Universitetet i Sørøst-Norge

PB1180 - Programmerbare logiske kretser
Final Project

Kandidatnr:
8022

September 6, 2025

1- Abstract

This is the report for the Final Project in PB-1180 - Programmerbare logiske kretser. There were two possible mutually exclusive projects to choose from and I chose to do Course Project B. There were given no criteria for report formatting, therefore it does not follow any given template and is customized to my own liking.

Programs used to complete this project:

- Visual Studio Code - as a general code writing editor
- VHDL by VHDLwhiz plug-in for VScode - for syntax highlighting
- GitHub - private repository to store code and figures for synchronizing
- ModelSim - to simulate using test benches
- Quartus - for implementation on FPGA and synthesizing RTL views
- Excel - to write tables used in the report
- Excel2LaTeX Excel add-inn - to generate LaTeXcode from Excel table
- SmartDraw - for making state transition diagram
- Affinity Photo 2 - to edit/crop photos for the report
- DaVinci Resolve 20 - for editing the included video
- Overleaf - to write the report using LaTeX

1.1 Table of Contents

1.2 List of Figures

1.3 List of Tables

2- Approach to solution

2.1 Introduction

My approach for a Project like this is to start by systematically segment it to smaller tasks. I started by writing a top level overview of how I want this report to be structured and then I started dissecting the problem. After the problem was understood I started with figuring out how I would like to solve the problem. Then after a high level overview of the solution is made I created a structure where I can write the code to be readable and easily understandable. Then it was onto actually writing the code, here Lab exercise 5 and 6 taught me a lot about the principals used in this project and a lot of the code is inspired from my completion of those. After the code was completed I first tested it on the FPGA, where it worked, and then later developed test benches to simulate. This is mostly because I was very certain that the logic was good and only thing that could be wrong would be some minor issues that could easily be fixed. In the end I started making this report, editing the photos and videos, and including them. And finally the report is written.

2.2 The Problem

To solve the problem presented in the task, the problem must first be understood at the basic level. When the problem is acknowledged a solution can be formed. The stated problem, in simplified form, is to make a circuit that can serially display a certain combination of blinks encoded to be interpreted as some characters. With some additional details, such as a reset button with light and a 7 segment display of the character.

2.3 Definitions

To avoid confusion these definitions and acronyms will be used:

- Letters - is all the characters the code will display, P, B, 1, 8, -, and so on
- Symbols - is the type of signal that makes up a character, dot, dash and bar
- FSM - Finite State Machine
- TLE - Top level entity
- FPGA - Field-Programmable Gate Array
- HDL - Hardware Descriptive Language
- VHDL - Very High Speed Integrated Circuit HDL

2.4 Solving

The first thing to realize when solving is that this is very similar to morse code. The exception is that morse code is only two symbols, dot and dash, and the gamma code has the extra vertical bar symbol. In addition one dot is 4 short pulses instead of a singular short pulse like in morse code. To solve this task it is good to start with the encoding. At any point during transmission the LED can only be in two states, on or off. This means it is possible to encode each character as a string of ones and zeros with each bit representing the state for a short time. This will make a code with little logic and a large lookup table. I felt this would have been a bad design and chose not to do it this way. Instead the encoding was made for each possible symbol that describes a letter. These three values they can be represented with two bits. 01 for dot, 10 for dash and 11 for bar. This leaves 00 free and used this encoding for the pause between each symbol. Using this encoding set, a FSM was designed as described in 3.1. A lot of the code I made was inspired by the code I made for Lab 05. Some parts of the code was copy pasted but most parts were redesigned for better readability and to fit the bigger letter encoding.

2.5 Structure

For better readability it was coded into 4 separate components:

1. Lookup table - for letter selection
2. Shift Register - for storing current Letter and shifting it along
3. Decoder - for the 7 segment display
4. Logic control - for the FSM transitions and output

Then the TLE connects all together and does not contain any logic except for a startup reset and the Reset button LED.

3- The FSM

3.1 State logic design

Using the encoding described in 2.4 a FSM diagram was started, where Dot, Dash, Bar and Between are 4 states. in addition the system needed a default state it can reset to and not show anything, this is the Idle state. The Dot, Dash, Bar and Between state could branch out from Idle directly, but that makes the transition logic cumbersome as then all states need to be able to go directly to all other states. Instead a segway state for controlling the timers and routing the states was made, the Running state. This state will only be on for a single clock cycle but does all of the routing logic. After this initial plan a modification to the Dot state was made. It is split into two states, a Dot On and Dot Off, where it alternates between to blink on 4 times before returning to Running. With the states set up like this the output was very simple, only depending on what state it was in, nothing else. In table 3.1 The state transitions is described in detail. Based on the current state (Columns) the next state will be the listed one based on if the input value (Rows) is true, a ”-” signifies no state change. Obviously this table can’t contain all possible values and still be readable so all non listed values will result in no state change. If this table seems the wrong way around it is because it is transposed to be able to fit the document. Table 3.2 describes the outputs, and because the choice of states it is very simple and only relies on the current state. Note that Display or Load is redundant as they are just opposite, both are included for readability, not out of necessity. There are multiple ways to encode the state variable and the Task specifies to use ”my preferred encoding method” and as I am not making any structural logic for the states it does not affect anything whatever the encoding is. When compiling the code in Quartus the encoding used is one-hot as shown in figure 3.1.

	Name	Between	Bar	Dash	DotOff	DotOn	Running	Idle
1	Idle	0	0	0	0	0	0	0
2	Running	0	0	0	0	0	1	1
3	DotOn	0	0	0	0	1	0	1
4	DotOff	0	0	0	1	0	0	1
5	Dash	0	0	1	0	0	0	1
6	Bar	0	1	0	0	0	0	1
7	Between	1	0	0	0	0	0	1

Figure 3.1: Encoding table made by Quartus

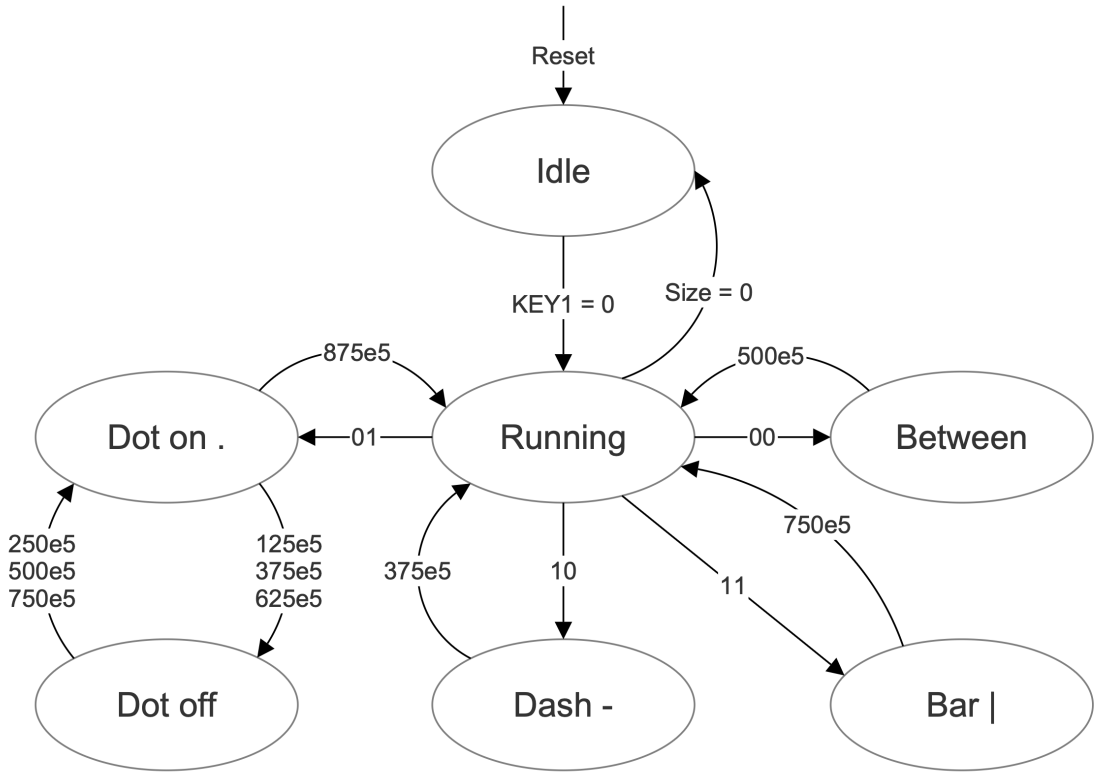

Figure 3.2: State transition diagram

In this diagram the numbers with the e5 suffix is representing the counter value and the two digit numbers is representing the value of the two most significant bits of the letter data. Dot on and Dot off have 3 different values where the state jump between them.

3.2 State tables

Table 3.1: State transitions

State transition			Current state						
			Idle	Running	DotOn	DotOff	Dash	Bar	Between
Input	Type	Value	S0	S1	S2	S3	S4	S5	S6
Keys	Reset	KEY0 = 0	S0	S0	S0	S0	S0	S0	S0
	Load	KEY1 = 0	S1	-	-	-	-	-	-
Letter	Size	Size = 0	-	S0	-	-	-	-	-
	Between	00	-	S6	-	-	-	-	-
	Dot	01	-	S2	-	-	-	-	-
	Dash	10	-	S4	-	-	-	-	-
	Bar	11	-	S5	-	-	-	-	-
Counter	0.25s	125e5	-	-	S3	-	-	-	-
	0.5s	25e6	-	-	-	S2	-	-	-
	0.75s	375e5	-	-	S3	-	S1	-	-
	1s	5e7	-	-	-	S2	-	-	S1
	1.25s	625e5	-	-	S3	-	-	-	-
	1.5s	75e6	-	-	-	S2	-	S1	-
	1.75s	875e5	-	-	S1	-	-	-	-

Table 3.2: State outputs

State output		Outputs			
Current state		LED	Display	Shift	Load
Idle	S0	0	0	0	1
Running	S1	0	1	0	0
Dot	S2	1	1	1	0
Dot off	S3	0	1	1	0
Dash	S4	1	1	1	0
Bar	S5	1	1	1	0
Between	S6	0	1	1	0

3.3 Moore and Mealy

Comparing this design with the fundamental Moore and Mealy machines it is very clear that this is a Moore machine. The main reasoning is that the outputs only depend on the current state. The reason for doing this is very simple, Moore machines are easier to make than Mealy machines. As this design is not very large anyway, in the context of this task, it seems unreasonable to focus on the optimization possible with a Mealy machine. The number of states and logic is not large enough to see any real improvements. It is worth noting that the 1 clock cycle output delay a Moore machine has can be seen in the simulation result in figure 5.11. This delay is because on the first clock cycle the State changes, then on the next clock cycle the output changes.

4- VHDL code

4.1 FSM code

Code 4.1: TLE that ties the system together

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

entity FinalProject is
    port(CLOCK_50 : in  std_logic;
         SW       : in  std_logic_vector(3 downto 0);
         KEY      : in  std_logic_vector(1 downto 0);
         LEDR     : out std_logic_vector(9 downto 0);
         HEX0     : out std_logic_vector(6 downto 0));
end entity;


architecture behavioural of FinalProject is

    signal startup                      : std_logic := '1';
    signal Reset, Load, Shift, Display  : std_logic;
    signal Symbol, LetterSize, SizeLoad : integer range 0 to 15;
    signal Letter, LetterLoad           : std_logic_vector(13 downto 0);
    -- All signals except startup is to connect modules
begin

    LEDR(9 downto 4) <= (others => '0');
    LEDR(1 downto 0) <= (others => '0');
    LEDR(2)          <= not KEY(0);

    process(CLOCK_50) is -- Self reseting on startup
    begin
        if rising_edge(CLOCK_50) and startup = '1' then
            startup <= '0';
        end if;
    end process;

    Reset <= KEY(0) and not(startup);

    -- Transforms switch selection to data
    Selector: entity work.LetterSelection(behavioural)
        port map(Clk          => CLOCK_50,
                 nRst         => Reset,
                 Selection    => SW(3 downto 0),
                 Symbol       => Symbol,
                 LetterData   => LetterLoad,
                 LetterSize   => SizeLoad);

    -- Saves selection and shifts data
    ShiftRegister: entity work.LetterRegister(behavioural)
        port map(nRst         => Reset,
                 Shift        => Shift,
                 Load         => Load,
                 LetterLoad   => LetterLoad,
                 SizeLoad     => SizeLoad,
                 LetterOut    => Letter,
                 SizeOut      => LetterSize);

```

```
    -- Decoder for the 7 segment display output
    DisplayDecoder: entity work.Decoder7Segment(behavioural)
        port map(nRst         => Reset,
                 Enable       => Display,
                 Load         => Load,
                 Symbol       => Symbol,
                 HEX          => HEX0);

    -- Controlls the logic with states and controlls register and output
    LogicUnit: entity work.GammaLogic(behavioural)
        port map(Clk          => CLOCK_50,
                 nRst         => Reset,
                 nEnable      => KEY(1),
                 Shift        => Shift,
                 Load         => Load,
                 Letter       => Letter,
                 LetterSize   => LetterSize,
                 LEDoutput    => LEDR(3),
                 Display      => Display);

end architecture;
```

Originally LED0 was the chosen output LED, but after trying to film and realizing LED0 is under the label I chose LED3 as it is directly under the used 7 segment display. For the Light on reset I simply not gated the reset button and used LED2 next to the output LED. LEDs 9 to 4 and 1 to 0 is not used and therefore constant low, if this was not set the LED would appear half on in an undefined state and that looks bad and could be confusing.

Code 4.2: Letter selector is a lookup table

VHDL

```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(3 downto 0);
         Symbol     : out integer range 0 to 15;
         LetterData : out std_logic_vector(13 downto 0);
         LetterSize : out integer range 0 to 15);
end entity;


architecture behavioural of LetterSelection is

    signal SymbolNr : integer range 0 to 15 := 0;

begin

    Symbol <= SymbolNr; -- Avoids reading the output in the case when

    process(Clk, nRst)
    begin
        if rising_edge(Clk) then
            SymbolNr <= to_integer(unsigned(Selection));
            case SymbolNr is    -- Data and size of letters stored here
                when 1 =>       -- P as - - .
```

```
                    LetterData <= "1000100001----"; -- 00 between symbols
                    LetterSize <= 5;                -- 01 for dots
                when 2 =>       -- B as | . .       -- 10 for dash
                    LetterData <= "1100010001----"; -- 11 for vertical bar
                    LetterSize <= 5;                -- -- for don't care
                when 3 =>       -- 1 as | . -
                    LetterData <= "1100010010----";
                    LetterSize <= 5;
                when 4 =>       -- 8 as . | .
                    LetterData <= "0100110001----";
                    LetterSize <= 5;
                when 5 =>       -- 0 as - . -
                    LetterData <= "1000010010----";
                    LetterSize <= 5;
                when 6 =>       -- - as -
                    LetterData <= "10------------";
                    LetterSize <= 1;
                when 7 =>       -- F as | - .
                    LetterData <= "1100100001----";
                    LetterSize <= 5;
                when 8 =>       -- G as | . . |
                    LetterData <= "11000100010011";
                    LetterSize <= 7;
                when 9 =>       -- A as . - .
                    LetterData <= "0100100001----";
                    LetterSize <= 5;
            when others =>      -- Whenever the input is something else
                    LetterData <= (others => '-');
                    LetterSize <= 0;
            end case;
        end if;

        if nRst = '0' then -- Asynchronous reset for outputs
            SymbolNr <= 0;
            LetterData <= (others => '-');
            LetterSize <= 0;
        end if;
    end process;
end architecture;
```

The Letter data is simply a string of the symbols that make up each letter. Then between each symbol 00 is inserted to have the wait in-between each symbol being displayed. Technically this could have been omitted and only the symbol part being part of the string, but this would make the state transitions unnecessary more complex. the string is always 14 bits, which would be a size of maximum 7 symbols and betweens. For the letters not using 4 symbols the remaing string is filled with ”-” which is the don’t care signal.

Code 4.3: Custom loadable shift register

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

entity LetterRegister is
    port(nRst       : in  std_logic;
         Shift      : in  std_logic;
         Load       : in  std_logic;
         LetterLoad : in  std_logic_vector(13 downto 0);
         SizeLoad   : in  integer range 0 to 15;
         LetterOut  : out std_logic_vector(13 downto 0);
         SizeOut    : out integer range 0 to 15);
end entity;


architecture behavioural of LetterRegister is

    signal Letter : std_logic_vector(13 downto 0) := (others => '0');
    signal Size   : integer range 0 to 15 := 0;

begin

    LetterOut <= Letter; -- Avoids reading the output while shifting
    SizeOut   <= Size;   -- Avoids reading the output while decreasing

    process(Shift, Load, LetterLoad, nRst) is
    begin -- LetterLoad in sensitivity list to update value while idle
        if rising_edge(Shift) then -- Shift acts as a clk signal
            Letter <= Letter (11 downto 0) & "--";
            Size   <= Size - 1;
        end if;
        if Load = '1' then -- Always updates on input change
            Letter <= LetterLoad;
            Size   <= SizeLoad;
        end if;
        if nRst = '0' then -- Asynchronous reset for outputs
            Letter <= (others => '0');
            Size   <= 0;
        end if;
    end process;

end architecture;
```

The shift register shifts two bits each time shift is enabled, causing the two most significant bits to be the next symbol or between for the logic to read. It shifts in two don't care signals as it doesn't matter what it resolves to, it is never read. it also subtracts the size of the letter by one at the same time. Both Load and LetterLoad is needed in the sensitivity list. This is because if the switches don't change after going into idle, Load needs to trigger the register to update the to current selected letter, and if the current selection changes while in idle LetterLoad will change and therefore update the process and update the output letter.

Code 4.4: Decoder with load for 7-segment display output

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

entity Decoder7Segment is
    port(nRst    : in  std_logic;
         Enable : in  std_logic;
         Load    : in  std_logic;
         Symbol : in  integer range 0 to 15;
         HEX     : out std_logic_vector(6 downto 0));
end entity;


architecture behavioural of Decoder7Segment is

    signal SavedSymbol : integer range 0 to 15;

begin

    process(Load, Symbol, nRst)
    begin -- Symbol in sensitivity list to continually update on input
        if Load = '1' then
            SavedSymbol <= Symbol;
        end if;

        if nRst = '0' then -- Asynchronous reset for saved symbol
            SavedSymbol <= 0;
        end if;
    end process;

    process(Enable, SavedSymbol, nRst)
    begin -- SavedSymbol in sensitivity list if it updates after Enable
        if Enable = '1' then
            case SavedSymbol is
                when 1 => -- P
                    HEX <= "0001100";
                when 2 => -- B
                    HEX <= "0000011";
                when 3 => -- 1
                    HEX <= "1111001";
                when 4 => -- 8
                    HEX <= "0000000";
                when 5 => -- 0
                    HEX <= "1000000";
                when 6 => -- -
                    HEX <= "0111111";
                when 7 => -- F
                    HEX <= "0001110";
                when 8 => -- G
                    HEX <= "1000010";
                when 9 => -- A
                    HEX <= "0001000";
                when others => -- Off for undefined symbols
                    HEX <= (others => '1');
            end case;
        else -- Off if display is not enabled
            HEX <= (others => '1');
        end if;

```

```
60          if nRst = '0' then -- Asynchronous reset for outputs
61              HEX <= (others => '1'); -- Display off
62          end if;
63      end process;
64
65  end architecture;
```

The decoder is simply a lookup table that outputs the loaded symbol to the display whenever enabled. The stored values was found using the 7-segment display pinout and making table 4.3. For the same reason as with the shift register both Load and Symbol needs to be in the sensitivity list for it to load correctly on idle. Both SavedSymbol and Enable is in the other sensitivity list as there could be a single clock cycle delay between those updating that could cause it to not be displayed correctly. This is also why the display is set to off while SavedSymbol is something else than the configured values, as technically the display is enabled in one cycle while state changes from idle to running and back when input is not in the defined area. Since this happens very fast it could appear as a dimmer display with some undefined value, therefore it is set to always be off when values are out of bounds.

Table 4.3: Output string from decoder

Symbol	NR	HEX							String
		6	5	4	3	2	1	0	"6543210"
P	1	0	0	0	1	1	0	0	"0001100"
B	2	0	0	0	0	0	1	1	"0000011"
1	3	1	1	1	1	0	0	1	"1111001"
8	4	0	0	0	0	0	0	0	"0000000"
0	5	1	0	0	0	0	0	0	"1000000"
-	6	0	1	1	1	1	1	1	"0111111"
F	7	0	0	0	1	1	1	0	"0001110"
G	8	1	0	0	0	0	1	0	"1000010"
A	9	0	0	0	1	0	0	0	"0001000"

Code 4.5: Logic control deciding FSM transitions and output

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

entity GammaLogic is
    port(Clk        : in  std_logic;
         nRst       : in  std_logic;
         nEnable    : in  std_logic;
         Shift      : out std_logic;
         Load       : out std_logic;
         Letter     : in  std_logic_vector(13 downto 0);
         LetterSize : in  integer range 0 to 15;
         LEDoutput  : out std_logic;
         Display    : out std_logic);
end entity;


architecture behavioural of GammaLogic is

    type GammaState is (       -- Custom type with states of choice
        Idle, Running, DotOn, DotOff, Dash, Bar, Between);
    signal State : GammaState; -- State variable of the custom type
    signal TimeCounter : integer range 0 to 1000e5; -- Within 27 bits

begin

    process(Clk, nRst) is
    begin
        if rising_edge(Clk) then
            TimeCounter <= TimeCounter + 1; -- Times based on 50Mhz clock
            case State is    -- State change case for controlling states

                when Idle => -- Waiting for button press
                    if nEnable = '0' then
                        State <= Running;
                    end if;

                when Running => -- Routing signal on/off
                    if LetterSize = 0 then
                        State <= Idle;    -- Return to Idle when done
                    else                  -- Routes based on letter data
                        TimeCounter <= 0; -- Restarts counter each symbol
                        if    Letter(13 downto 12) = "00" then
                            State <= Between;
                        elsif Letter(13 downto 12) = "01" then
                            State <= DotOn;
                        elsif Letter(13 downto 12) = "10" then
                            State <= Dash;
                        elsif Letter(13 downto 12) = "11" then
                            State <= Bar;
                        else -- As a safety measure return to Idle
                            State <= Idle;
                        end if;
                    end if;

                when Between => -- Waiting 1s between two symbols
                    if TimeCounter >= 500e5 then
                        State <= Running;
                    end if;
```

```
60
61                  when DotOn => -- Displaying dot . for 0.25s 4 times
62                      if    TimeCounter >= 875e5 then
63                          State <= Running;
64                      elsif (TimeCounter = 125e5  or
65                             TimeCounter = 375e5  or
66                             TimeCounter = 625e5) then
67                          State <= DotOff;
68                      end if;
69
70                  when DotOff => -- Turning dot off for 0.25s 3 times
71                      if (TimeCounter =  250e5  or
72                          TimeCounter =  500e5  or
73                          TimeCounter >= 750e5) then
74                          State <= DotOn;
75                      end if;
76
77                  when Dash => -- Displaying dash - for 0.75s 1 time
78                      if TimeCounter >= 375e5 then
79                          State <= Running;
80                      end if;
81
82                  when Bar => -- Displaying bar | for 1.5s 1 time
83                      if TimeCounter >= 750e5 then
84                          State <= Running;
85                      end if;
86              end case;
87          end if;
88
89          if nRst = '0' then -- Asynchronous reset for state and counter
90              State <= Idle;
91              TimeCounter <= 0;
92          end if;
93      end process;
94
95      process(State, nRst) is
96      begin
97          case State is -- Output case for controlling outputs
98
99              when Idle =>
100                 LEDoutput <= '0';
101                 Display   <= '0';
102                 Shift     <= '0';
103                 Load      <= '1';
104
105             when Running =>
106                 LEDoutput <= '0';
107                 Display   <= '1';
108                 Shift     <= '0';
109                 Load      <= '0';
110
111             when Between =>
112                 LEDoutput <= '0';
113                 Display   <= '1';
114                 Shift     <= '1';
115                 Load      <= '0';
116
117             when DotOn =>
118                 LEDoutput <= '1';
119                 Display   <= '1';
120                 Shift     <= '1';
```

```
121                 Load      <= '0';
122 
123             when DotOff =>
124                 LEDoutput <= '0';
125                 Display   <= '1';
126                 Shift     <= '1';
127                 Load      <= '0';
128 
129             when Dash =>
130                 LEDoutput <= '1';
131                 Display   <= '1';
132                 Shift     <= '1';
133                 Load      <= '0';
134 
135             when Bar =>
136                 LEDoutput <= '1';
137                 Display   <= '1';
138                 Shift     <= '1';
139                 Load      <= '0';
140         end case;
141 
142         if nRst = '0' then -- Asynchronous reset for outputs
143             LEDoutput <= '0';
144             Display   <= '0';
145             Shift     <= '0';
146             Load      <= '0';
147         end if;
148     end process;
149 
150 end architecture;
```

Here is the logic from the State transition diagram in figure 3.2. The counter used for timing is running on the internal 50Mhz clock, this makes 125e5 counts equivalent to waiting 0.25 seconds. The timing for the on period of the dot is specified to be 0.25s each blink. As there were not specified any time interval for the off period in between I made this the same at 0.25s. The time on for dash was specified to be at 0.75s and for the bar 1.5s was specified. However no timing for the off interval between each symbol was mentioned I then choose to make this time an even 1s long as that seems in line with the other symbol lengths. Note: it is also very clear to see in the output process that it is a Moore machine, it is only a simple case when for each state.

4.2 RTL view

Using Quartus to synthesize the RTL view of the TLE and of each sub module is a useful tool for inspecting the inner workings. Unfortunately Quartus does not let you move around objects and usually makes either very wide or very long diagrams. This does not make it feasible to include in the report and keep it readable. The following pictures will therefore most likely be very hard to read or blurry as of resolution restrictions. Because of this all of these RTL views are uploaded together with this report in a pdf file that includes infinitely scalable drawings where zooming in far enough to see every detail is possible.

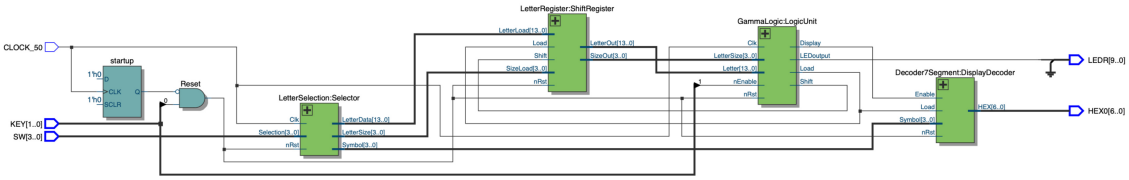

Figure 4.3: RTL of the TLE

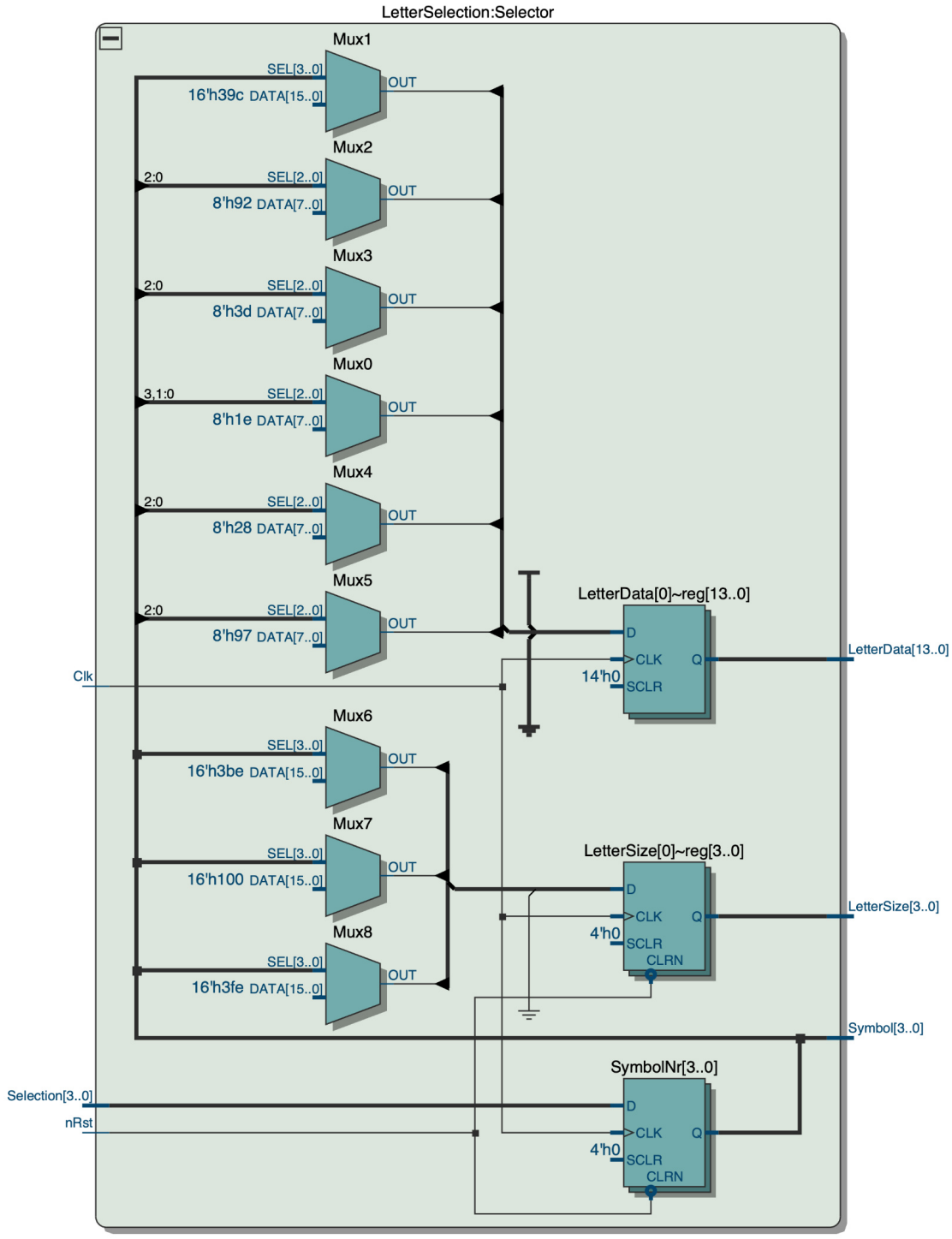

Figure 4.4: RTL of the letter selector component

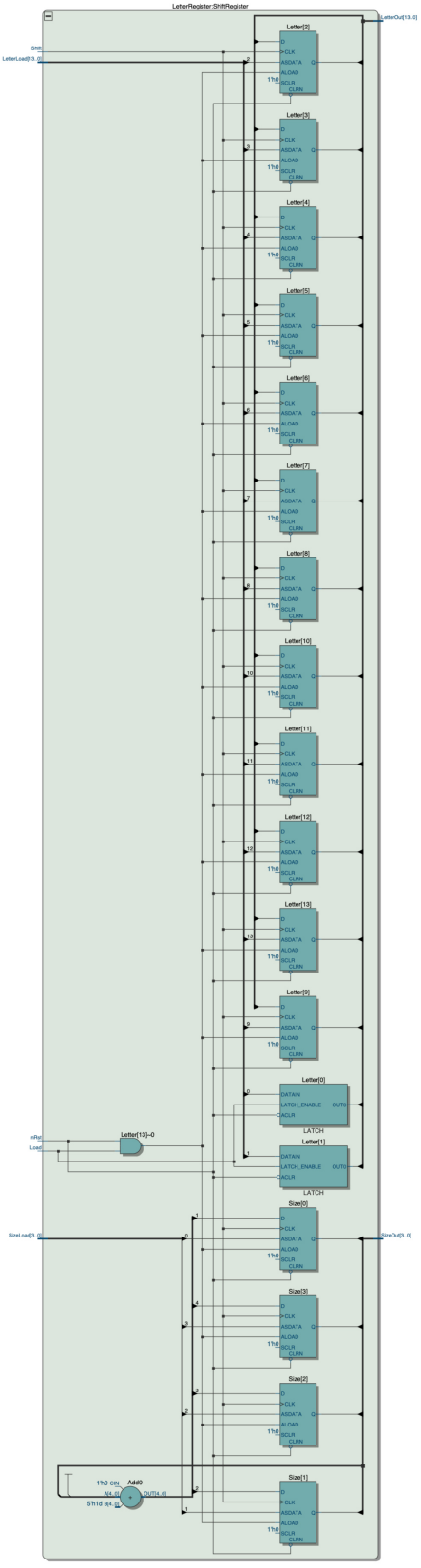

Figure 4.5: RTL of the letter shift register component

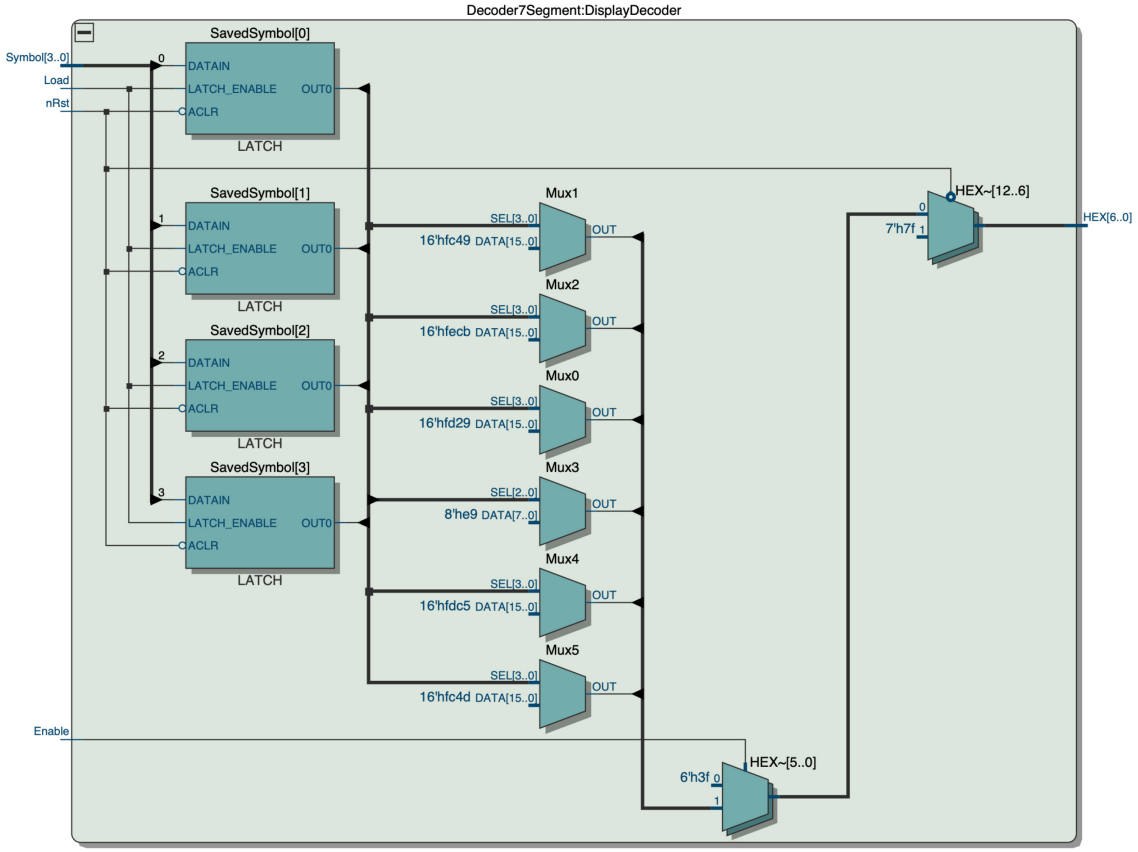

Figure 4.6: RTL of the 7 segment decoder component

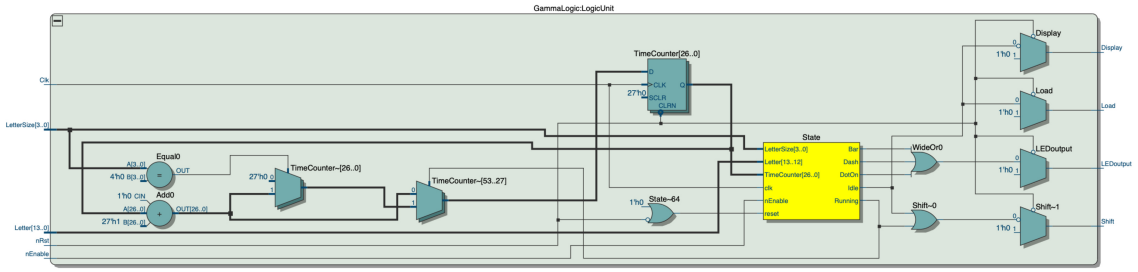

Figure 4.7: RTL of the logic component

Quartus correctly recognizes that this is a FSM as seen in figure 4.7 but it will not make a state transition diagram even though it is not that complex. Instead refer to the diagram in figure 3.2.

4.3 Test bench code

Code 4.6: Testbench for the Letter selector

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

entity LetterSelectionTB is
end entity;


architecture behavioural of LetterSelectionTB is

    signal Clock      : std_logic := '0';
    signal Reset      : std_logic := '0';
    signal Selection  : std_logic_vector(3 downto 0) := "0000";
    signal Symbol     : integer range 0 to 15;
    signal LetterData : std_logic_vector(13 downto 0);
    signal LetterSize : integer range 0 to 15;

    signal Done   : std_logic;

    constant HalfPeriod : time := 10 ns;

begin

    Clock <= not Clock after HalfPeriod when Done /= '1' else '0';

    process
    begin
        Done <= '0';
        wait for 10 ns;
        Reset <= '1';
        wait for 100 ns;
        Selection <= "1000";
        wait for 100 ns;
        Selection <= "1111";
        wait for 100 ns;
        Selection <= "0011";
        wait for 100 ns;
        Reset <= '0';
        wait for 100 ns;
        Reset <= '1';
        wait for 100 ns;
        Done <= '1';
        wait;
    end process;

    Selector: entity work.LetterSelection(behavioural)
        port map(Clk          => Clock,
                 nRst         => Reset,
                 Selection    => Selection,
                 Symbol       => Symbol,
                 LetterData   => LetterData,
                 LetterSize   => LetterSize);

end architecture;
```

Code 4.7: Testbench for the Letter shift register

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

entity LetterRegisterTB is
end entity;


architecture behavioural of LetterRegisterTB is

    signal Reset      : std_logic := '0';
    signal Shift      : std_logic := '0';
    signal Load       : std_logic := '0';
    signal LetterLoad : std_logic_vector(13 downto 0) := (others => '0');
    signal SizeLoad   : integer range 0 to 15 := 0;
    signal LetterOut  : std_logic_vector(13 downto 0);
    signal SizeOut    : integer range 0 to 15;

begin

    process
    begin
        wait for 20 ns;
        Reset <= '1';
        wait for 20 ns;
        LetterLoad <= "11001100110011";
        SizeLoad <= 5;
        Load <= '1';
        wait for 20 ns;
        LetterLoad <= "00011011------";
        SizeLoad <= 7;
        wait for 20 ns;
        Load <= '0';
        wait for 20 ns;
        LetterLoad <= "00000000000000";
        SizeLoad <= 0;
        wait for 20 ns;
        Shift <= '1';
        wait for 20 ns;
        Shift <= '0';
        wait for 20 ns;
        Shift <= '1';
        wait for 20 ns;
        Reset <= '0';
        wait for 20 ns;
        wait;
    end process;

    ShiftRegister: entity work.LetterRegister(behavioural)
        port map(nRst         => Reset,
                 Shift        => Shift,
                 Load         => Load,
                 LetterLoad   => LetterLoad,
                 SizeLoad     => SizeLoad,
                 LetterOut    => LetterOut,
                 SizeOut      => SizeOut);

end architecture;
```

Code 4.8: Testbench for the 7 segment decoder

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

entity Decoder7SegmentTB is
end entity;


architecture behavioural of Decoder7SegmentTB is

    signal Reset    : std_logic := '0';
    signal Enable   : std_logic := '0';
    signal Load     : std_logic := '0';
    signal SymbolNr : integer range 0 to 15 := 0;
    signal HEX      : std_logic_vector(6 downto 0);

begin

    process
    begin
        wait for 20 ns;
        Reset <= '1';
        wait for 20 ns;
        SymbolNr <= 5;
        Load <= '1';
        wait for 20 ns;
        SymbolNr <= 3;
        wait for 20 ns;
        Load <= '0';
        Enable <= '1';
        wait for 20 ns;
        SymbolNr <= 7;
        wait for 20 ns;
        Enable <= '0';
        wait for 20 ns;
        Enable <= '1';
        wait for 20 ns;
        Load <= '1';
        wait for 20 ns;
        SymbolNr <= 10;
        wait for 20 ns;
        Reset <= '0';
        wait for 20 ns;
        wait;
    end process;


    -- Decoder for the 7 segment display output
    DisplayDecoder: entity work.Decoder7Segment(behavioural)
        port map(nRst         => Reset,
                 Enable       => Enable,
                 Load         => Load,
                 Symbol       => SymbolNr,
                 HEX          => HEX);

end architecture;
```

Code 4.9: Testbench for the modified Logic control

VHDL

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

entity GammaLogicTB is
end entity;


architecture behavioural of GammaLogicTB is

    signal Clock      : std_logic := '0';
    signal Reset      : std_logic := '0';
    signal Enable     : std_logic := '1';
    signal Shift      : std_logic;
    signal Load       : std_logic;
    signal Letter     : std_logic_vector(13 downto 0) := (others => '0');
    signal LetterSize : integer range 0 to 15 := 0;
    signal LED        : std_logic;
    signal Display    : std_logic;

    signal Done   : std_logic;

    constant HalfPeriod : time := 10 ns;

begin

    Clock <= not Clock after HalfPeriod when Done /= '1' else '0';

    process
    begin
        Done <= '0';
        wait for 20 ns;
        Reset <= '1';
        wait for 20 ns;
        Letter <= "0100100011----";
        LetterSize <= 5;
        wait for 20 ns;
        Enable <= '0';
        wait for 250 ns;
        Enable <= '1';
        Letter <= "00100011------";
        LetterSize <= 4;
        wait for 200 ns;
        Letter <= "100011--------";
        LetterSize <= 3;
        wait for 100 ns;
        Letter <= "0011----------";
        LetterSize <= 2;
        wait for 200 ns;
        Letter <= "11------------";
        LetterSize <= 1;
        wait for 250 ns;
        Letter <= "--------------";
        LetterSize <= 0;
        wait for 150 ns;
        Reset <= '0';
        wait for 100 ns;
        Done <= '1';
        wait;
    end process;
```

```
61
62      LogicUnit: entity work.GammaLogicFast(behavioural)
63          port map(Clk          => Clock,
64                   nRst         => Reset,
65                   nEnable      => Enable,
66                   Shift        => Shift,
67                   Load         => Load,
68                   Letter       => Letter,
69                   LetterSize   => LetterSize,
70                   LEDoutput    => LED,
71                   Display      => Display);
72
73  end architecture;
```

All test benches was made in a similar way. Testing both ordinary function and edge cases. Initially the test benches was much larger that the supplied code, this made for unreadable simulation results and therefore the test benches was simplified to the smallest amount of tests that could be done to be able to include the simulation results in this report.

The final testbench for the logic circuit also had a problem, it would just wait for at least 0.25s to move to the next state. This is unacceptable in a simulation tool. This is why a modified file is included, where the counter for the timer was reduced from counting up in the millions to at most 13. Here I represent 0.25s as two clock cycles instead of 125e5 cycles. There is no other change in code so this should work identical to the actual used code, only sped up for simulation purposes. For clarity sake the fast version is also included in the report and as a file.

Code 4.10: Logic control modified with a very small counter

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

entity GammaLogicFast is
    port(Clk        : in  std_logic;
         nRst       : in  std_logic;
         nEnable    : in  std_logic;
         Shift      : out std_logic;
         Load       : out std_logic;
         Letter     : in  std_logic_vector(13 downto 0);
         LetterSize : in  integer range 0 to 15;
         LEDoutput  : out std_logic;
         Display    : out std_logic);
end entity;


architecture behavioural of GammaLogicFast is

    type GammaState is (       -- Custom type with states of choice
        Idle, Running, DotOn, DotOff, Dash, Bar, Between);
    signal State : GammaState; -- State variable of the custom type
    signal TimeCounter : integer range 0 to 100; -- Low for simulation

begin

    process(Clk, nRst) is
    begin
        if rising_edge(Clk) then
            TimeCounter <= TimeCounter + 1; -- Time is short for
    simulation
            case State is    -- State change case for controlling states

                when Idle => -- Waiting for button press
                    if nEnable = '0' then
                        State <= Running;
                    end if;

                when Running => -- Routing signal on/off
                    if LetterSize = 0 then
                        State <= Idle;    -- Return to Idle when done
                    else                  -- Routes based on letter data
                        TimeCounter <= 0; -- Restarts counter each symbol
                        if    Letter(13 downto 12) = "00" then
                            State <= Between;
                        elsif Letter(13 downto 12) = "01" then
                            State <= DotOn;
                        elsif Letter(13 downto 12) = "10" then
                            State <= Dash;
                        elsif Letter(13 downto 12) = "11" then
                            State <= Bar;
                        else -- As a safety measure return to Idle
                            State <= Idle;
                        end if;
                    end if;

                when Between => -- Waiting 8 cycles between two symbols
                    if TimeCounter >= 7 then
                        State <= Running;
```

```
                    end if;

                when DotOn => -- Displaying dot . for 2 cycles 4 times
                    if    TimeCounter >= 13 then
                        State <= Running;
                    elsif (TimeCounter = 1  or
                           TimeCounter = 5  or
                           TimeCounter = 9) then
                        State <= DotOff;
                    end if;

                when DotOff => -- Turning dot off for 2 cycles 3 times
                    if (TimeCounter =  3  or
                        TimeCounter =  7  or
                        TimeCounter >= 11) then
                        State <= DotOn;
                    end if;

                when Dash => -- Displaying dash - for 6 cycles 1 time
                    if TimeCounter >= 5 then
                        State <= Running;
                    end if;

                when Bar => -- Displaying bar | for 12 cycles 1 time
                    if TimeCounter >= 11 then
                        State <= Running;
                    end if;
            end case;
        end if;

        if nRst = '0' then -- Asynchronous reset for state and counter
            State <= Idle;
            TimeCounter <= 0;
        end if;
    end process;

    process(State, nRst) is
    begin
        case State is -- Output case for controlling outputs

            when Idle =>
                LEDoutput <= '0';
                Display   <= '0';
                Shift     <= '0';
                Load      <= '1';

            when Running =>
                LEDoutput <= '0';
                Display   <= '1';
                Shift     <= '0';
                Load      <= '0';

            when Between =>
                LEDoutput <= '0';
                Display   <= '1';
                Shift     <= '1';
                Load      <= '0';

            when DotOn =>
                LEDoutput <= '1';
                Display   <= '1';
```

```
                Shift     <= '1';
                Load      <= '0';

            when DotOff =>
                LEDoutput <= '0';
                Display   <= '1';
                Shift     <= '1';
                Load      <= '0';

            when Dash =>
                LEDoutput <= '1';
                Display   <= '1';
                Shift     <= '1';
                Load      <= '0';

            when Bar =>
                LEDoutput <= '1';
                Display   <= '1';
                Shift     <= '1';
                Load      <= '0';
        end case;

        if nRst = '0' then -- Asynchronous reset for outputs
            LEDoutput <= '0';
            Display   <= '0';
            Shift     <= '0';
            Load      <= '0';
        end if;
    end process;

end architecture;
```

5- Results

5.1 Simulation

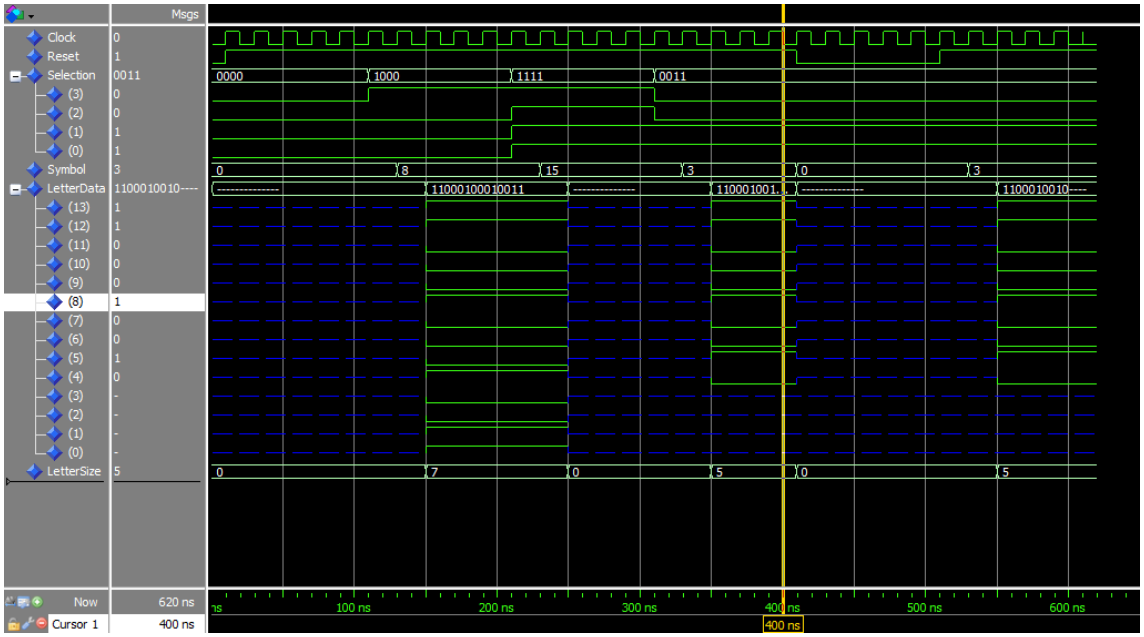

Figure 5.8: TB Selector

In this simulation it is clear to see that as long as reset is not pressed the value of Symbol, LetterData and LetterSize is following the input switches, with no delay for Symbol and one clock cycle for the others. This is because the 0th cycle upon changing the switches they are still read as old values, if in simulation these were changed only a single time unit before the clock edge rises it would be registers as the new value. on the 1st clock cycle after changing the value is read and stored in SymbolNr, then the instantly also saved in Symbol. The 2nd cycle is where the new saved value of SymbolNr is passing into the lookup table. This is because ModelSim simulates everything instantly for each time unit and therefore signal values don't change within a process like in traditional programming. When pressing the reset the value is instantly set no delay as it is asynchronous.

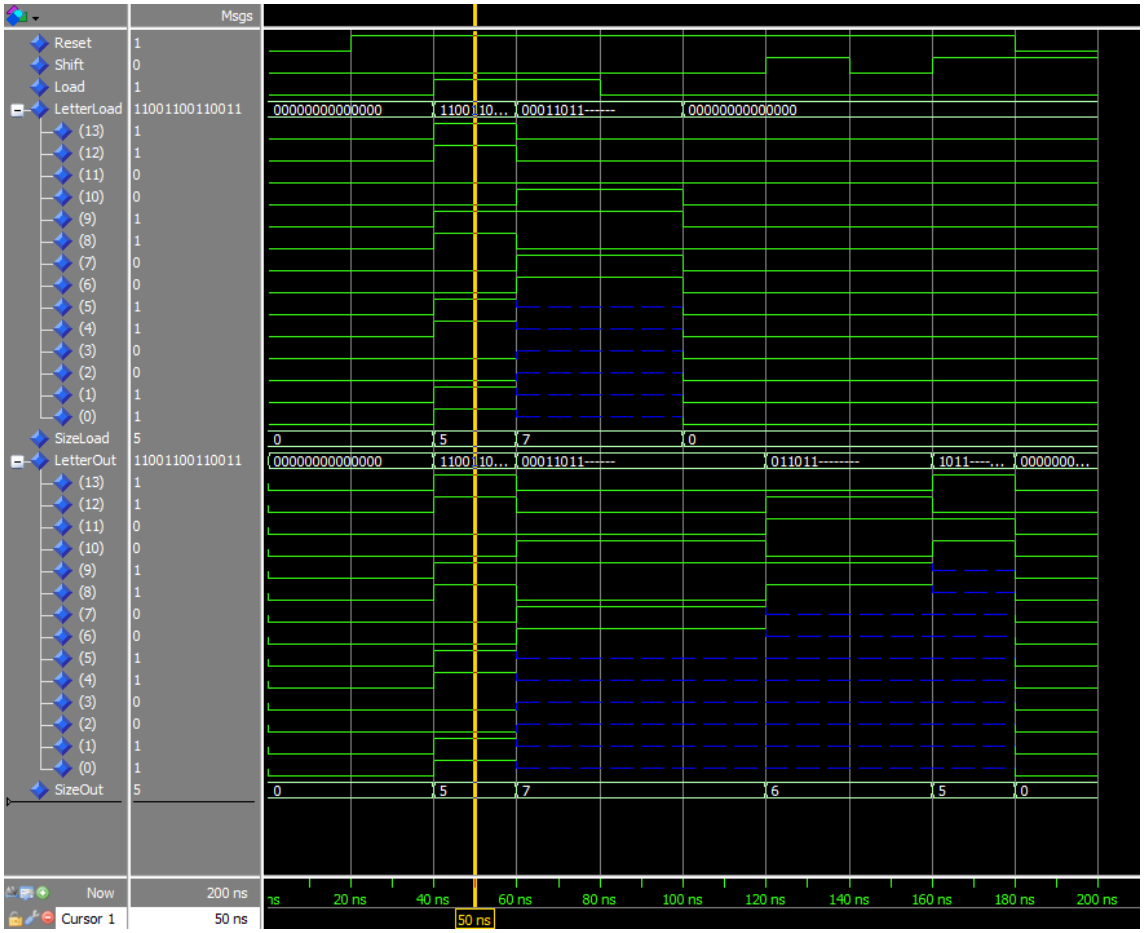

Figure 5.9: TB Register

In this simulation it is clear that when Load is high that it overrides current output instantly and when load is low again input values don't affect the output any more. It is also clear to see the shifting that happens on rising edge of Shift, where each time it adds the don't care and moves the values by two and counts down size by 1. The reset is asynchronous and clears output immediately when pressed.

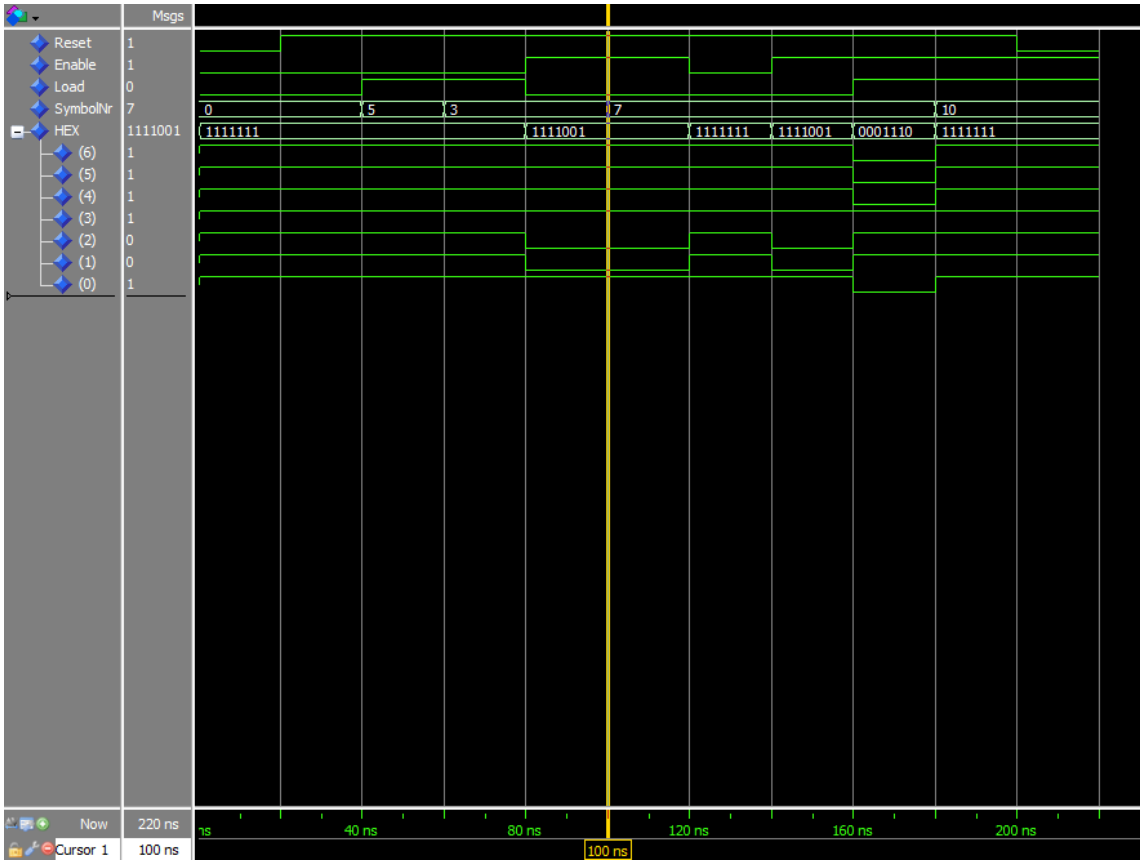

Figure 5.10: TB Decoder

In this simulation it is easy to see that the display is active whenever enable is high and not active when low. It is also possible to see how when Load is low and enable is high that it does not update the display. This is important as the display should remain the same regardless of switch positions throughout the sequence of symbols. Enable and Load are outputs from the FSM and is technically just not gated from the other, this means that the display should never change midway as it never goes to Idle unless done.

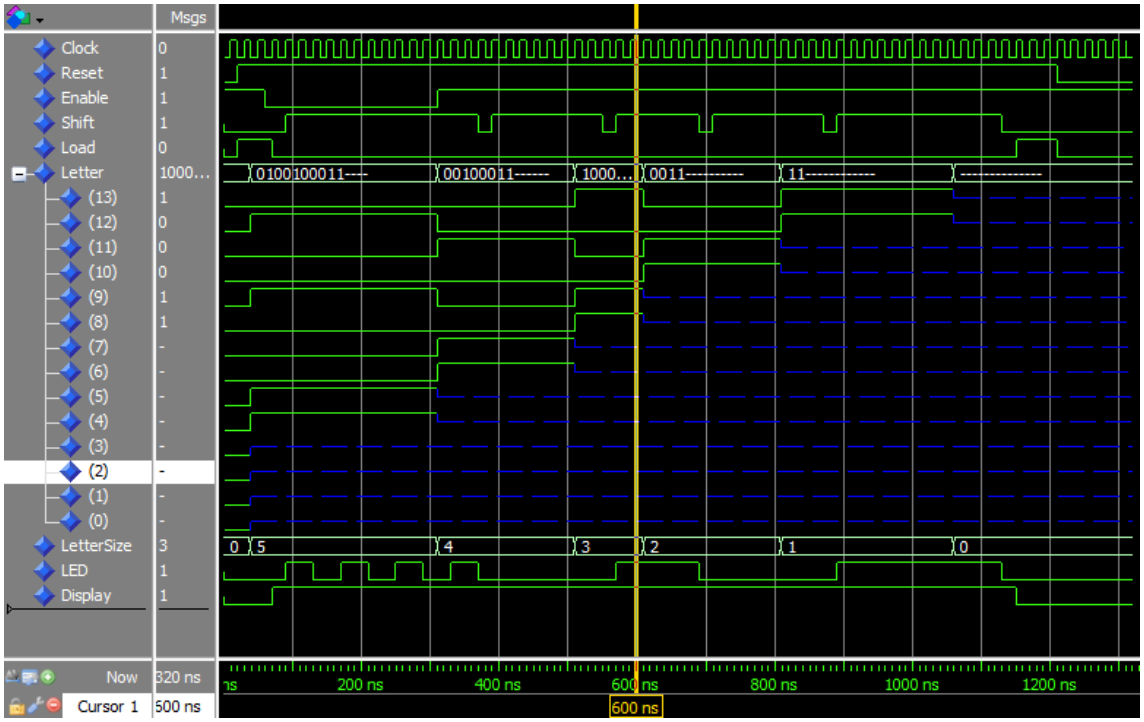

Figure 5.11: TB Logic

This is the most interesting simulation where at the bottom the dot, dash, bar and between can easily be observed. This is only possible because of simulating using the counter modified logic code. Looking at enable, the start sending keypress, it activates the logic sequence to happen and the FSM moves along and shifts the LetterData as it goes. This shifting of data is done manually in the test bench as the logic is not connected to the shift register during the test bench. At the end it goes back to the idle state, signified by load being turned high. And when pressing reset all outputs are set to zero.

5.2 Implementation

Implementation was done on the FPGA board and a video file uploaded showing it working as intended trying every combination and some more.

6- Conclusion

In the end I learned a lot from doing this Project. I also felt a high level of understanding and mastery as I was able to write all the code in its entirety without any simulation or compilation and only trying to compile it 7 times and fixing minor errors such as missing semicolons or reserved names before it worked as intended, with no change to the actual logic. I think there are a few areas of improvement that can be done with my code. I think removing the 00 for between in the LetterData and maybe have a boolean for ”have waited” true or false, that would be false when returning to running from a symbol state and after being in the between state it goes to true and is then proceed to the next symbol state. This would also have to be true when coming from idle to not introduce a start delay, and it needs to be checked after size is checked to not add end delay. Other improvements I have thought about is cleaning up the code. I have tried to make the code as clean as possible but I still fell like the signal names and module names are not the most intuitive names and could be a bit confusing. The final improvement that could be made is to remove the timer counter from the logic and make a resettable ticker counter that gives a 1 cycle synchronous pulse that can be read by the state transition logic. Then the Running state could reset it whenever initiated and use a small counter to keep track of each states length. Although I am on the fence about that as I like that having it clocked to 50MHz is for circuits like this essentially the same as having actions happen instantly. There is no input delay and transitions happen immediately, unlike a 4Hz clock where if timed bad could give a ”whole” 0.249s input delay, maybe this even causes the input to not register if fast enough. All in all I am happy with the way I solved the task and how the code works and I personally don’t think I can improve it by much without adding other problems or complications that makes it not as clean of a result. ”In anything at all, perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away, when a body has been stripped down to its nakedness.” by Antoine de Saint-Exupery. This is a quote I really like and try to think about when designing anything such that it functions in the most simple and elegant way I can think of. I am by no means saying this code is absolute perfection, but it in my vision it cannot be simplified any more to my knowledge and thereby I am happy about my design.