**Lab:\_\_6\_\_ Report Evaluation Form: EE 316 Computer Engineering Junior Lab: Spring 2016**

**Report Writer \_\_\_James Bruska\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Grade \_\_\_\_\_\_\_/50**

**I. Report Content (40 points): For each area below, 0 points are given if it** is **completely missing.**

**A. Summary (Executive summary or abstract) \_\_\_\_/4**

Should summarize the design problem, the overall approach, and the results in terms of how well the specifications were met.

\_\_\_\_Excellent \_\_\_\_\_Good \_\_\_\_\_Average \_\_\_\_\_Fair \_\_\_\_\_None

**B. Design Problem Statement (Requirements and Specifications)** **\_\_\_\_/4**  
Should specify what was to be designed, what assumptions were made, what constraints were placed on your design.

\_\_\_\_Excellent \_\_\_\_\_Good \_\_\_\_\_Average \_\_\_\_\_Fair \_\_\_\_\_None

**C. Problem Decomposition** \_\_\_\_**/4**

Should describe the overall design in terms of functional blocks needed to meet the specifications. Should describe what each functional block is intended to do. There should be a block diagram that includes inputs and outputs of each functional block.

\_\_\_\_Excellent \_\_\_\_\_Good \_\_\_\_\_Average \_\_\_\_\_Fair \_\_\_\_\_None

**D. Detailed Design** **and Module level Testing (Design for Testing)** \_\_\_\_**/16**

* **(5 points)** Should describe how the design works in sufficient detail that a knowledgeable person is able to understand and verify that the design is technically sound.
* **(6 points)** There should be documentation sufficient to allow a knowledgeable person to duplicate your design. You should have schematics for all hardware and source code for all software, programmable logic, VHDL, etc. Source code is **not** needed for the “obvious” parts; you should have pseudo or actual code for the significant parts of your source code. [Include a CD, if needed].
* **(5 points)** Should describe how the design was tested at the module level and corrections made.

\_\_\_\_Excellent \_\_\_\_\_Good \_\_\_\_\_Average \_\_\_\_\_Fair \_\_\_\_\_None

**E. Alternative Designs** \_\_\_\_**/4**

Should describe alternative design approaches to meeting the specifications. You should give the rationale for selecting the design approach you used in favor of others.

\_\_\_\_Excellent \_\_\_\_\_Good \_\_\_\_\_Average \_\_\_\_\_Fair \_\_\_\_\_None

**F. Specification Testing \_\_\_\_/4**

Should describe how the completed design was tested to verify, it meets specifications or to determine deviations from the specs.

\_\_\_\_Excellent \_\_\_\_\_Good \_\_\_\_\_Average \_\_\_\_\_Fair \_\_\_\_\_None

**G. Results and Analysis \_\_\_\_/4**

Should describe how well your design met the specifications and discuss any deviations from those specifications. Give an analysis of any deviations from the specification and **explain** what should be done to correct these.

\_\_\_\_Excellent \_\_\_\_\_Good \_\_\_\_\_Average \_\_\_\_\_Fair \_\_\_\_\_None

**II. Report form. (10 points) The report should be produced in a professional manner. Basis of Evaluation:**

**2 - no deficiencies, 1- no more than 2 minor deficiencies, 0-1 major or multiple minor deficiencies**

|  |  |
| --- | --- |
| The report is well organized, with explanatory text for each section. | **\_\_\_/2** |
| The written language is clear and grammatically correct. | **\_\_\_/2** |
| All figures and tables are referenced in the text and appear shortly after the reference, or are located in an appendix with appropriate text to refer the reader there. All figures and tables have a caption | **\_\_\_/2** |
| All diagrams and schematics are drawn using CAD type software with no hand drawing or lettering | **\_\_\_/2** |
| The report includes a list of references used in completing the design and documenting your results | **\_\_\_/2** |

**Synchronous Digital System Using Xilinx MicroBlaze Processor**

**Author: James Bruska**

**Teammates: Daniel Heck and Matthew Michaels**

**Instructor: Dr. Abul Khondker**

**April 28, 2017**

**Executive Summary**

This project aimed to create a system in which a Xilinx’s NexysTM 4 board was able to display an output sequence using a Xilinx MicroBlaze processor. The system would display a given sequence on four LCDs and change every second. There were also three buttons: one toggle sequence direction (forward or backward); one paused the sequence; one reset the system. The system was created and tested in Vivado 2016.3. The code was written using VHDL. The system functions according to the provided design criteria.

**Design Problem Statement**

This system displays a, ROM initialized, user changeable output sequence from SRAM. A 19-key keypad provides input to control the system state and hexadecimal input for address and data values. An LCD and seven-segment display present the address and data of the SRAM. The LCD and an LED displays state information. The system can exist in two states: operation or programming. Operation mode allows the user to display the sequence values in the SRAM by going through the addresses incrementally. This sequence can be paused and viewed in the forward and backward direction. In programming mode, the user can change the SRAM values. The new value and the address it will be located at are presented on the LCD and seven-segment display.

The specific design criteria:

Hardware Usage:

* Design for Altera DE2 board
* ROM IP core must be used to store initialized data
* LCD must be implemented with the correct pins
* Seven-segment display must be implemented with the correct pins
* SRAM controller must use the onboard SRAM and be implemented with the correct pins
* LEDG0 must be used to display operation and programming state
* The keypad must be connected to the DE2 with a breadboard, 5 5kΩ resistors, and a 40 pin ribbon cable.

Coding limitations:

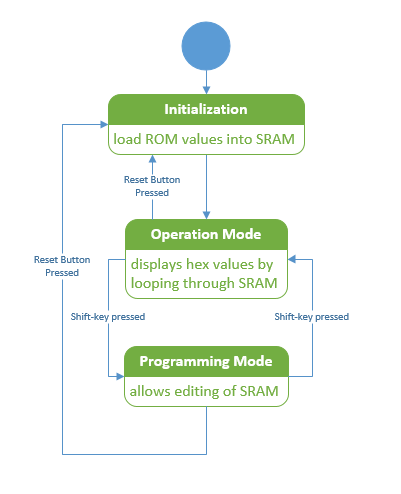
* Implement using Altera’s Quartus II program
* No IP cores can be used except for the ROM
* Debug the design using SignalTap II Logic Analyzer
* Number of states in design should be as low as possible

System Operation Specifications:

* System starts with a power on reset
* All resets load the default sequence from the 256x16 bit ROM into the SRAM
* Initialized output data sequence defined in the “sine.mif” file.
* After initialization, the system goes to operation mode and displays the contents from address 0x00 and does not count
* The Shift key is used to toggle between operation mode and programming mode
* LEDG0 will be on when in operation mode and off when in programming mode
* Operation Mode
  + - A counter cycles through the addresses of the SRAM and displays the data
    - Counter changes every one second
    - H key toggles the counter to start and stop
    - L key toggles the forward and backward direction of the counter
    - The seven-segment displays will show the SRAM address on HEX5 to HEX4 and the SRAM data on HEX3 to HEX0
* Programming Mode
  + - H key toggles whether the data or address of the new SRAM input is edited
    - L key loads the new SRAM data into memory at the new SRAM address
    - The seven-segment displays will show the new SRAM address on HEX5 to HEX4 and the new SRAM data on HEX3 to HEX0
    - When editing the address, if a key on the keypad is pressed then HEX4 will display its HEX value. If a second key is pressed, then the digit at HEX4 will move to HEX5 and the new HEX value will be displayed at HEX4
    - When editing the data, if a key on the keypad is pressed then HEX0 will display its HEX value. If a second key is pressed, then the digit at HEX0 will move to HEX1 and the new HEX value will be displayed at HEX0. If a third and fourth key are pressed then the numbers will shift to the left (HEX2 and HEX3) until HEX3 through HEX0 display a 16-bit binary number
* LCD displays the output of the counter and the mode (operation or programming)

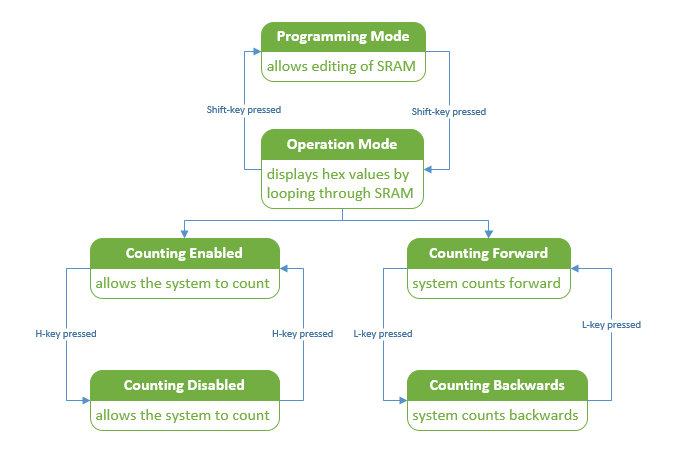
**Problem Decomposition**

Figure 1 displays the overall operational state machine. This is a significant simplification of the operation of the system.

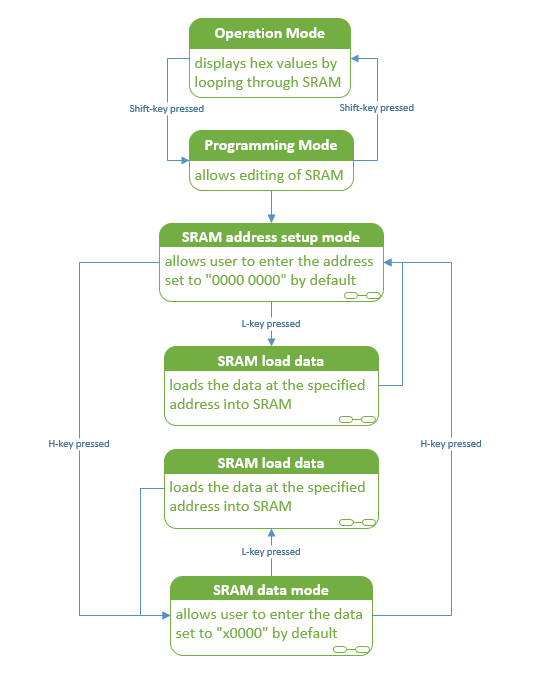


*Figure 1 – Overall Operational State Machine*

Figure 2 and Figure 3 show the details that are present within the operation and programming mode while providing a layer of abstraction in regards to the implementation of each action. These operational state machines do not provide all information about the system outputs or displays and do not include resets. The reset will bring the system to the initialization state, which is not shown in these diagrams.

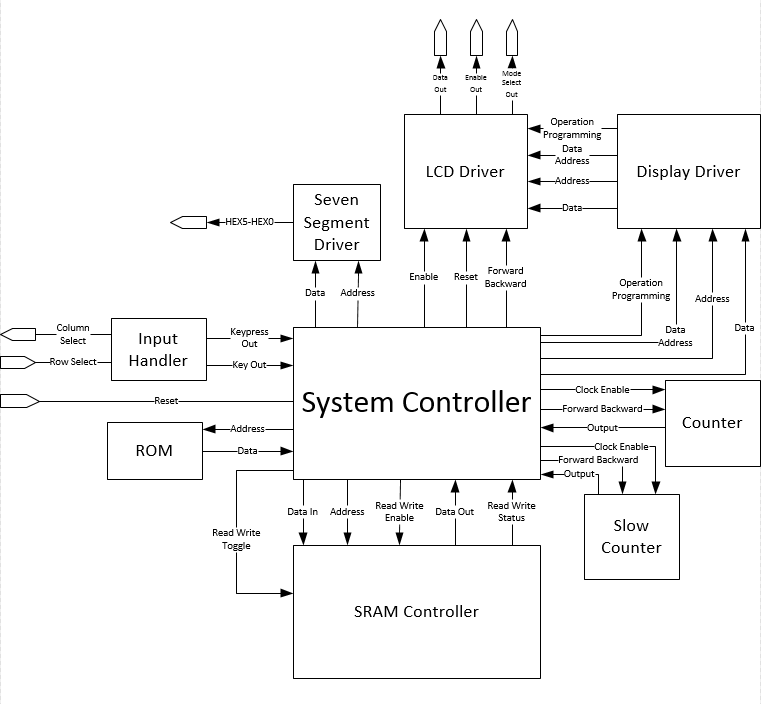


*Figure 2 – Operation Mode Operational State Machine*



*Figure 3 – Programming Mode Operational State Machine*

Figure 4 shows an overview of the main components required to implement the described system.



*Figure 4 – System Controller (Main System Components)*

All components shown use the main 50 MHz clock except for the seven-segment driver. This driver uses no clock.

“Column Select” is the 4-bit signal sent to the keypad to supply specific voltages such

that the “row select” will provide information about a keypress. This functions based on a ring counter where everything is ‘1’ except for a single ‘0.’

“Row Select” is the 5-bit signal received from the keypad that allows the system to determine when keypress is occurring as well as which key it is.

“Reset” is a button supplied reset signal. This will synchronously reset the system including the SRAM data. It will also put the system back into operation mode and stop the system from counting until the user allows it.

“HEX5-HEX0” are six 4-bit signals that describe the values that are to be output to the rightmost six seven-segment displays.

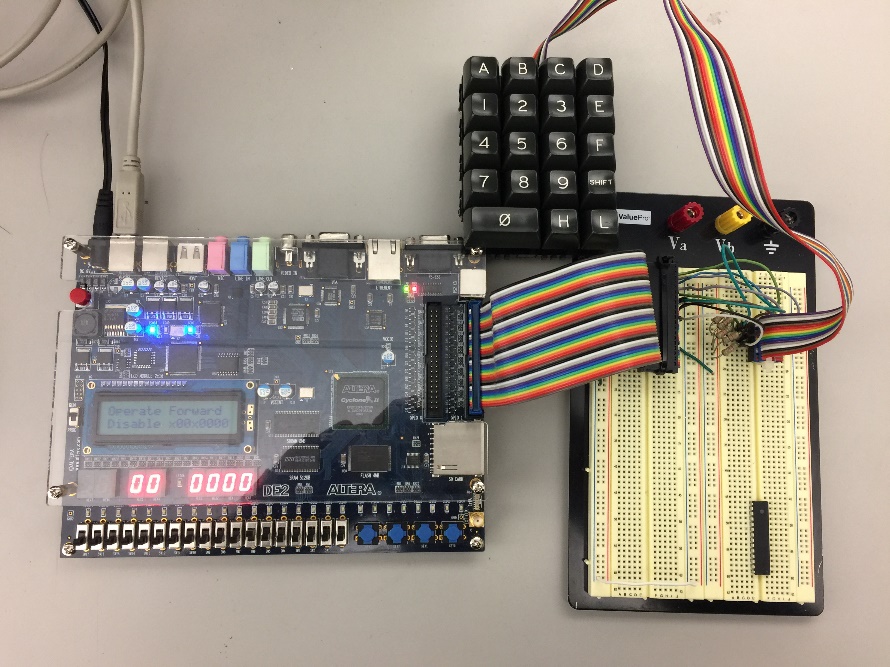
“Data Out” is the 7-bit output signal of data to display to the LCD.

“Enable Out” is a signal that controls the LCD enabled and disabled state.

“Mode Select Out” is the signal that informs the LCD whether to display a character (‘1’) or perform an operation (‘0’).

**Detailed Design**

This is the design of the system seen below in Figure 5.



*Figure 5 – Physical Picture of the System*

Input Handler

Input Handler acts as a ring counter. It sends out a 4-bit signal called “Column Select” that is composed of all ‘1’s except for a single ‘0.’ This ‘0’ rotates around each Column. When a button is pushed and the ring counter has ‘0’ on the correct column, the entire row becomes a ‘0.’ In other words the line becomes grounded. This row position is then saved to the internal logic by a ‘0’ within the 5-bit “Row Select” signal. Thus, with the row and column of the keypress, the correct key can be determined with a simple look up table.

ROM

The ROM is created using the IP core provided. It takes in the 50 MHz clock and an Address and return the data at the address on the clock cycle. This data is preloaded by the “sine.mif” file that was imported into the project. This can be found in Appendix D.

Counters

The Counter and Slow Counter are counters that loop at a certain value. They start at 0x00 and count up by one at every clock cycle until they reach 0xFF. At this point, they go to 0x00 and repeat. They also accept two inputs: “Clock Enable” and “Forward Backward.” The “Clock Enable” signal halts the counting within the system if it is set to ‘0.’ The “Forward Backward” signal informs the counter whether to count up to 0xFF or count down to 0x00. If it is counting down and it reaches 0x00 then it start at 0xFF and continues. The only difference between Counter and Slow Counter is the Counter counts to 5,000 while Slow Counter counts to 50,000,000. This allows Counter to generate a 10 kHz cycle while the Slow Counter generates a 1 Hz cycle.

Display Driver

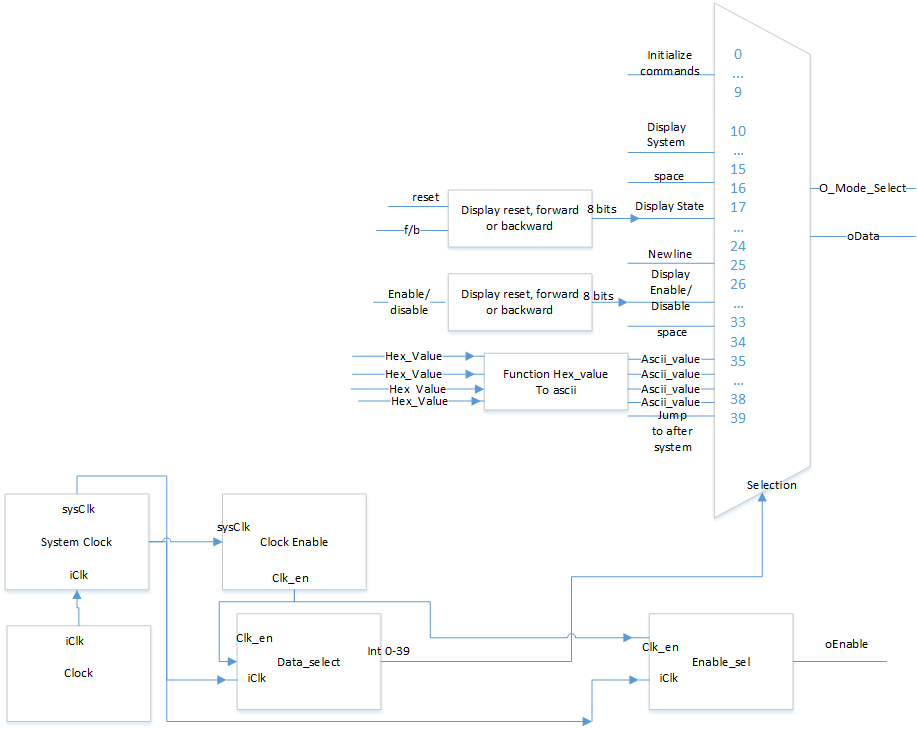
The Display Driver is a very basic controller that assigns what outputs the LCD and seven-segment display receive. It takes in an 8-bit keycode, the SRAM address, the SRAM data, the single bit to indicate if the system is writing to data or address, and the single bit to indicate if the system is in programming or operation mode. It then outputs the correct address and data for the LCD and seven-segment display. This driver implements the shift registers from the single input data/address hex to generate the full output data/address value.

Seven-Segment Driver

The Seven-Segment Driver is a simplistic output device. The driver has two inputs: a 16-bit data and an 8-bit address. There are six 8-bit outputs that go directly to the board pins: hex5, hex4, hex3, hex2, hex1, and hex0. The address displays on hex5 and hex4 while the data is display on hex3-hex0. The driver also converts the hexadecimal values to seven-segment display value via a lookup table function called “hex\_to\_seven.” With this design, the seven-segment display will receive the correct values to display the data and address as specified.

LCD Driver

The LCD Driver is composed of one main state machine used to initialize and write output to the LCD. The main state machine also uses different multiplexers to control the different display options that vary with respect to the internal state. The three modes of operation that the system can be in are: reset, forward, and backwards. The inputs are as follows: input clock, active high reset, forward backwards toggle (‘1’ is forward), active high system enabler, and the current 4-digit hexadecimal value to be displayed. The outputs are as follows: the 8-bit value to be sent to the LCD to display, an active high LCD enabler, and a single bit command output control (‘0’ is a command). This can be seen below in Figure 6.



*Figure 6 – LCD Driver Circuit Diagram*

SRAM Controller

The SRAM Controller is a synchronous controller than allows for easy interfacing with the SRAM on the board. It take in an 18-bit address, a 16-bit data input, a read write bit, and a read write enable bit. The address is the address of the SRAM that will be read from or written to. The data input is the data that will be written to the SRAM at the address specified by the address input. In order to remove the fight condition between data input and data output, the data in value is set to high impedance for single data bus to the SRAM when in reading mode. This problem occurs due to the single data in/out line on the SRAM fighting for control. The read write bit informs the controller whether it’s supposed to read or write from the SRAM. A rising edge for the read write enable will initiate a read or write from the SRAM.

The SRAM controller then returns two signals: the data output from the SRAM at the address and a read write status bit. The read write status bit indicates when the SRAM is finished with the last read or write sequence. It will be ‘0’ when the SRAM is free to read or write and ‘1’ when it is in the middle of another operation. If this bit is a ‘1’ then a rising edge of the read write enable will not cause the SRAM to start another operation.

It also outputs six signals to the SRAM (the six signals needed for the SRAM to function properly). The CE\_N, UB\_N, and LB\_N are always set to ‘0’ for convenience. For this project they do not need to deviate from this value at any point. The circuit diagram can be seen in Figure 7 and the state machine table can be seen in Figure 8.

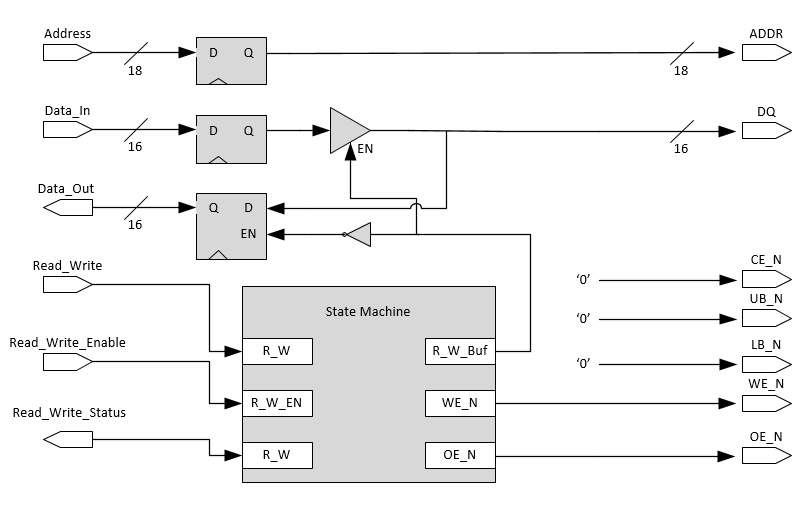


Figure 7 *– SRAM Controller Circuit Diagram*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **State 0**  RW Status = ‘0’  RW Buffer = RW | | **State 1**  RW Status = ‘1’  RW Buffer = RW Buffer | | | **State 2**  RW Status = ‘1’  RW Buffer = RW Buffer | |
| **Read Mode**  OE\_N = ‘1’  WE\_N = ‘1’ | **Write Mode**  WE\_N = ‘1’  OE\_N = ‘1’ | **Read Mode**  OE\_N = ‘0’  WE\_N = ‘1’ | **Write** **Mode**  OE\_N = ‘0’  OE\_N = ‘1’ | **Read Mode**  OE\_N = ‘0’  WE\_N = ‘1’ | | **Write Mode**  OE\_N = ‘0’  OE\_N = ‘1’ |

*Figure 8 – SRAM Controller State Table*

System Controller

The system controller is one state machine that connects and controls the interactions of rest of the components. As a result, the system controller will be described below with a state machine.

In order to provide a concise description, the state machine is described with a table instead of a state diagram. This is because the state machine was either too large, clustered with all of the inputs tangled, or provided the same information with a list inside of a state while providing little more understanding. The states of the state machine are described in Figure 9.

|  |  |
| --- | --- |
| **Initialization Mode** | |
| **Counter Out = 0xFF**  Forward Backward = ‘0’  Counter Clock Enable = ‘0’  Address or Data = ‘0’  Counter Reset = ‘0’  Go to Operation Mode | **Counter Out = 0xFF**  Buffer Forward Backward = ‘0’  Invert Counter Clock Enable  Address or Data = ‘0’  SRAM Read Write = ‘1’  SRAM Address = Counter Out  ROM Address = Counter Out  SRAM Data In = ROM Out  Counter Reset = ‘1’ |

|  |  |
| --- | --- |
| **Operation Mode**  Operation or Programming Mode = ‘0’  SRAM Read Write = ‘0’  Invert SRAM Read Write Enable  SRAM Address = Counter Out  Counter Reset = ‘0’ | |
| **L key is Pressed**  Invert Forward Backward | **H key is Pressed**  Invert Counter Clock Enable |
| **Shift key is Pressed**  SRAM Read Write Enable = ‘0’  Go to Programming Mode | **Reset key is Pressed**  Counter Reset = ‘1’  Go to Initialization Mode |

|  |  |
| --- | --- |
| **Programming Mode**  Operation or Programming Mode = ‘1’  Counter Clock Enable = ‘0’  SRAM Read Write = ‘1’  SRAM Address = Keypad Address  SRAM Data In = Keypad Data | |
| **L key is Pressed**  SRAM Read Write Enable = ‘1’ | **H key is Pressed**  Invert Address or Data |
| **Shift key is Pressed**  Counter Reset = ‘1’  Go to Operation Mode | **Reset key is Pressed**  Counter Reset = ‘1’  Go to Initialization Mode |
| **Otherwise**  SRAM Read Write Enable = ‘0’ |  |

*Figure 9 – System Controller State Table*

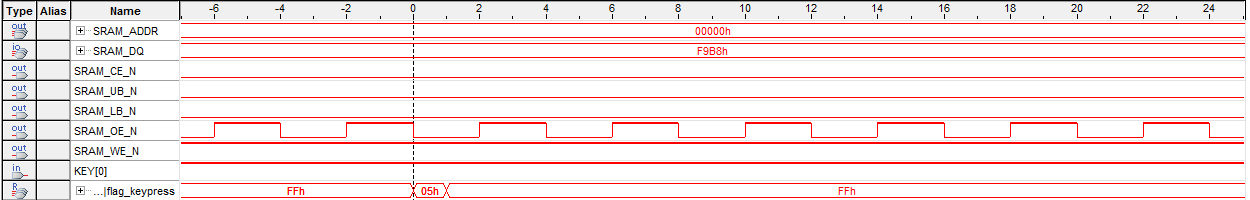
With all of these parts, the system should be capable of functioning if the parts function correctly. Proof of their correctness is shown in the next section.

**Module and Specifications Testing**

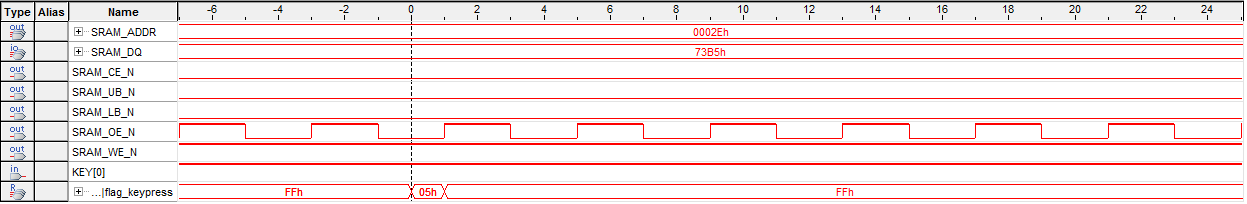
1

Each component of the system was tested using a logic analyzer. The results of each method are shown below. The signals shown area all related to SRAM for the input because this demonstrates that the same values were being passed to the SRAM controller as well. In the state machine, the values were directly passed through so these graphs still correctly demonstrate functionality of the component.

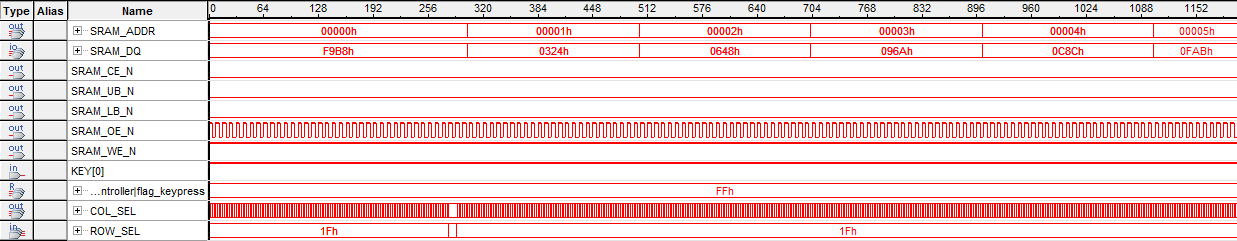
The logic analyzer has a limit on the amount of data it can hold. In order to see a significant range of the outputs, it was necessary to trigger on button press. This allowed for the faster timing, a 50 MHz clock, to display the needed information. In order to show proof of the slower system operations, a slower 200 Hz clock was also used. The results of the SignalTap II logic analyzer can be seen in Figures 10-21.



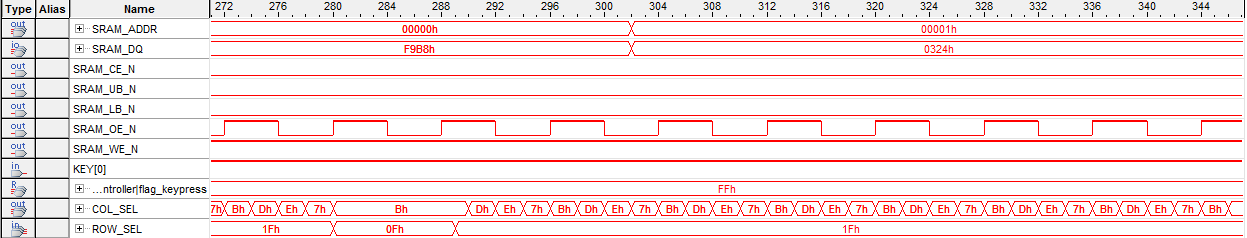
*Figure 10 – 50 MHz Clock. Shows the reset that occurs at the beginning and all of the initial states for the SRAM. It also displays the reset bug that occurs on a rare occasion (SRAM\_DQ is at 0xF9B8 and not 0x0000).*



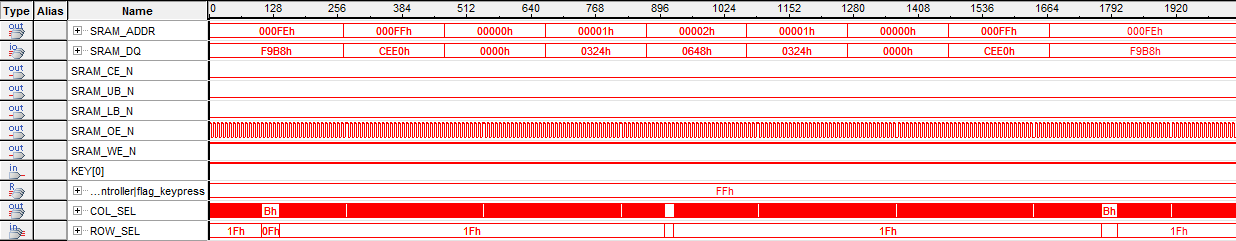
*Figure 11 – 50 MHz Clock. Shows a value later on in the sequence to show that the SRAM is using the correct data.*



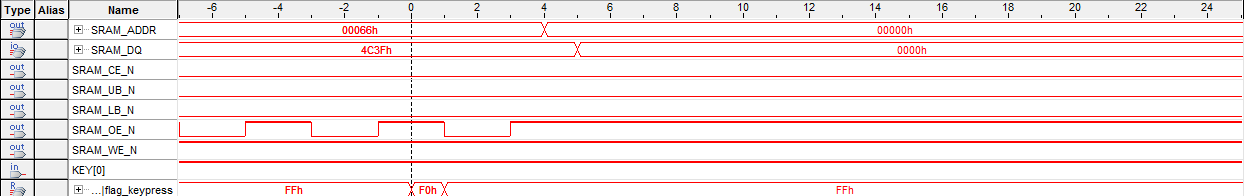
*Figure 12 – 200 Hz Clock. Shows the counter counting through a few states.*



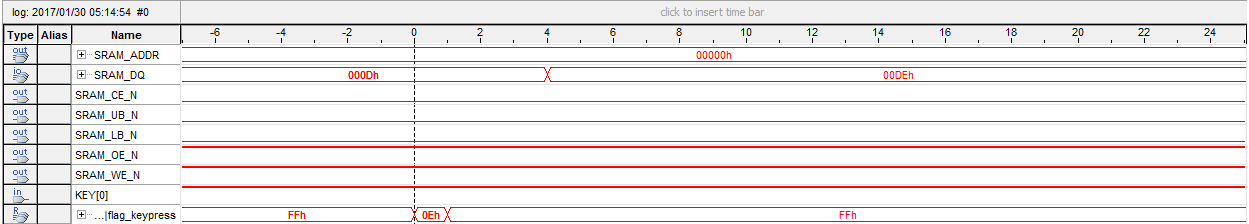
*Figure 13- 200 Hz Clock. Shows how the Row Select and Column Select function.*



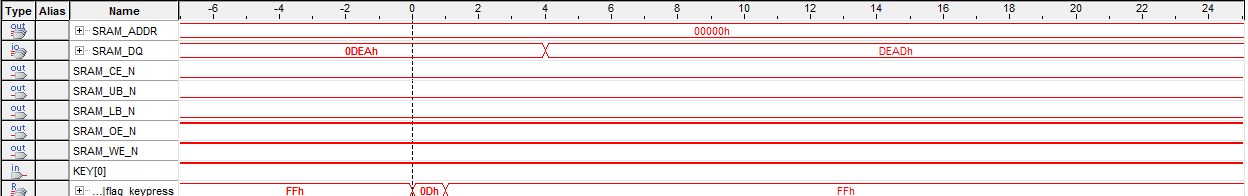
*Figure 14 – 200 Hz Clock. Shows the counter correctly going from address 0xFE to 0x01 and then reversing the counter, with an L-Key click, and going from 0x01 to 0xFE correctly.*



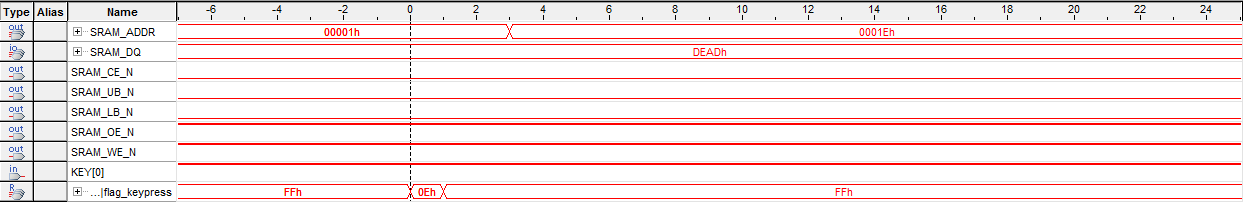
*Figure 15 – 50 MHz Clock. Shows the entering of Program mode correctly with output enable staying high, the new address as 0x00, and the new data as 0x0000.*



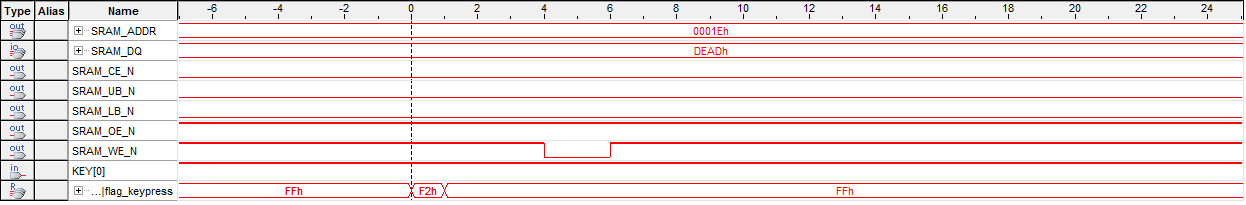
*Figure 16 – 50 MHz Clock. Shows that the shifting worked for the first two bits of the data field.*



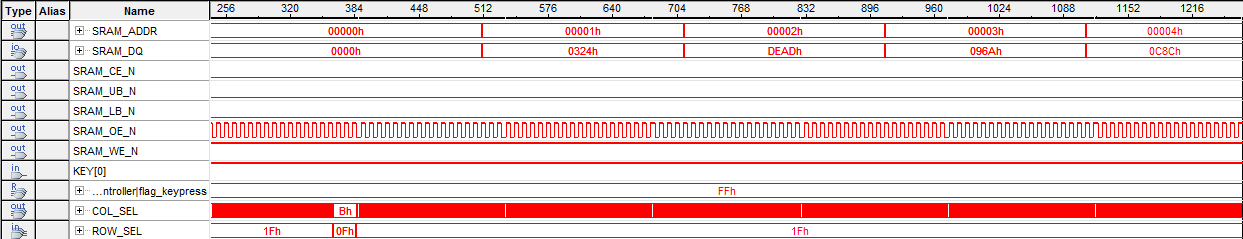
*Figure 17 – 50 MHz Clock. Shows that the shifting worked for all 4 bits of the data field.*



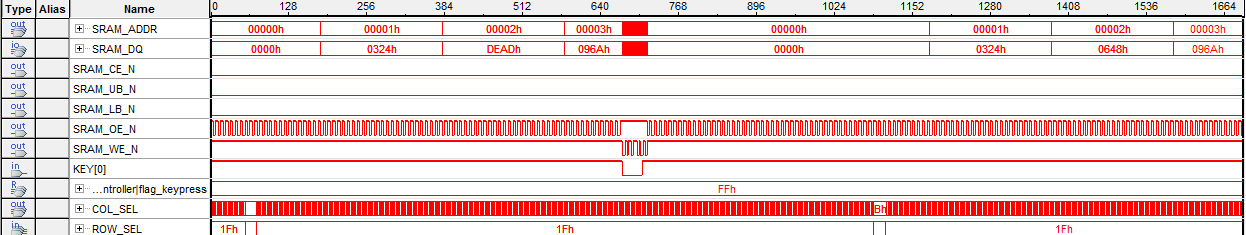
*Figure 18 – 50 MHz Clock. Shows that the shifting worked for all 2 bits of the address field.*



*Figure 19 – 50 MHz Clock. Shows that the load triggered the writing to the SRAM with the timing needed for a successful write.*



*Figure 20 –200 Hz Clock. Shows the new value of 0xDEAD at address 0x02 in the sequence.*



*Figure 21 – 200 Hz Clock. Shows the reset and how the value at 0x1E is back to 0xFF5F.*

It is important to note that the LCD Driver and Seven-Segment Driver do not show testing here due to their thorough testing during creation. For more information please see Zander Blasingame’s final project report for EE365.

After the module testing, the system was tested as a whole. This was performed by carefully trying every state seen in Figures 1-3 as well as the reset key. Every path between each of these states was also tested to work correctly. The list of specifications from the system operation specifications in the design problem statement were then checked as well. This showed that the system functioned correctly as a whole in addition to each component functioning correctly. The only deviation was the reset bug that would occur non-deterministically.

**Alternative Designs**

There were many things changed over the course of the design process. Listed below are the two of the most significant changes. The system started as the one described by Prof. Khondker in class on January 16. The first major change was that one massive system controller would control the system. The previous design also allowed for a smaller system controller that could pass state information around which could then be succinctly linked in the top level. This had merit, but caused two problems that the larger design solved: it was delegating problems to a component who should not have the job and it made each internal component account for the state of the system. The first problem stemmed from the idea that the top level should only be used to connect the signals to the board pins. This makes for a clean design. By creating a smaller system controller, the top level would have to take some of the responsibility of connecting the parts. The other problem is that the smaller component would have to account for the state of the system, such as the SRAM controller remembering if it was in read or write mode. This ruined the modularity of each component. It also would make it significantly harder to move the components to a new project. As a result, larger system controller was used.

The other change was made due to a modularity issue. At first the seven-segment display driver and the LCD driver were part of the same component. This was convenient because the information that needed to be generated for output could be controlled within one component. Unfortunately, this made the design rigid and unadaptable. Thus, a new system was created in which the seven-segment driver and LCD driver were separate, but controlled but a display driver. This display driver handles the generation of the outputs. These values are then fed to the seven-segment and LCD drivers. This allows for modularity without the wastefulness of determining the output information for each circuit component.

**Results and Analysis**

This design followed all of the instructions set forth within the design constraints. This can be seen by the Module and Specifications Testing. There exists an error will occasionally occur on reset. In these rare cases, it will change address 0x00 to 0xF9B8, which is the value of address 0xFE. This cause of this problem is currently unknown. It could probably be solved with a hold on counter after reset. This hold would only release after the reset button has been released to provide a way to ensure that the counter had stopped after the initial reset loop. The top-level pin control is provided in Appendix A. The VHDL code for the system controller as specified in figure 1 is in appendix B. Appendix C contains the changed pin assignments using Pin Planner in Quartus II. Appendix D contains the ROM memory information. The rest of the code is in the following appendices.

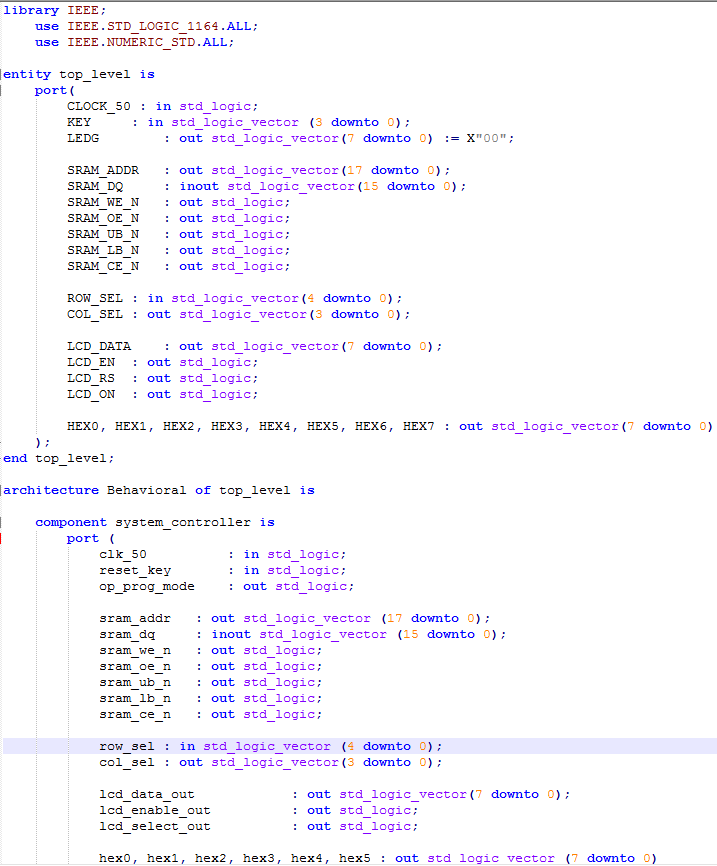
**References**

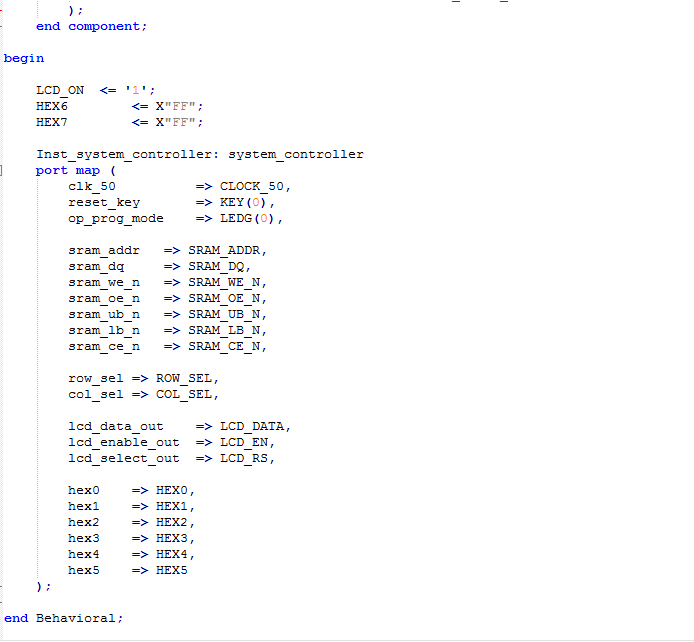
"DE2 User Manual." Altera., 2003. Web. <ftp://ftp.altera.com/up/pub/Webdocs/DE2\_UserManual.pdf>.

A. Khondker, “SRAM Controller (A design guide)”. EE316, 2016. Web. https://moodle.clarkson.edu/pluginfile.php/483826/course/section/97348/DE2\_schematics.pdf

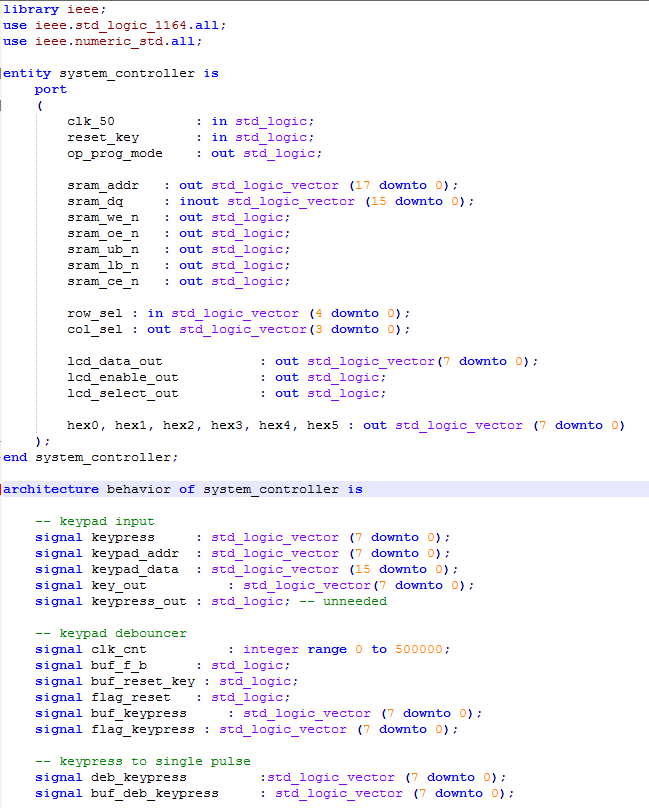
**Appendices**

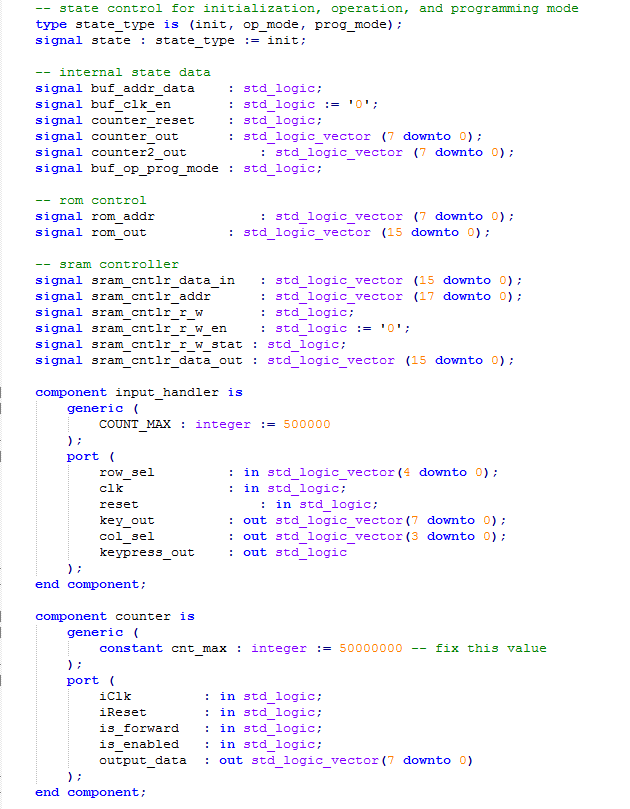
1. top\_level.vhd

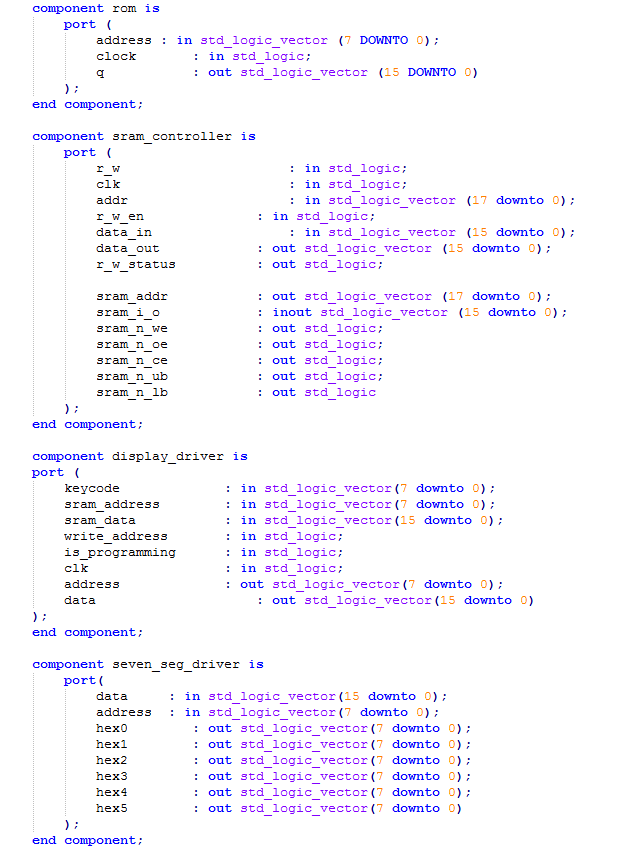


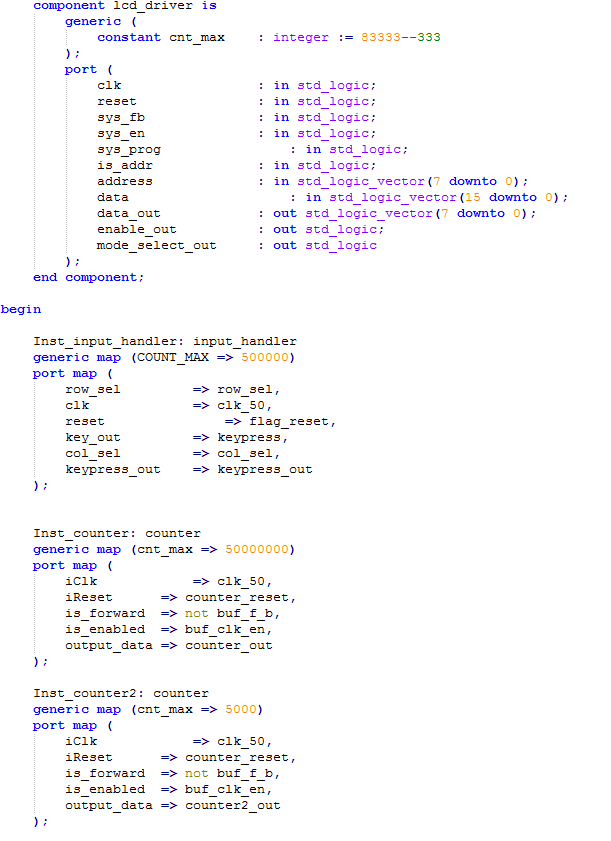


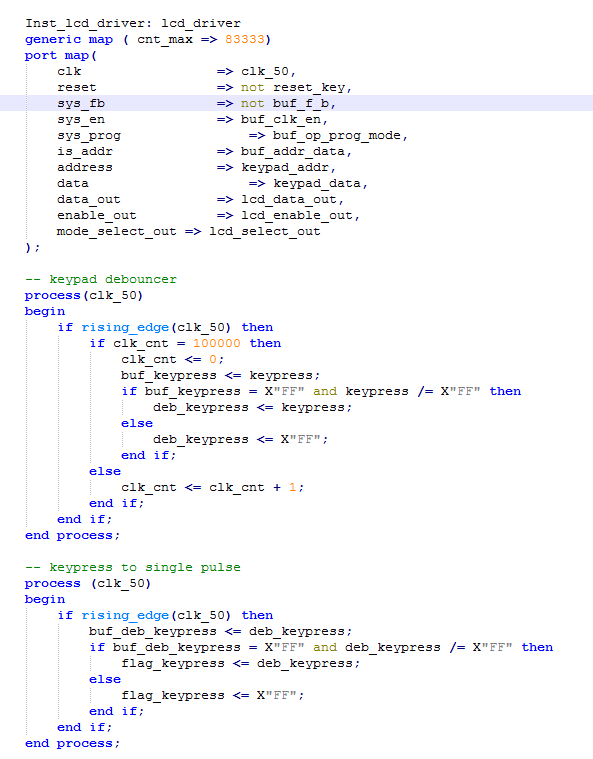
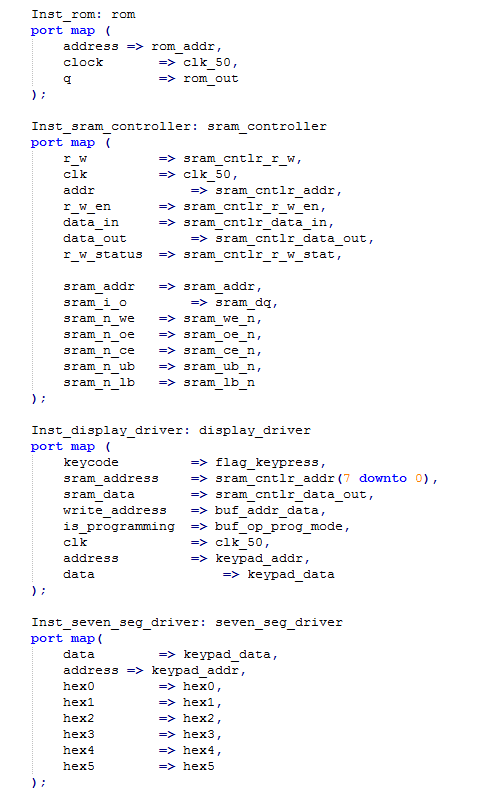
1. system\_controller.vhd

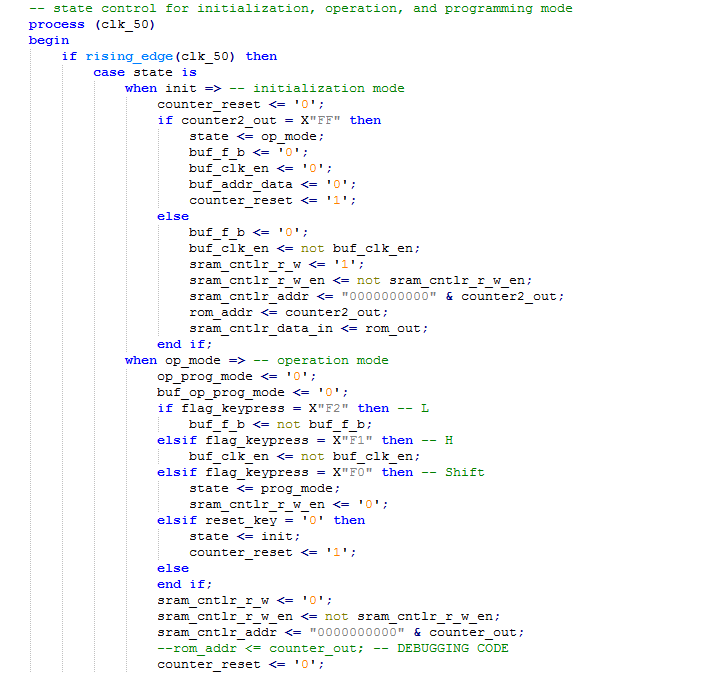


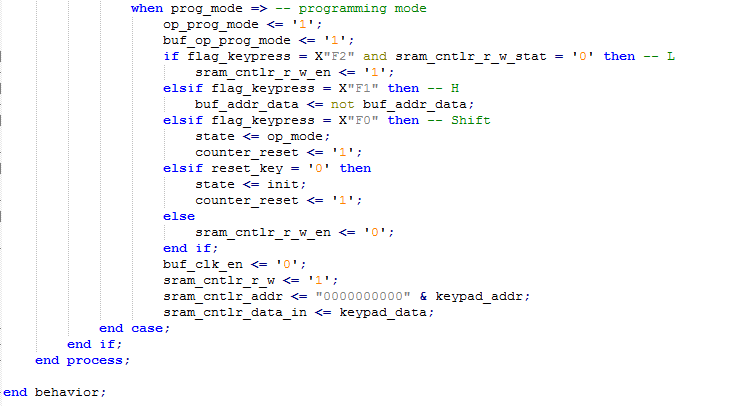




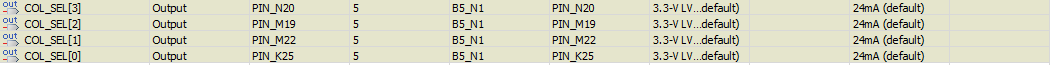


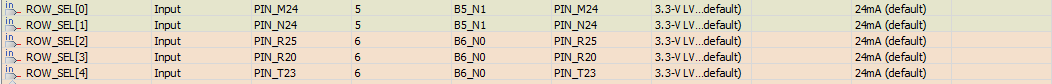






1. Changed Pinouts from the Default File





1. ROM Initialization Data

-- Memory Initialization File

-- Generated by .MIF File Generator Utility v1.2

-- by Rune Baeverrud

-- Angle Range 0-360 degrees (quadrant 1-2-3-4)

-- Function type : Sine

-- Options : Normal

-- Peak Amplitude : 32767.00

-- Offset : 0

-- Number of Samples: 256

-- SNR: : SNR: 97.95dB over i = 0 to 255

DEPTH = 256;

WIDTH = 16;

ADDRESS\_RADIX = DEC;

DATA\_RADIX = DEC;

CONTENT

BEGIN

0 : 0;

1 : 804;

2 : 1608;

3 : 2410;

4 : 3212;

5 : 4011;

6 : 4808;

7 : 5602;

8 : 6393;

9 : 7179;

10 : 7962;

11 : 8739;

12 : 9512;

13 : 10278;

14 : 11039;

15 : 11793;

16 : 12539;

17 : 13279;

18 : 14010;

19 : 14732;

20 : 15446;

21 : 16151;

22 : 16846;

23 : 17530;

24 : 18204;

25 : 18868;

26 : 19519;

27 : 20159;

28 : 20787;

29 : 21403;

30 : 22005;

31 : 22594;

32 : 23170;

33 : 23731;

34 : 24279;

35 : 24811;

36 : 25329;

37 : 25832;

38 : 26319;

39 : 26790;

40 : 27245;

41 : 27683;

42 : 28105;

43 : 28510;

44 : 28898;

45 : 29268;

46 : 29621;

47 : 29956;

48 : 30273;

49 : 30571;

50 : 30852;

51 : 31113;

52 : 31356;

53 : 31580;

54 : 31785;

55 : 31971;

56 : 32137;

57 : 32285;

58 : 32412;

59 : 32521;

60 : 32609;

61 : 32678;

62 : 32728;

63 : 32757;

64 : 32767;

65 : 32757;

66 : 32728;

67 : 32678;

68 : 32609;

69 : 32521;

70 : 32412;

71 : 32285;

72 : 32137;

73 : 31971;

74 : 31785;

75 : 31580;

76 : 31356;

77 : 31113;

78 : 30852;

79 : 30571;

80 : 30273;

81 : 29956;

82 : 29621;

83 : 29268;

84 : 28898;

85 : 28510;

86 : 28105;

87 : 27683;

88 : 27245;

89 : 26790;

90 : 26319;

91 : 25832;

92 : 25329;

93 : 24811;

94 : 24279;

95 : 23731;

96 : 23170;

97 : 22594;

98 : 22005;

99 : 21403;

100 : 20787;

101 : 20159;

102 : 19519;

103 : 18868;

104 : 18204;

105 : 17530;

106 : 16846;

107 : 16151;

108 : 15446;

109 : 14732;

110 : 14010;

111 : 13279;

112 : 12539;

113 : 11793;

114 : 11039;

115 : 10278;

116 : 9512;

117 : 8739;

118 : 7962;

119 : 7179;

120 : 6393;

121 : 5602;

122 : 4808;

123 : 4011;

124 : 3212;

125 : 2410;

126 : 1608;

127 : 804;

128 : 0;

129 : 64732;

130 : 63928;

131 : 63126;

132 : 62324;

133 : 61525;

134 : 60728;

135 : 59934;

136 : 59143;

137 : 58357;

138 : 57574;

139 : 56797;

140 : 56024;

141 : 55258;

142 : 54497;

143 : 53743;

144 : 52997;

145 : 52257;

146 : 51526;

147 : 50804;

148 : 50090;

149 : 49385;

150 : 48690;

151 : 48006;

152 : 47332;

153 : 46668;

154 : 46017;

155 : 45377;

156 : 44749;

157 : 44133;

158 : 43531;

159 : 42942;

160 : 42366;

161 : 41805;

162 : 41257;

163 : 40725;

164 : 40207;

165 : 39704;

166 : 39217;

167 : 38746;

168 : 38291;

169 : 37853;

170 : 37431;

171 : 37026;

172 : 36638;

173 : 36268;

174 : 35915;

175 : 35580;

176 : 35263;

177 : 34965;

178 : 34684;

179 : 34423;

180 : 34180;

181 : 33956;

182 : 33751;

183 : 33565;

184 : 33399;

185 : 33251;

186 : 33124;

187 : 33015;

188 : 32927;

189 : 32858;

190 : 32808;

191 : 32779;

192 : 32769;

193 : 32779;

194 : 32808;

195 : 32858;

196 : 32927;

197 : 33015;

198 : 33124;

199 : 33251;

200 : 33399;

201 : 33565;

202 : 33751;

203 : 33956;

204 : 34180;

205 : 34423;

206 : 34684;

207 : 34965;

208 : 35263;

209 : 35580;

210 : 35915;

211 : 36268;

212 : 36638;

213 : 37026;

214 : 37431;

215 : 37853;

216 : 38291;

217 : 38746;

218 : 39217;

219 : 39704;

220 : 40207;

221 : 40725;

222 : 41257;

223 : 41805;

224 : 42366;

225 : 42942;

226 : 43531;

227 : 44133;

228 : 44749;

229 : 45377;

230 : 46017;

231 : 46668;

232 : 47332;

233 : 48006;

234 : 48690;

235 : 49385;

236 : 50090;

237 : 50804;

238 : 51526;

239 : 52257;

240 : 52997;

241 : 53743;

242 : 54497;

243 : 55258;

244 : 56024;

245 : 56797;

246 : 57574;

247 : 58357;

248 : 59143;

249 : 59934;

250 : 60728;

251 : 61525;

252 : 62324;

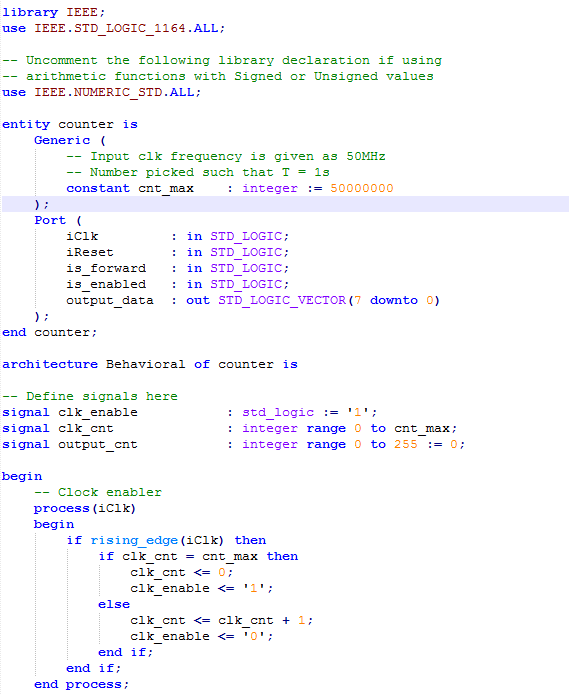
253 : 63126;

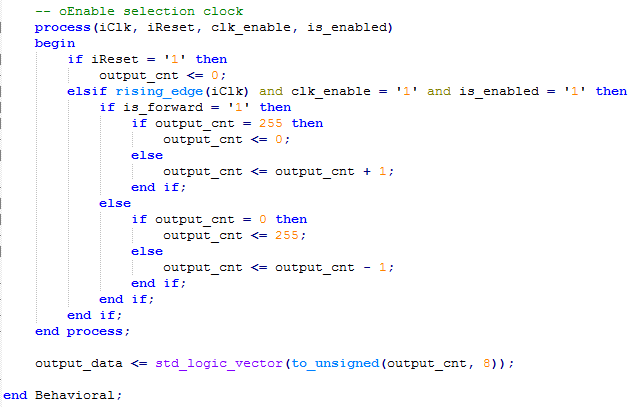
254 : 63928;

255 : 64732;

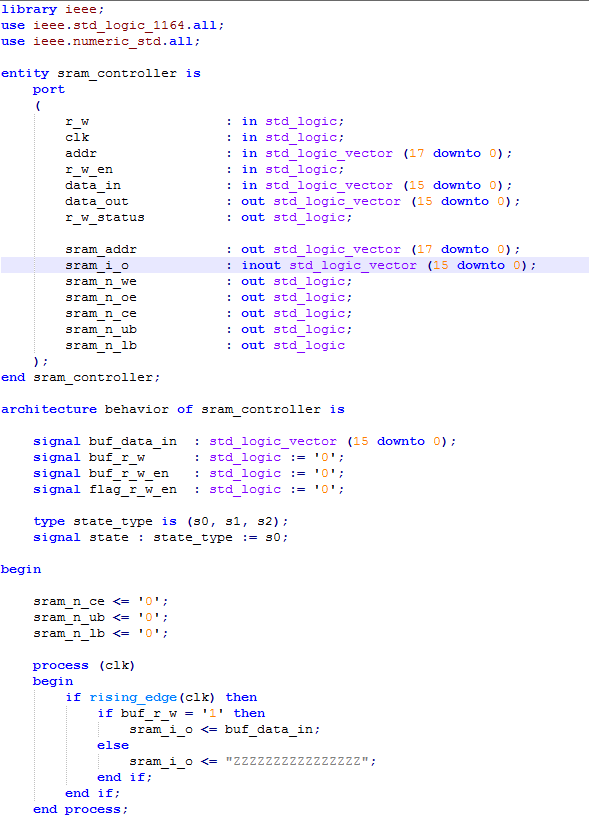
END;

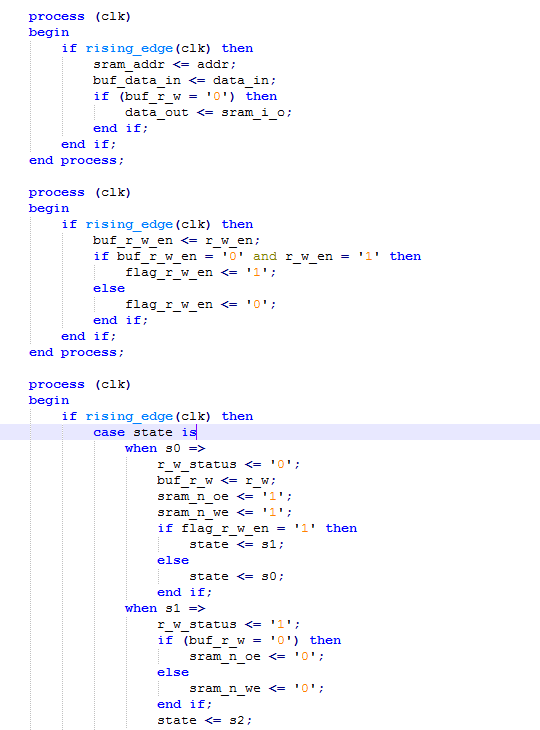
1. counter.vhd

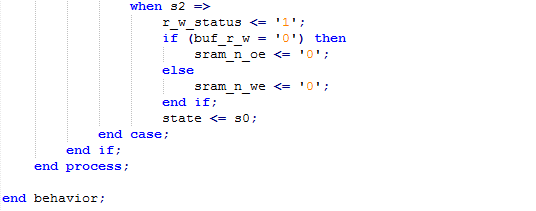




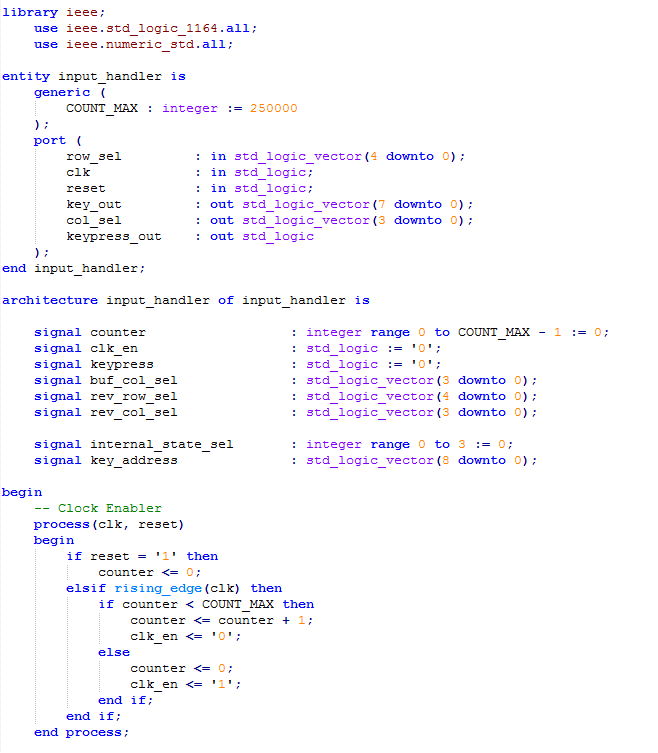
1. sram\_controller

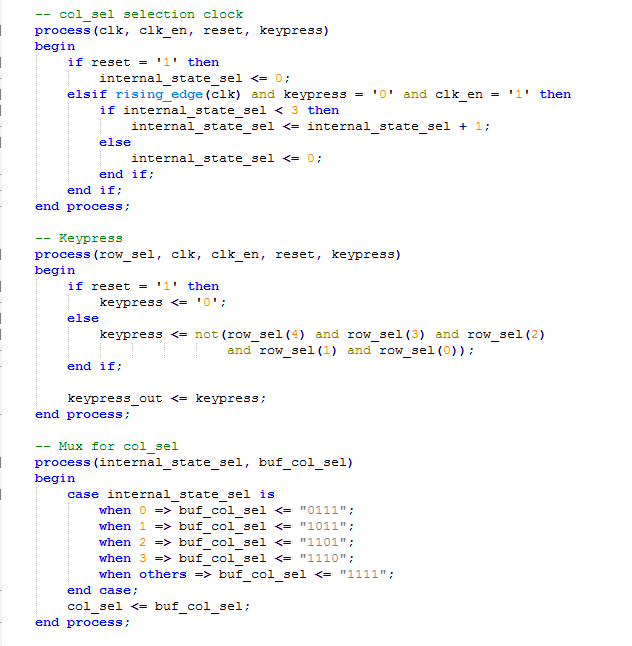


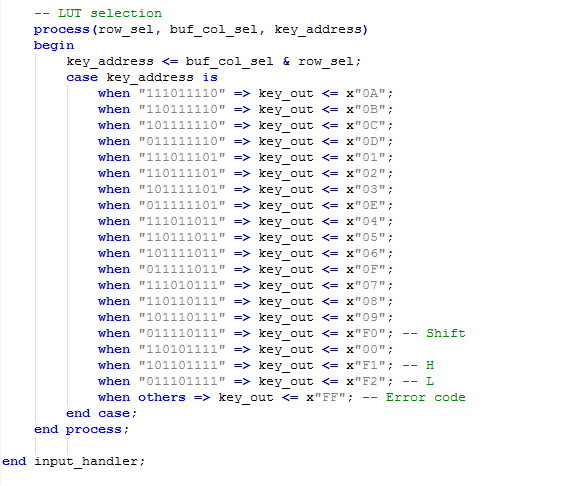




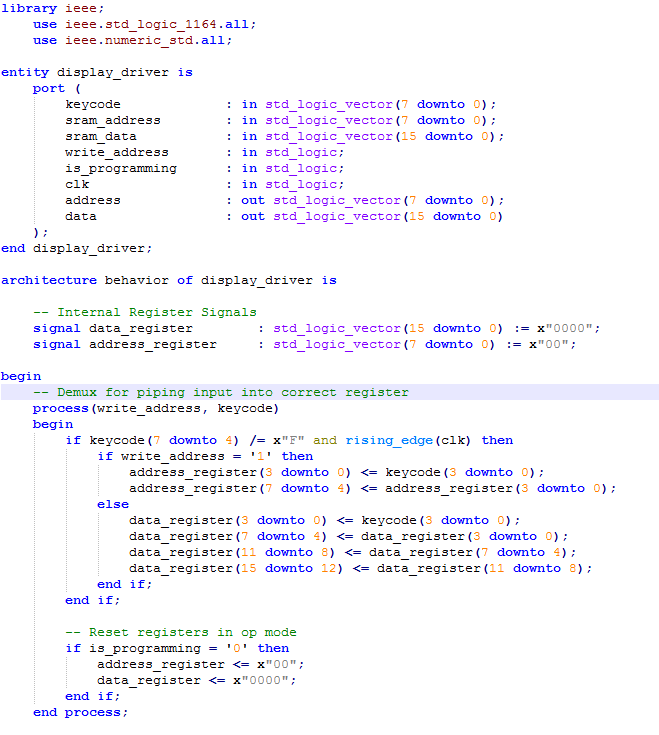
1. input\_handler.vhd

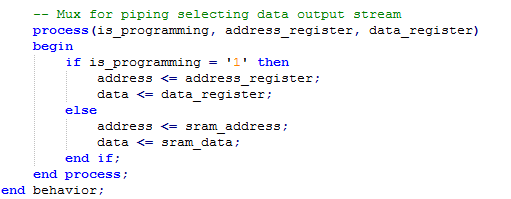




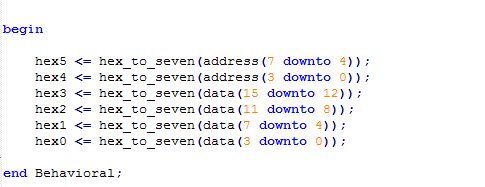
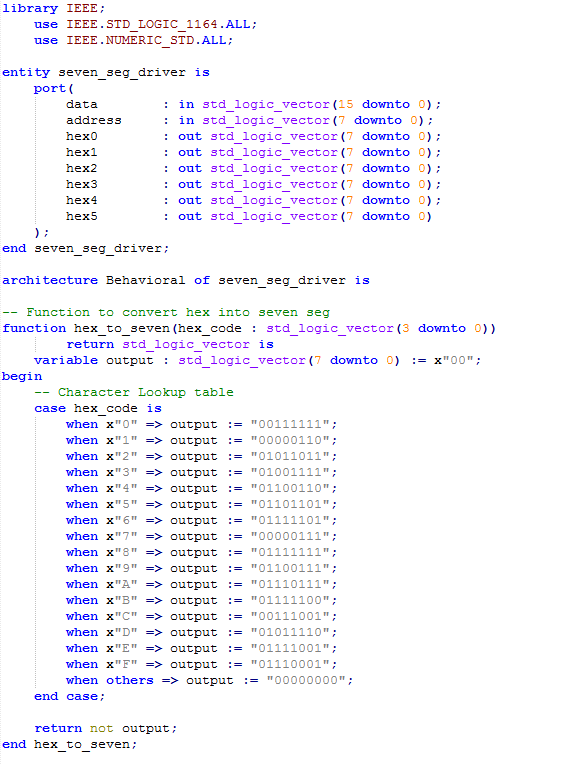


1. display\_driver.vhd





1. seven\_seg\_driver.hvd



1. lcd\_driver.vhd

