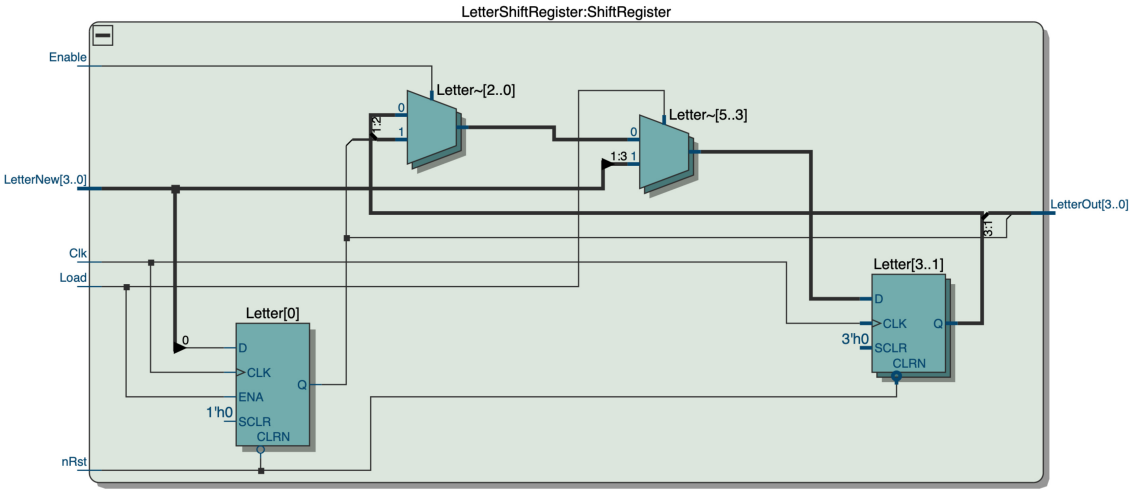


Figure 3.4: RTL of the letter shift register

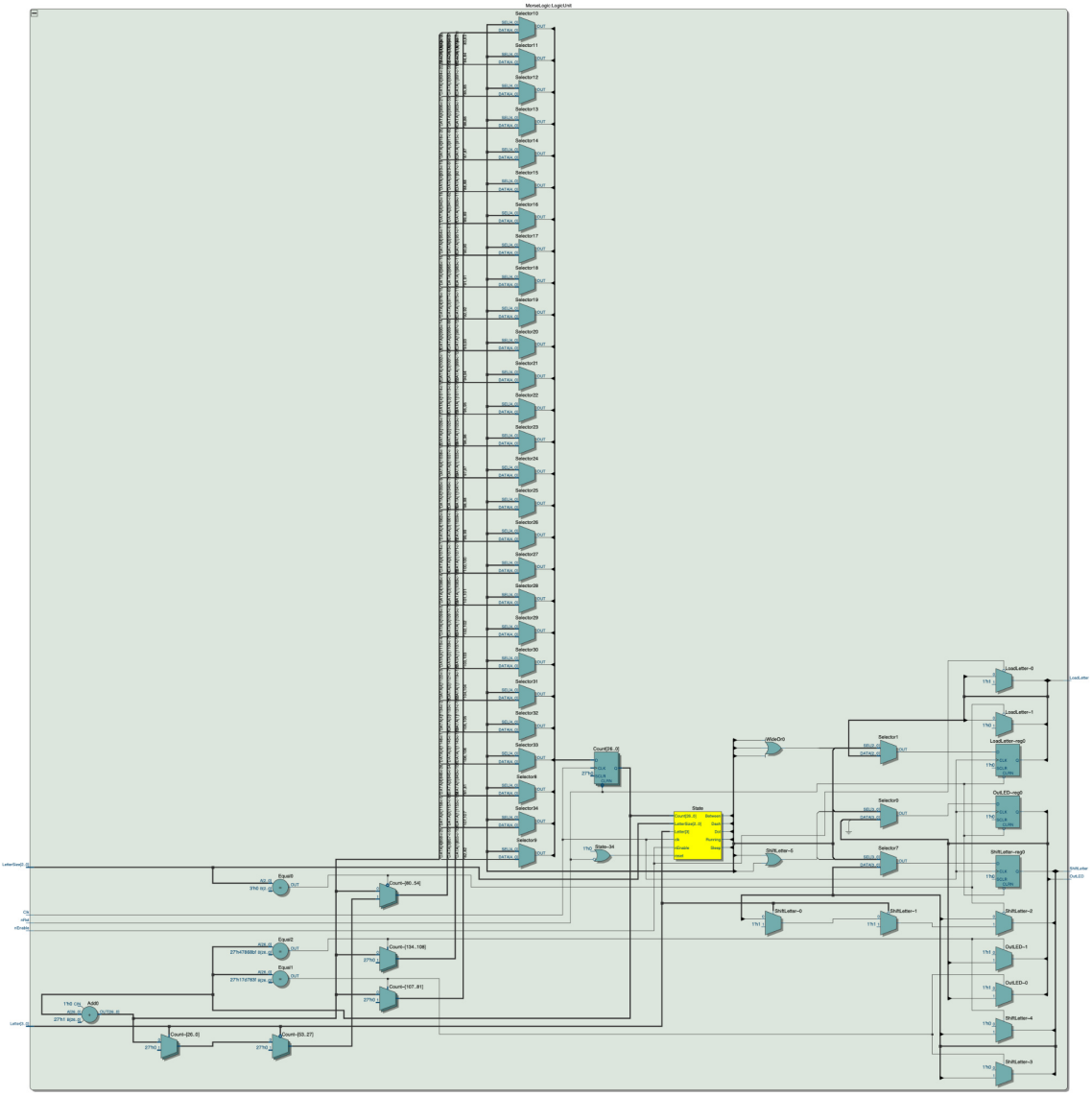


Figure 3.5: RTL of the Logic

3.3. Results

It proved to be inefficient to take still pictures of a timed sequence and therefore is a video uploaded instead to YouTube. You can watch it here.

Full url: https://youtu.be/hirlx2AdUFk?si=sx53VB_D5rsin8en