

# 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-in - to generate  $\text{\LaTeX}$ code 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  $\text{\LaTeX}$

## **1.1 Table of Contents**

|                                |           |
|--------------------------------|-----------|
| <b>2- Approach to solution</b> | <b>4</b>  |
| 2.1 Introduction .....         | 4         |
| 2.2 The Problem .....          | 4         |
| 2.3 Definitions .....          | 4         |
| 2.4 Solving .....              | 5         |
| 2.5 Structure .....            | 5         |
| <b>3- The FSM</b>              | <b>6</b>  |
| 3.1 State logic design .....   | 6         |
| 3.2 State tables .....         | 8         |
| 3.3 Moore and Mealy .....      | 8         |
| <b>4- VHDL code</b>            | <b>9</b>  |
| 4.1 FSM code .....             | 9         |
| 4.2 RTL view .....             | 18        |
| 4.3 Test bench code .....      | 22        |
| <b>5- Results</b>              | <b>30</b> |
| 5.1 Simulation .....           | 30        |
| 5.2 Implementation .....       | 33        |
| <b>6- Conclusion</b>           | <b>34</b> |

## 1.2 List of Figures

|                                                                       |    |
|-----------------------------------------------------------------------|----|
| Figure 3.1: Encoding table made by Quartus . . . . .                  | 6  |
| Figure 3.2: State transition diagram . . . . .                        | 7  |
| Code 4.1: TLE that ties the system together . . . . .                 | 9  |
| Code 4.2: Letter selector is a lookup table . . . . .                 | 10 |
| Code 4.3: Custom loadable shift register . . . . .                    | 12 |
| Code 4.4: Decoder with load for 7-segment display output . . . . .    | 13 |
| Code 4.5: Logic control deciding FSM transitions and output . . . . . | 15 |
| Figure 4.3: RTL of the TLE . . . . .                                  | 18 |
| Figure 4.4: RTL of the letter selector component . . . . .            | 19 |
| Figure 4.5: RTL of the letter shift register component . . . . .      | 20 |
| Figure 4.6: RTL of the 7 segment decoder component . . . . .          | 21 |
| Figure 4.7: RTL of the logic component . . . . .                      | 21 |
| Code 4.6: Testbench for the Letter selector . . . . .                 | 22 |
| Code 4.7: Testbench for the Letter shift register . . . . .           | 23 |
| Code 4.8: Testbench for the 7 segment decoder . . . . .               | 24 |
| Code 4.9: Testbench for the modified Logic control . . . . .          | 25 |
| Code 4.10: Logic control modified with a very small counter . . . . . | 27 |
| Figure 5.8: TB Selector . . . . .                                     | 30 |
| Figure 5.9: TB Register . . . . .                                     | 31 |
| Figure 5.10: TB Decoder . . . . .                                     | 32 |
| Figure 5.11: TB Logic . . . . .                                       | 33 |

## 1.3 List of Tables

|                                          |    |
|------------------------------------------|----|
| 3.1 State transitions . . . . .          | 8  |
| 3.2 State outputs . . . . .              | 8  |
| 4.3 Output string from decoder . . . . . | 14 |

## **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

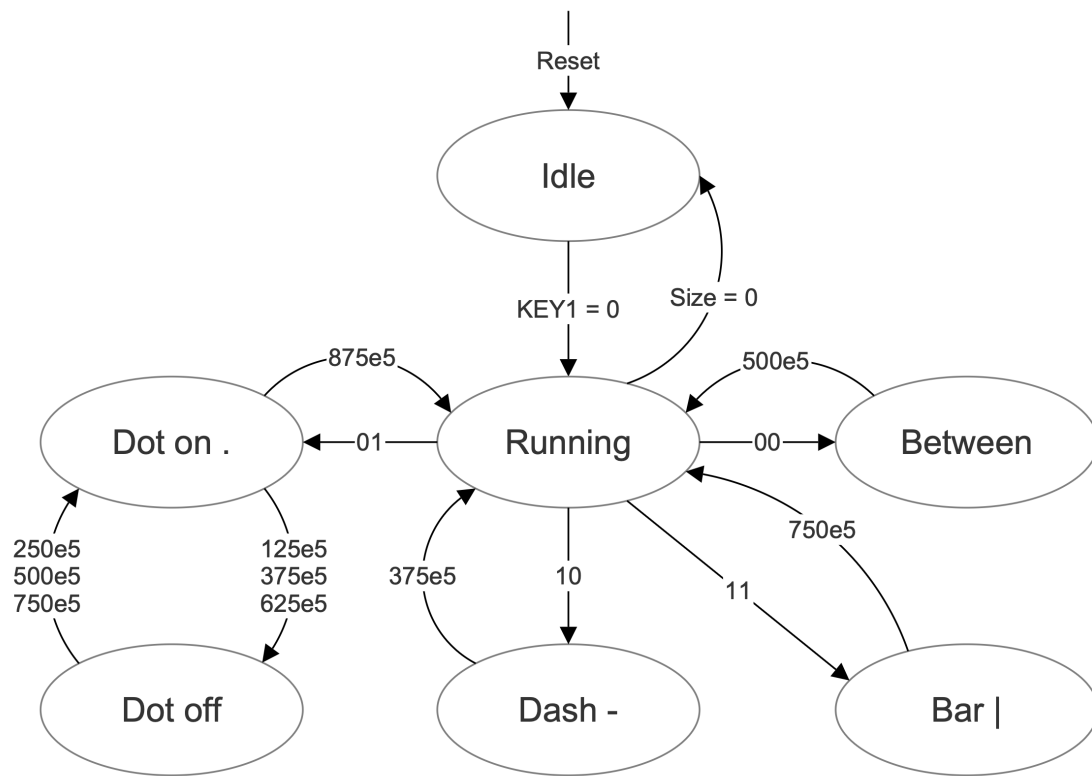


Figure 3.2: State transition diagram

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                                                         |
| 1  | library ieee;                                                |
| 2  | use ieee.std_logic_1164.all;                                 |
| 3  | use ieee.numeric_std.all;                                    |
| 4  |                                                              |
| 5  | entity FinalProject is                                       |
| 6  | port(CLOCK_50 : in std_logic;                                |
| 7  | SW : in std_logic_vector(3 downto 0);                        |
| 8  | KEY : in std_logic_vector(1 downto 0);                       |
| 9  | LEDR : out std_logic_vector(9 downto 0);                     |
| 10 | HEX0 : out std_logic_vector(6 downto 0));                    |
| 11 | end entity;                                                  |
| 12 |                                                              |
| 13 |                                                              |
| 14 | architecture behavioural of FinalProject is                  |
| 15 |                                                              |
| 16 | signal startup : std_logic := '1';                           |
| 17 | signal Reset, Load, Shift, Display : std_logic;              |
| 18 | signal Symbol, LetterSize, SizeLoad : integer range 0 to 15; |
| 19 | signal Letter, LetterLoad : std_logic_vector(13 downto 0);   |
| 20 | -- All signals except startup is to connect modules          |
| 21 | begin                                                        |
| 22 |                                                              |
| 23 | LEDR(9 downto 4) <= (others => '0');                         |
| 24 | LEDR(1 downto 0) <= (others => '0');                         |
| 25 | LEDR(2) <= not KEY(0);                                       |
| 26 |                                                              |
| 27 | process(CLOCK_50) is -- Self resetting on startup            |
| 28 | begin                                                        |
| 29 | if rising_edge(CLOCK_50) and startup = '1' then              |
| 30 | startup <= '0';                                              |
| 31 | end if;                                                      |
| 32 | end process;                                                 |
| 33 |                                                              |
| 34 | Reset <= KEY(0) and not(startup);                            |
| 35 |                                                              |
| 36 | -- Transforms switch selection to data                       |
| 37 | Selector: entity work.LetterSelection(behavioural)           |
| 38 | port map(Clk => CLOCK_50,                                    |
| 39 | nRst => Reset,                                               |
| 40 | Selection => SW(3 downto 0),                                 |
| 41 | Symbol => Symbol,                                            |
| 42 | LetterData => LetterLoad,                                    |
| 43 | LetterSize => SizeLoad);                                     |
| 44 |                                                              |
| 45 | -- Saves selection and shifts data                           |
| 46 | ShiftRegister: entity work.LetterRegister(behavioural)       |
| 47 | port map(nRst => Reset,                                      |
| 48 | Shift => Shift,                                              |
| 49 | Load => Load,                                                |
| 50 | LetterLoad => LetterLoad,                                    |
| 51 | SizeLoad => SizeLoad,                                        |
| 52 | LetterOut => Letter,                                         |
| 53 | SizeOut => LetterSize);                                      |
| 54 |                                                              |

```

55 -- Decoder for the 7 segment display output
56 DisplayDecoder: entity work.Decoder7Segment(behavioural)
57     port map(nRst      => Reset,
58             Enable     => Display,
59             Load       => Load,
60             Symbol     => Symbol,
61             HEX        => HEX0);
62
63 -- Controls the logic with states and controls register and output
64 LogicUnit: entity work.GammaLogic(behavioural)
65     port map(Clk       => CLOCK_50,
66             nRst       => Reset,
67             nEnable    => KEY(1),
68             Shift      => Shift,
69             Load       => Load,
70             Letter     => Letter,
71             LetterSize  => LetterSize,
72             LEDoutput  => LEDR(3),
73             Display    => Display);
74
75 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

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity LetterSelection is
6      port(Clk      : in std_logic;
7           nRst     : in std_logic;
8           Selection : in std_logic_vector(3 downto 0);
9           Symbol   : out integer range 0 to 15;
10          LetterData : out std_logic_vector(13 downto 0);
11          LetterSize : out integer range 0 to 15);
12 end entity;
13
14
15 architecture behavioural of LetterSelection is
16
17     signal SymbolNr : integer range 0 to 15 := 0;
18
19 begin
20
21     Symbol <= SymbolNr; -- Avoids reading the output in the case when
22
23     process(Clk, nRst)
24     begin
25         if rising_edge(Clk) then
26             SymbolNr <= to_integer(unsigned(Selection));
27             case SymbolNr is -- Data and size of letters stored here
28                 when 1 => -- P as - - .

```

VHDL

```

29         LetterData <= "1000100001----"; -- 00 between symbols
30         LetterSize <= 5; -- 01 for dots
31         when 2 => -- B as | . . -- 10 for dash
32             LetterData <= "1100010001----"; -- 11 for vertical bar
33             LetterSize <= 5; -- -- for don't care
34         when 3 => -- 1 as | . -
35             LetterData <= "1100010010----";
36             LetterSize <= 5;
37         when 4 => -- 8 as . | .
38             LetterData <= "0100110001----";
39             LetterSize <= 5;
40         when 5 => -- 0 as - . -
41             LetterData <= "1000010010----";
42             LetterSize <= 5;
43         when 6 => -- - as -
44             LetterData <= "10-----";
45             LetterSize <= 1;
46         when 7 => -- F as | - .
47             LetterData <= "1100100001----";
48             LetterSize <= 5;
49         when 8 => -- G as | . . |
50             LetterData <= "11000100010011";
51             LetterSize <= 7;
52         when 9 => -- A as . - .
53             LetterData <= "0100100001----";
54             LetterSize <= 5;
55         when others => -- Whenever the input is something else
56             LetterData <= (others => '-');
57             LetterSize <= 0;
58     end case;
59 end if;
60
61     if nRst = '0' then -- Asynchronous reset for outputs
62         SymbolNr <= 0;
63         LetterData <= (others => '-');
64         LetterSize <= 0;
65     end if;
66 end process;
67 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 between. For the letters not using 4 symbols the remaining string is filled with "-" which is the don't care signal.

Code 4.3: Custom loadable shift register

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| <pre> 1  library ieee; 2  use ieee.std_logic_1164.all; 3  use ieee.numeric_std.all; 4 5  entity LetterRegister is 6      port(nRst      : in  std_logic; 7           Shift     : in  std_logic; 8           Load      : in  std_logic; 9           LetterLoad : in  std_logic_vector(13 downto 0); 10          SizeLoad   : in  integer range 0 to 15; 11          LetterOut  : out std_logic_vector(13 downto 0); 12          SizeOut    : out integer range 0 to 15); 13 end entity; 14 15 16 architecture behavioural of LetterRegister is 17 18     signal Letter : std_logic_vector(13 downto 0) := (others =&gt; '0'); 19     signal Size   : integer range 0 to 15 := 0; 20 21 begin 22 23     LetterOut &lt;= Letter; -- Avoids reading the output while shifting 24     SizeOut   &lt;= Size;   -- Avoids reading the output while decreasing 25 26     process(Shift, Load, LetterLoad, nRst) is 27     begin -- LetterLoad in sensitivity list to update value while idle 28         if rising_edge(Shift) then -- Shift acts as a clk signal 29             Letter &lt;= Letter (11 downto 0) &amp; "--"; 30             Size   &lt;= Size - 1; 31         end if; 32         if Load = '1' then -- Always updates on input change 33             Letter &lt;= LetterLoad; 34             Size   &lt;= SizeLoad; 35         end if; 36         if nRst = '0' then -- Asynchronous reset for outputs 37             Letter &lt;= (others =&gt; '0'); 38             Size   &lt;= 0; 39         end if; 40     end process; 41 42 end architecture; </pre> | VHDL |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|

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

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Decoder7Segment is
6      port(nRst   : in  std_logic;
7           Enable : in  std_logic;
8           Load   : in  std_logic;
9           Symbol  : in  integer range 0 to 15;
10          HEX     : out std_logic_vector(6 downto 0));
11 end entity;
12
13
14 architecture behavioural of Decoder7Segment is
15
16     signal SavedSymbol : integer range 0 to 15;
17
18 begin
19
20     process(Load, Symbol, nRst)
21     begin -- Symbol in sensitivity list to continually update on input
22         if Load = '1' then
23             SavedSymbol <= Symbol;
24         end if;
25
26         if nRst = '0' then -- Asynchronous reset for saved symbol
27             SavedSymbol <= 0;
28         end if;
29     end process;
30
31     process(Enable, SavedSymbol, nRst)
32     begin -- SavedSymbol in sensitivity list if it updates after Enable
33         if Enable = '1' then
34             case SavedSymbol is
35                 when 1 => -- P
36                     HEX <= "0001100";
37                 when 2 => -- B
38                     HEX <= "0000011";
39                 when 3 => -- 1
40                     HEX <= "1111001";
41                 when 4 => -- 8
42                     HEX <= "0000000";
43                 when 5 => -- 0
44                     HEX <= "1000000";
45                 when 6 => -- -
46                     HEX <= "0111111";
47                 when 7 => -- F
48                     HEX <= "0001110";
49                 when 8 => -- G
50                     HEX <= "1000010";
51                 when 9 => -- A
52                     HEX <= "0001000";
53                 when others => -- Off for undefined symbols
54                     HEX <= (others => '1');
55             end case;
56         else -- Off if display is not enabled
57             HEX <= (others => '1');
58         end if;
59     end process;

```

```

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

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity GammaLogic is
6      port(Clk          : in  std_logic;
7            nRst         : in  std_logic;
8            nEnable      : in  std_logic;
9            Shift        : out std_logic;
10           Load         : out std_logic;
11           Letter       : in  std_logic_vector(13 downto 0);
12           LetterSize   : in  integer range 0 to 15;
13           LEDoutput    : out std_logic;
14           Display      : out std_logic);
15 end entity;
16
17
18 architecture behavioural of GammaLogic is
19
20     type GammaState is (      -- Custom type with states of choice
21         Idle, Running, DotOn, DotOff, Dash, Bar, Between);
22     signal State : GammaState; -- State variable of the custom type
23     signal TimeCounter : integer range 0 to 1000e5; -- Within 27 bits
24
25 begin
26
27     process(Clk, nRst) is
28     begin
29         if rising_edge(Clk) then
30             TimeCounter <= TimeCounter + 1; -- Times based on 50Mhz clock
31             case State is -- State change case for controlling states
32
33                 when Idle => -- Waiting for button press
34                     if nEnable = '0' then
35                         State <= Running;
36                     end if;
37
38                 when Running => -- Routing signal on/off
39                     if LetterSize = 0 then
40                         State <= Idle; -- Return to Idle when done
41                     else -- Routes based on letter data
42                         TimeCounter <= 0; -- Restarts counter each symbol
43                         if Letter(13 downto 12) = "00" then
44                             State <= Between;
45                         elsif Letter(13 downto 12) = "01" then
46                             State <= DotOn;
47                         elsif Letter(13 downto 12) = "10" then
48                             State <= Dash;
49                         elsif Letter(13 downto 12) = "11" then
50                             State <= Bar;
51                         else -- As a safety measure return to Idle
52                             State <= Idle;
53                         end if;
54                     end if;
55
56                 when Between => -- Waiting 1s between two symbols
57                     if TimeCounter >= 500e5 then
58                         State <= Running;
59                     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.

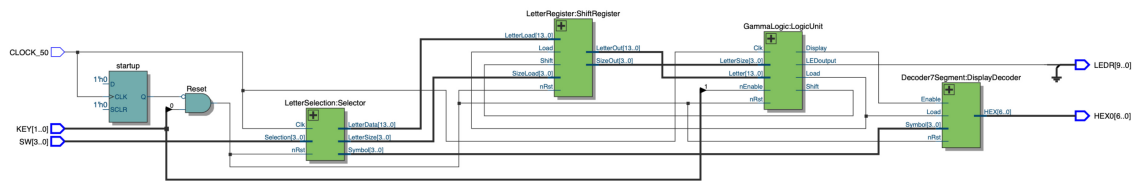


Figure 4.3: RTL of the TLE

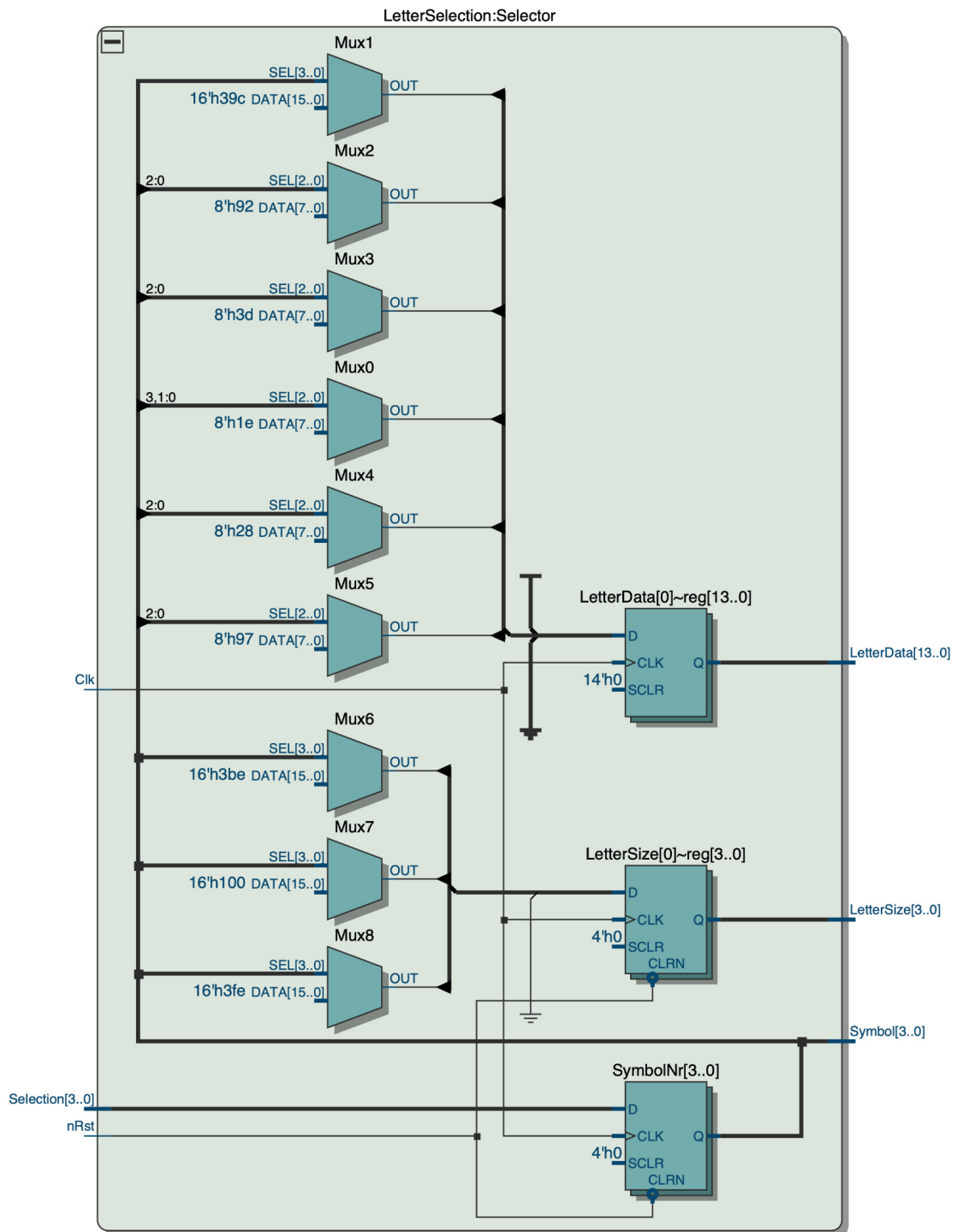


Figure 4.4: RTL of the letter selector component



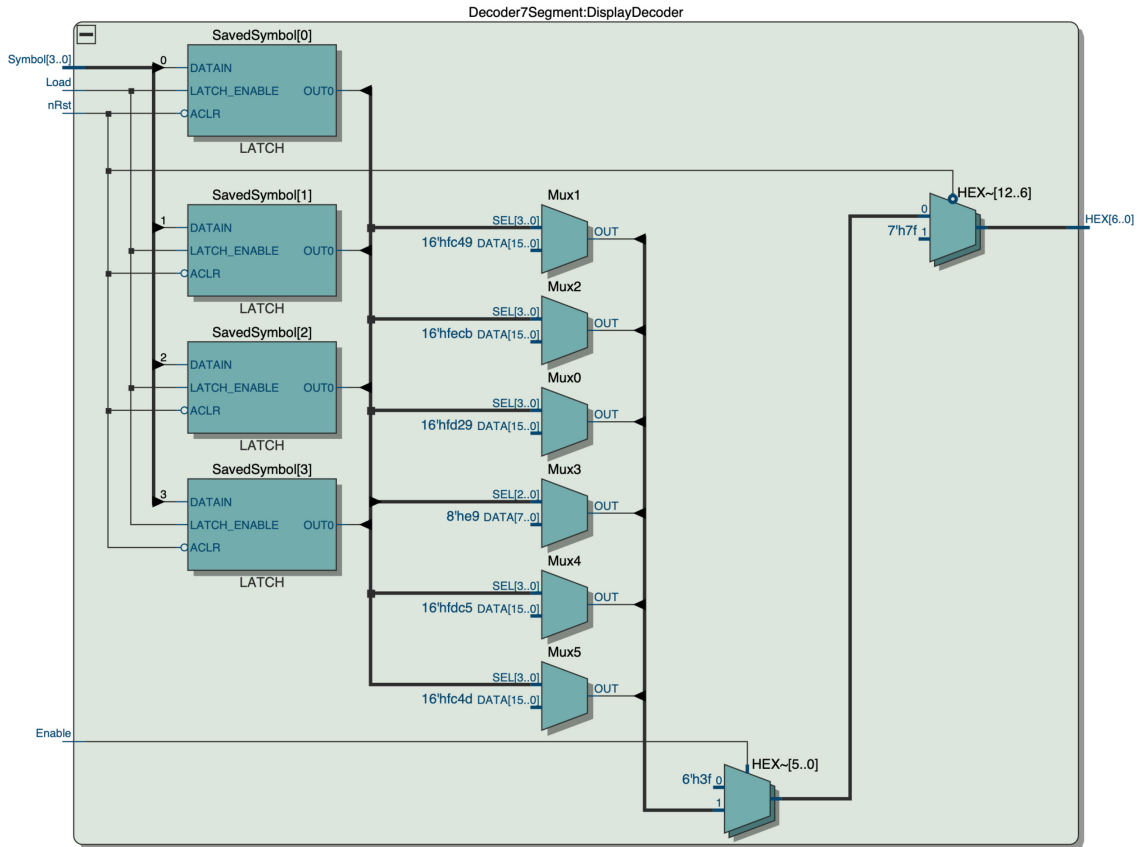


Figure 4.6: RTL of the 7 segment decoder component

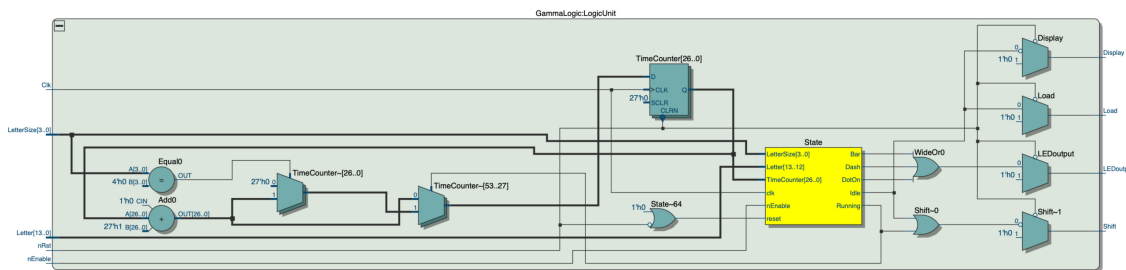


Figure 4.7: RTL of the logic component

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

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity LetterSelectionTB is
6  end entity;
7
8
9  architecture behavioural of LetterSelectionTB is
10
11     signal Clock      : std_logic := '0';
12     signal Reset      : std_logic := '0';
13     signal Selection   : std_logic_vector(3 downto 0) := "0000";
14     signal Symbol      : integer range 0 to 15;
15     signal LetterData  : std_logic_vector(13 downto 0);
16     signal LetterSize  : integer range 0 to 15;
17
18     signal Done        : std_logic;
19
20     constant HalfPeriod : time := 10 ns;
21
22 begin
23
24     Clock <= not Clock after HalfPeriod when Done /= '1' else '0';
25
26     process
27     begin
28         Done <= '0';
29         wait for 10 ns;
30         Reset <= '1';
31         wait for 100 ns;
32         Selection <= "1000";
33         wait for 100 ns;
34         Selection <= "1111";
35         wait for 100 ns;
36         Selection <= "0011";
37         wait for 100 ns;
38         Reset <= '0';
39         wait for 100 ns;
40         Reset <= '1';
41         wait for 100 ns;
42         Done <= '1';
43         wait;
44     end process;
45
46     Selector: entity work.LetterSelection(behavioural)
47         port map(Clk      => Clock,
48                 nRst      => Reset,
49                 Selection  => Selection,
50                 Symbol     => Symbol,
51                 LetterData  => LetterData,
52                 LetterSize  => LetterSize);
53
54 end architecture;
```

Code 4.7: Testbench for the Letter shift register

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity LetterRegisterTB is
6  end entity;
7
8
9  architecture behavioural of LetterRegisterTB is
10
11     signal Reset      : std_logic := '0';
12     signal Shift       : std_logic := '0';
13     signal Load        : std_logic := '0';
14     signal LetterLoad  : std_logic_vector(13 downto 0) := (others => '0');
15     signal SizeLoad    : integer range 0 to 15 := 0;
16     signal LetterOut   : std_logic_vector(13 downto 0);
17     signal SizeOut     : integer range 0 to 15;
18
19  begin
20
21     process
22     begin
23         wait for 20 ns;
24         Reset <= '1';
25         wait for 20 ns;
26         LetterLoad <= "11001100110011";
27         SizeLoad <= 5;
28         Load <= '1';
29         wait for 20 ns;
30         LetterLoad <= "00011011-----";
31         SizeLoad <= 7;
32         wait for 20 ns;
33         Load <= '0';
34         wait for 20 ns;
35         LetterLoad <= "00000000000000";
36         SizeLoad <= 0;
37         wait for 20 ns;
38         Shift <= '1';
39         wait for 20 ns;
40         Shift <= '0';
41         wait for 20 ns;
42         Shift <= '1';
43         wait for 20 ns;
44         Reset <= '0';
45         wait for 20 ns;
46         wait;
47     end process;
48
49     ShiftRegister: entity work.LetterRegister(behavioural)
50     port map(nRst      => Reset,
51             Shift      => Shift,
52             Load       => Load,
53             LetterLoad  => LetterLoad,
54             SizeLoad    => SizeLoad,
55             LetterOut   => LetterOut,
56             SizeOut     => SizeOut);
57
58  end architecture;

```

Code 4.8: Testbench for the 7 segment decoder

VHDL

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Decoder7SegmentTB is
6  end entity;
7
8
9  architecture behavioural of Decoder7SegmentTB is
10
11     signal Reset      : std_logic := '0';
12     signal Enable     : std_logic := '0';
13     signal Load       : std_logic := '0';
14     signal SymbolNr   : integer range 0 to 15 := 0;
15     signal HEX        : std_logic_vector(6 downto 0);
16
17 begin
18
19     process
20     begin
21         wait for 20 ns;
22         Reset <= '1';
23         wait for 20 ns;
24         SymbolNr <= 5;
25         Load <= '1';
26         wait for 20 ns;
27         SymbolNr <= 3;
28         wait for 20 ns;
29         Load <= '0';
30         Enable <= '1';
31         wait for 20 ns;
32         SymbolNr <= 7;
33         wait for 20 ns;
34         Enable <= '0';
35         wait for 20 ns;
36         Enable <= '1';
37         wait for 20 ns;
38         Load <= '1';
39         wait for 20 ns;
40         SymbolNr <= 10;
41         wait for 20 ns;
42         Reset <= '0';
43         wait for 20 ns;
44         wait;
45     end process;
46
47
48     -- Decoder for the 7 segment display output
49     DisplayDecoder: entity work.Decoder7Segment(behavioural)
50         port map(nRst      => Reset,
51                 Enable    => Enable,
52                 Load      => Load,
53                 Symbol    => SymbolNr,
54                 HEX       => HEX);
55
56 end architecture;

```

Code 4.9: Testbench for the modified Logic control



```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity GammaLogicTB is
6  end entity;
7
8
9  architecture behavioural of GammaLogicTB is
10
11     signal Clock      : std_logic := '0';
12     signal Reset      : std_logic := '0';
13     signal Enable     : std_logic := '1';
14     signal Shift      : std_logic;
15     signal Load       : std_logic;
16     signal Letter     : std_logic_vector(13 downto 0) := (others => '0');
17     signal LetterSize : integer range 0 to 15 := 0;
18     signal LED        : std_logic;
19     signal Display    : std_logic;
20
21     signal Done       : std_logic;
22
23     constant HalfPeriod : time := 10 ns;
24
25 begin
26
27     Clock <= not Clock after HalfPeriod when Done /= '1' else '0';
28
29     process
30     begin
31         Done <= '0';
32         wait for 20 ns;
33         Reset <= '1';
34         wait for 20 ns;
35         Letter <= "0100100011----";
36         LetterSize <= 5;
37         wait for 20 ns;
38         Enable <= '0';
39         wait for 250 ns;
40         Enable <= '1';
41         Letter <= "00100011-----";
42         LetterSize <= 4;
43         wait for 200 ns;
44         Letter <= "100011-----";
45         LetterSize <= 3;
46         wait for 100 ns;
47         Letter <= "0011-----";
48         LetterSize <= 2;
49         wait for 200 ns;
50         Letter <= "11-----";
51         LetterSize <= 1;
52         wait for 250 ns;
53         Letter <= "-----";
54         LetterSize <= 0;
55         wait for 150 ns;
56         Reset <= '0';
57         wait for 100 ns;
58         Done <= '1';
59         wait;
60     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

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity GammaLogicFast is
6      port(Clk      : in  std_logic;
7            nRst     : in  std_logic;
8            nEnable  : in  std_logic;
9            Shift    : out std_logic;
10           Load     : out std_logic;
11           Letter    : in  std_logic_vector(13 downto 0);
12           LetterSize : in  integer range 0 to 15;
13           LEDoutput : out std_logic;
14           Display   : out std_logic);
15 end entity;
16
17
18 architecture behavioural of GammaLogicFast is
19
20     type GammaState is (      -- Custom type with states of choice
21         Idle, Running, DotOn, DotOff, Dash, Bar, Between);
22     signal State : GammaState; -- State variable of the custom type
23     signal TimeCounter : integer range 0 to 100; -- Low for simulation
24
25 begin
26
27     process(Clk, nRst) is
28     begin
29         if rising_edge(Clk) then
30             TimeCounter <= TimeCounter + 1; -- Time is short for
simulation
31             case State is      -- State change case for controlling states
32
33                 when Idle => -- Waiting for button press
34                     if nEnable = '0' then
35                         State <= Running;
36                     end if;
37
38                 when Running => -- Routing signal on/off
39                     if LetterSize = 0 then
40                         State <= Idle;      -- Return to Idle when done
41                     else
42                         -- Routes based on letter data
43                         TimeCounter <= 0; -- Restarts counter each symbol
44                         if Letter(13 downto 12) = "00" then
45                             State <= Between;
46                         elsif Letter(13 downto 12) = "01" then
47                             State <= DotOn;
48                         elsif Letter(13 downto 12) = "10" then
49                             State <= Dash;
50                         elsif Letter(13 downto 12) = "11" then
51                             State <= Bar;
52                         else -- As a safety measure return to Idle
53                             State <= Idle;
54                         end if;
55                     end if;
56
57                 when Between => -- Waiting 8 cycles between two symbols
58                     if TimeCounter >= 7 then
59                         State <= Running;
60                     end if;
61                 end case;
62             end if;
63         end process;
64
65     -- Output logic
66     Shift <= nEnable;
67     Load <= nRst;
68     LEDoutput <= State = Running;
69     Display <= State = Running;
70
71 end architecture;

```

```

59         end if;
60
61         when DotOn => -- Displaying dot . for 2 cycles 4 times
62             if TimeCounter >= 13 then
63                 State <= Running;
64             elsif (TimeCounter = 1 or
65                   TimeCounter = 5 or
66                   TimeCounter = 9) then
67                 State <= DotOff;
68             end if;
69
70         when DotOff => -- Turning dot off for 2 cycles 3 times
71             if (TimeCounter = 3 or
72                 TimeCounter = 7 or
73                 TimeCounter >= 11) then
74                 State <= DotOn;
75             end if;
76
77         when Dash => -- Displaying dash - for 6 cycles 1 time
78             if TimeCounter >= 5 then
79                 State <= Running;
80             end if;
81
82         when Bar => -- Displaying bar | for 12 cycles 1 time
83             if TimeCounter >= 11 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;

```

## 5- Results

### 5.1 Simulation

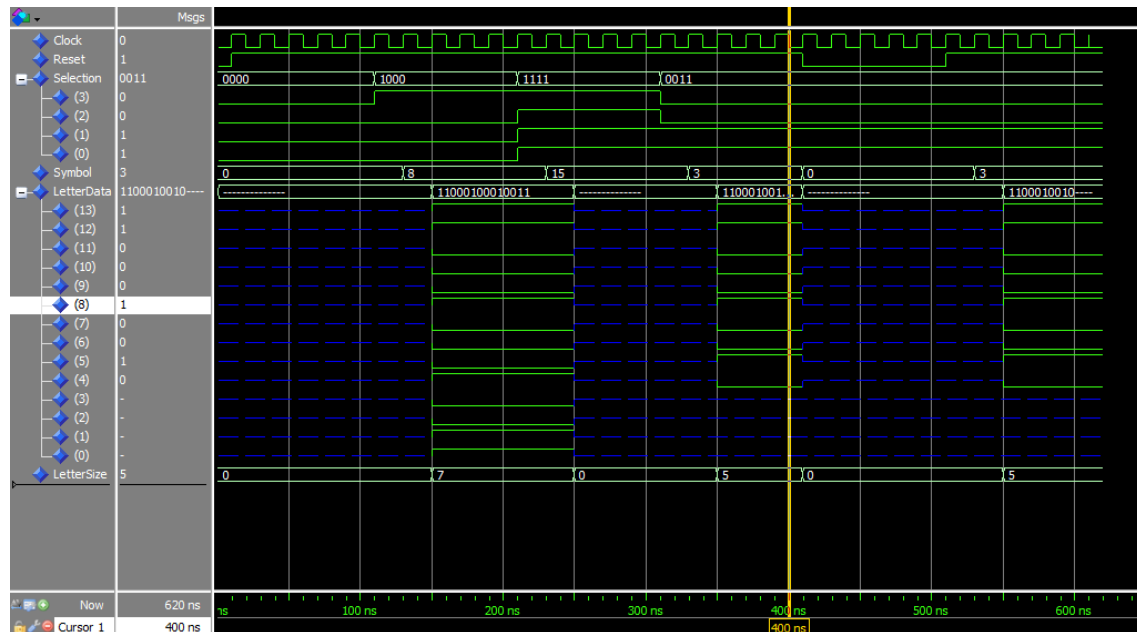


Figure 5.8: TB Selector

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.



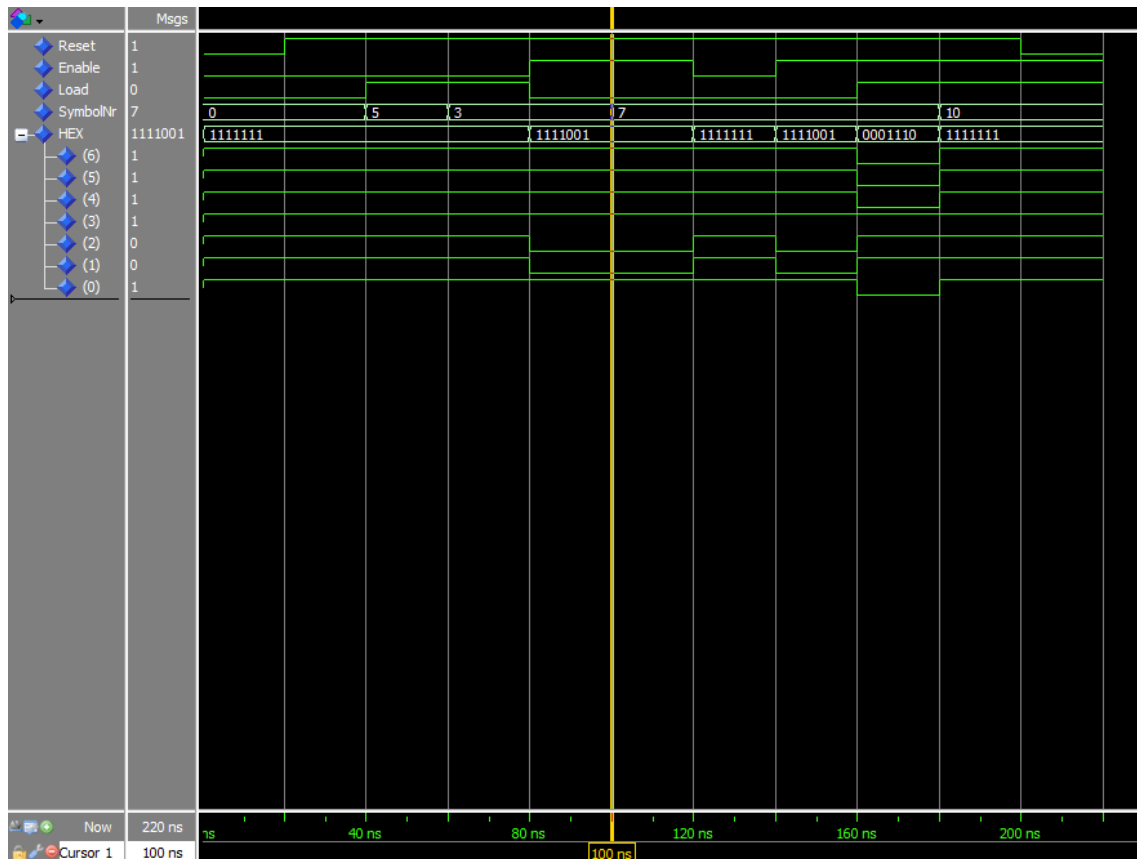


Figure 5.10: TB Decoder

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.



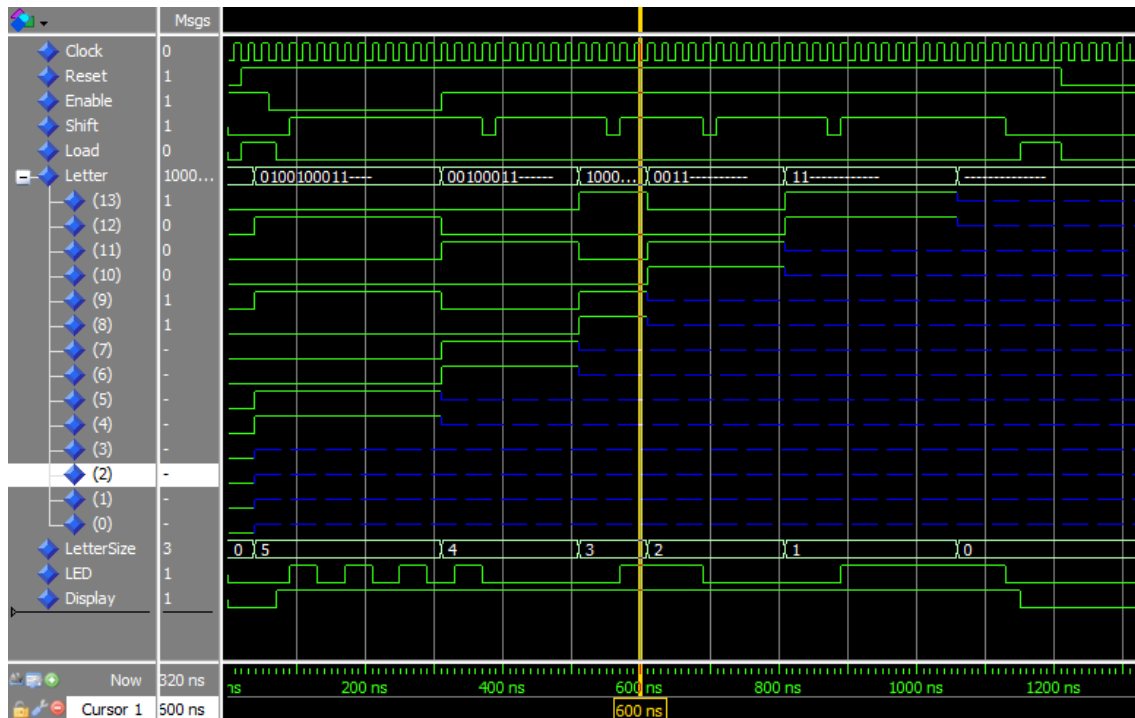


Figure 5.11: TB Logic

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.