

Justin Sim and Mina Gao  
EE 371  
January 12, 2024  
Lab 1 Report

## Procedure

This lab can be segmented into three tasks: create an FSM that models enter/exit behavior of a vehicle garage, create a counting mechanism to keep track of number of enters and exists, and lastly output the count onto a display all on Quartus. Then simulate these approaches on ModelSim and demonstrate them on an FPGA.

### Task #1

First we knew that an FSM had to be implemented in order to represent the sequential logic for a/b sensor data. One important aspect of this Mealy FSM is the adaptability to the case where a vehicle may hit sensor a or b, as to indicate entrance or exit, but then back up from the sensor and leave. If a 4-state FSM had been implemented, this could lead to a faulty output, which is why the 7-state design was necessary, to accommodate for this specific case. I began this by creating the state diagram in Figure 1, along with the Truth table and circuit diagram in Figure 2 and 3.

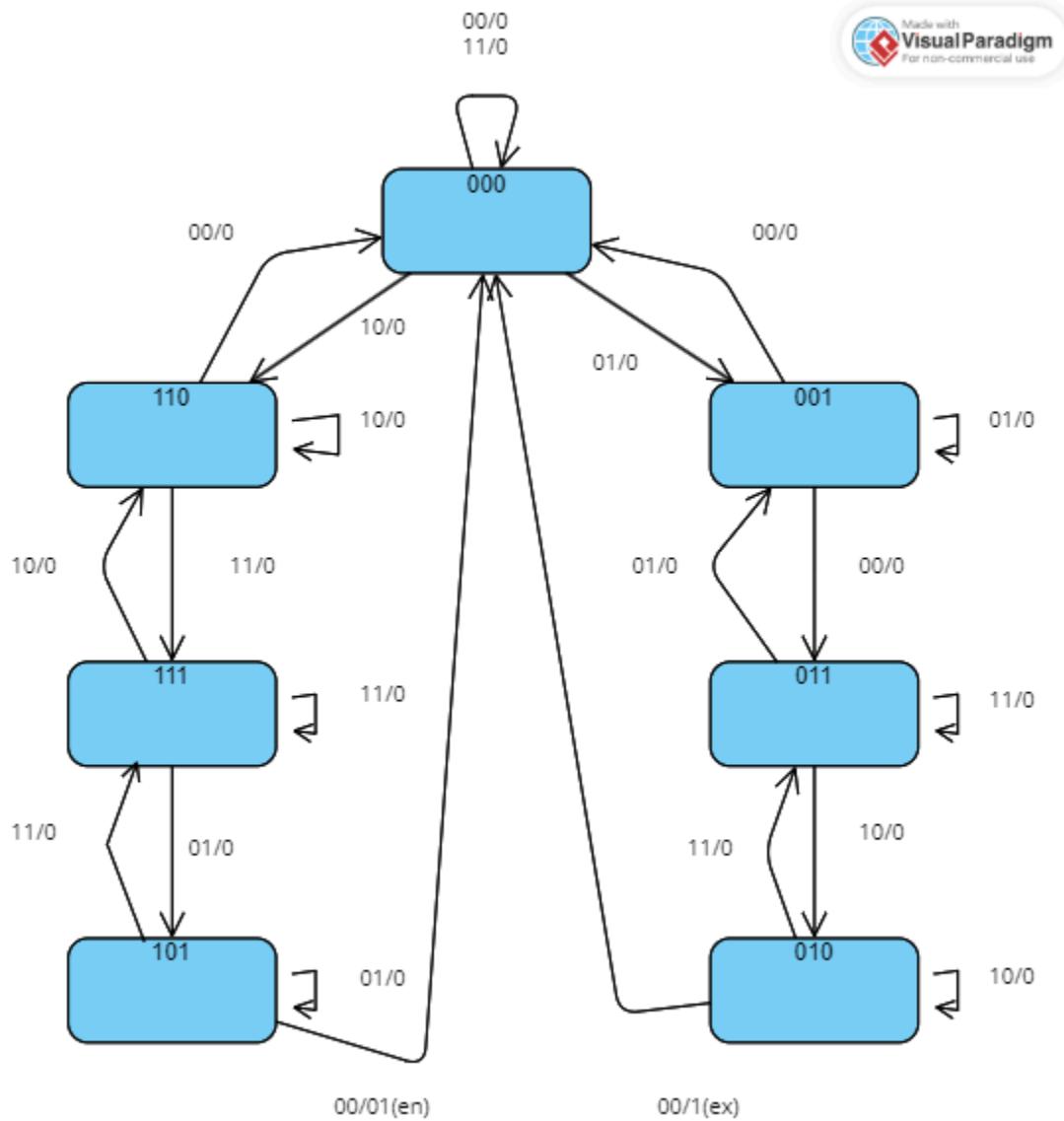
### Task #2

For completing the second task, we are asked to design a counter with two input signals (inc, dec) with a max capacity of 25 (or 5 for the demo purpose). Since the max capacity is 25 (11001 in binary), I decided to use 5 bit binary numbers to store the count. In order to demonstrate that I ended up parameterizing the counter modules which can provide more flexibility / adjustability with the counter design. As shown in the code, the parameter MAX can be assigned to any value. The logic inside of the counter module is that the counter will only update (increasing or decreasing) if its current value is in the range of 0-MAX. When reset, the counter will always start with 0 (00000 in binary). While the value is in the range, the counter will add 1 if inc is true and minus 1 if dec is true.

### Task #3

The display module is designed to take in the value that comes out from the counter module (5 bit binary) and display the value on HEX0-HEX5. HEX takes 7 segment values(7 bit binary), therefore we need to assign different values to HEX0-HEX5 for each possible case that comes

out from the counter modules, which will be 0-25. For displaying 1-24, we only need to activate and assign value to HEX0 and HEX1. The case that HEX0-HEX5 will all be activated and assigned to values is for count is 0 since it needs to display word CLEAR0. And when the counter reaches its MAX(25 or 5 for demo) it needs to display the word FULL, which uses HEX0-HEX3. I was able to get 7 bit binary values for HEX to display number 0-9 from my previous EE271 labs.



**Figure 1: State Diagram of enter/exit FSM - Task 1**

PSx	PSA	PSB	Ain	Bin	NSx	NSa	NSb	en	ex
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0
0	0	0	1	0	1	1	0	0	0
0	0	0	1	1	x	x	x	x	x
0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	0	0
0	0	1	1	0	x	x	x	x	x
0	0	1	1	1	0	1	1	0	0
0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	x	x	x	x	x
0	1	0	1	0	0	1	0	0	0
0	1	0	1	1	0	1	1	0	0
0	1	1	0	0	x	x	x	x	x
0	1	1	0	1	0	0	1	0	0
0	1	1	1	0	0	1	0	0	0
0	1	1	1	1	0	1	1	0	0
1	0	0	0	0	x	x	x	x	x
1	0	0	0	1	x	x	x	x	x
1	0	0	1	0	x	x	x	x	x
1	0	0	1	1	x	x	x	x	x
1	0	1	0	0	0	0	0	1	0
1	0	1	0	1	1	0	1	0	0
1	0	1	1	0	x	x	x	x	x
1	0	1	1	1	1	1	1	0	0
1	1	0	0	0	0	0	0	0	0
1	1	0	0	1	x	x	x	x	x
1	1	0	1	0	1	1	0	0	0
1	1	0	1	1	1	1	1	0	0
1	1	1	0	0	x	x	x	x	x
1	1	1	0	1	1	0	1	0	0
1	1	1	1	0	1	1	0	0	0
1	1	1	1	1	1	1	1	0	0

Figure 2: Truth Table for enter/exit FSM - Task 1

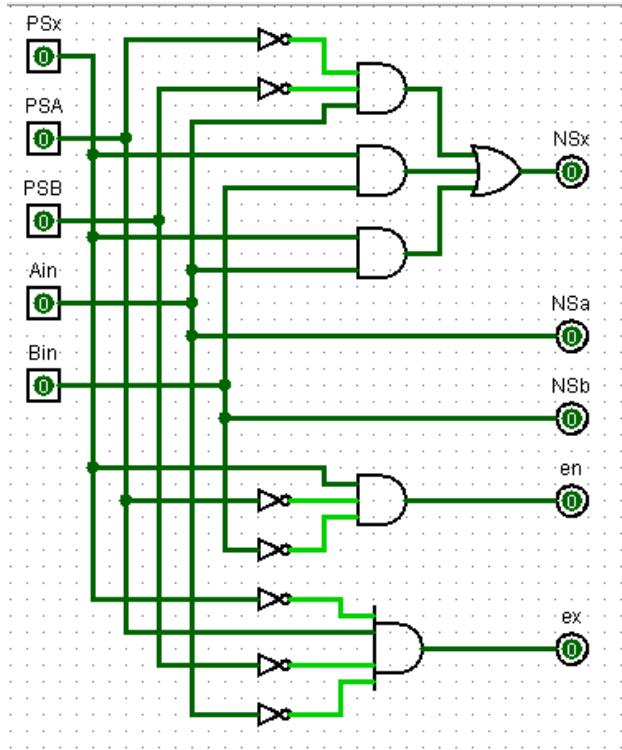


Figure 3: Circuit Diagram for enter/exit FSM - Task 1

## Results

### DE1\_SoC\_testbench():

This module tests 4 cases:

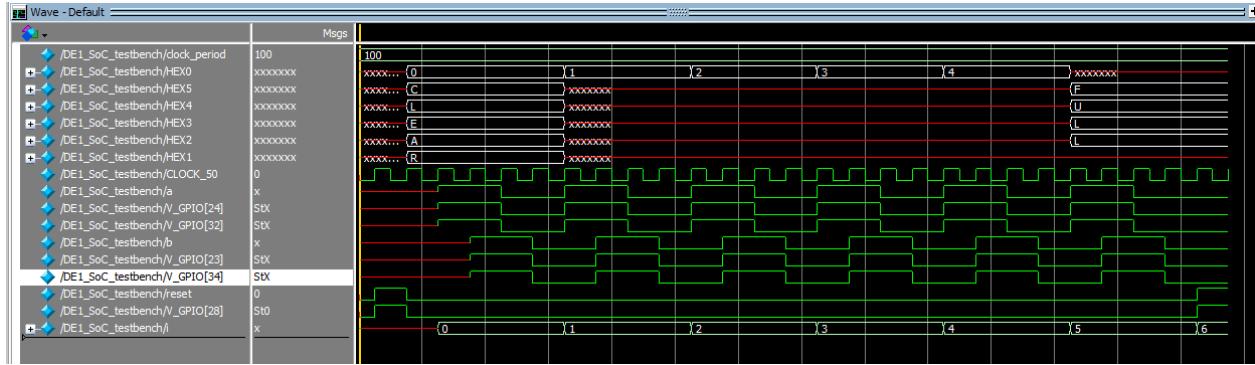
- 1) Enter
- 2) Exit
- 3) One car stalls at entrance while one car fully exits
- 4) Backup from entrance

It has 3 outputs:

- 1) [6:0] HEX
- 2) GPIO[32] - LED output for a
- 3) GPIO[34] - LED output for b

It has 3 inputs:

- 1) a : GPIO[24]
- 2) b : GPIO[23]
- 3) reset : GPIO[28]



**Figure 4: DE1\_SoC testbench ModelSim**

According to Figure 4, Looking at this signal with the given inputs and outputs, we can see that it does show a “CLEAR0” display when the reset is switched to one. Upon 5 (marked by “i”) instances of an enter sequence ( $ab = 10,11,01,00$ ) , we get the “FULL” display. Also, we can see the HEX0 output increment by one for each entrance. These qualities are aligned with our expectations for the module’s behavior

### enex\_testbench():

This module tests 4 cases:

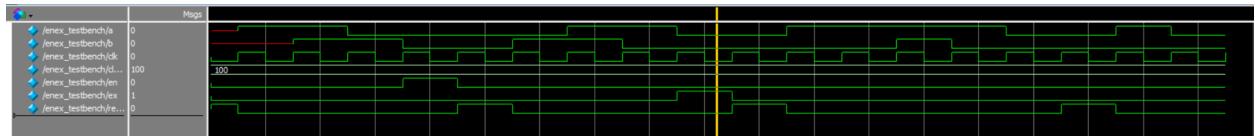
- 1) Enter
- 2) Exit
- 3) One car stalls at entrance while one car fully exits
- 4) Backup from entrance

It has 2 outputs:

- 1) en - enter
- 2) ex - exit

It has 2 inputs:

- 1) a
- 2) b



**Figure 5: enex testbench ModelSim**

According to Figure 5, Looking at this signal with the given inputs and outputs, we can see that it simulates an entrance case when the inputs are in the given sequence at the positive clock edge ( $ab = 10,11,01,00$ ). Next, we can see that it resets and simulates an exit case where the inputs are ( $ab = 01,11,10,00$ ). Lastly, we can see that it does not give any output when the

inputs are in the sequence (ab = 10,11,10,00). All three cases are aligned with our expectations.

### Counter\_testbench():

This module tests 2 cases:

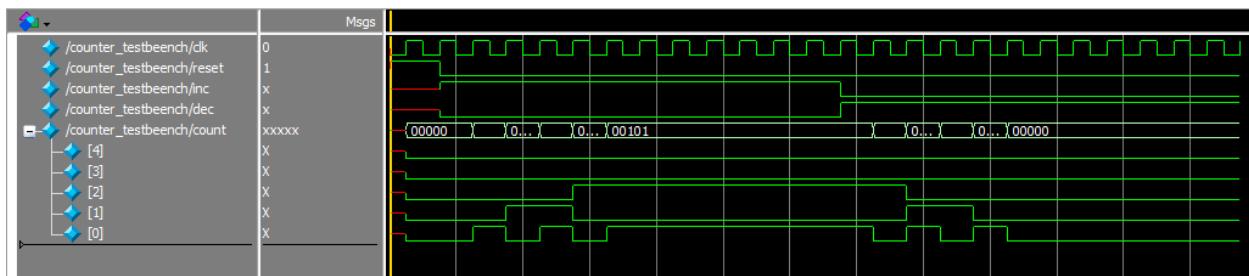
- 1) See if counter will increase accordingly without exceeding its upper bound (MAX)
- 2) See if counter stop get updated when it hits its MAX (MAX = 5)
- 3) See if counter will decrease accordingly without going below 0
- 4) See if counter will stop decreasing when it hits 0

It has 1 outputs:

- 1) count

It has 2 input:

- 1) inc - increasing
- 2) dec - decreasing



**Figure 6: counter testbench ModelSim**

According to Figure 6, Looking at this signal with the given inputs and outputs, we can see that it simulates cases that input inc is 1 at the positive clock edge for 6 times, which test if the counter will increase accordingly without exceeding its MAX bound; and then input dec is 1 at the positive clock edge for 6 times, which test will decrease accordingly without going below 0. All of the test cases are aligned with our expectations.

### display\_testbench():

This module tests 8 cases:

- 1) When the count is 0
- 2) When the count is 1
- 3) When the count is 2
- 4) When the count is 3
- 5) When the count is 4
- 6) When the count is 5
- 7) When the count is 6

8) When the count is 25

It has 6 outputs:

- 1) HEX0
- 2) HEX1
- 3) HEX2
- 4) HEX3
- 5) HEX4
- 6) HEX5

It has 1 input:

- 1) count

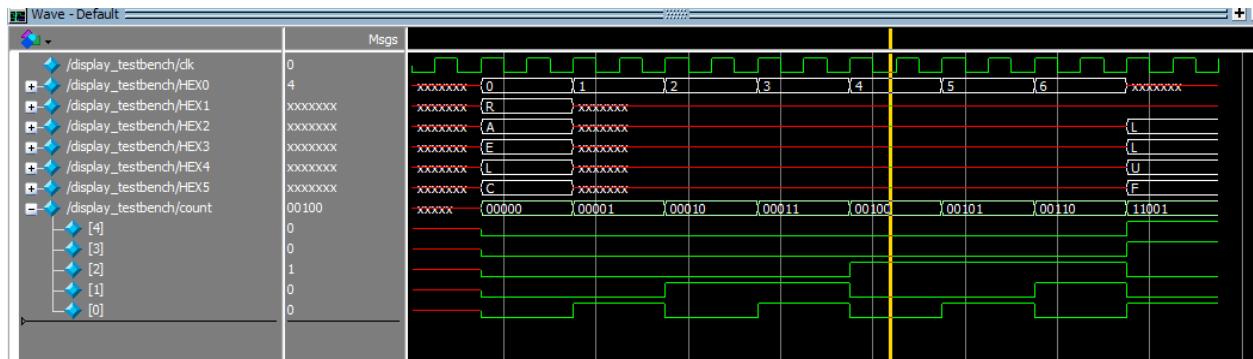


Figure 7: display testbench ModelSim

According to Figure 7, Looking at this signal with the given inputs and outputs, we can see that HEX0-HEX5 show a “CLEAR0” display when count is 0; As the count increases by 1, HEX0-HEX5 show the number accordingly; when the count hit its max 25, HEX0-HEX5 show the word “FULL”. All of the test cases are aligned with our expectations.

## Appendix

```
75      /*Write "9" to Address 21*/
76      SW[3:0] = 9;
77      SW[9] <= 1;          @(posedge KEY[0]);
78      SW[9] <= 0;          @(posedge KEY[0]);
79
80      $stop;
81  end //initial
82 endmodule //De1_SoC_testbench
```

```

1  /*
2   * Justin Sim and Mina Gao
3   * 1/23/2024
4   * EE 371 Hussein
5   * Lab 2, Task 1
6
7   * This module is a memory-unit-like design which models after a
8   * a 32x4 RAM chip
9
10  inputs:
11    clk          clock
12    [4:0] addr  address label
13    [3:0] din   data in
14    write        write enable
15  outputs:
16    [3:0] dout  data out
17
18 */
19 module M10K_32_4(clk,addr,din,write,dout);
20   input logic clk;
21   input logic [4:0] addr;
22   input logic [3:0] din;
23   input logic write;
24   output logic [3:0] dout;
25
26   logic [3:0] mem [31:0];
27
28   always_ff @(posedge clk) begin
29     if (write)
30       // Write data to the specified address on the rising edge of the clock
31       mem[addr] <= din;
32   end
33
34   // Unregistered data output
35   assign dout = mem[addr];
36
37 endmodule           //M10K_32_4
38
39 //-----*/
40 // M10K_32_4_testbench tests the expected and unexpected behaviors,
41 // reset behaviors, and random input behaviors
42
43 module M10K_32_4_testbench ();
44   logic clk;
45   logic [4:0] addr;
46   logic [3:0] din;
47   logic write;
48   logic [3:0] dout;
49
50   M10K_32_4 dut(clk,addr,din,write,dout);
51
52   // clock setup
53   parameter clock_period = 100;
54   initial begin
55     clk <= 0;
56     forever #(clock_period / 2) clk <= ~clk;
57   end
58
59   initial begin
60     /*Write data to address 5*/
61     addr <= 5;
62     din <= 4'b1010;
63     write <= 1; @(posedge clk)
64     write <= 0; @(posedge clk)
65
66     /*Read data from address 5*/
67     addr = 5; @(posedge clk)
68     $stop;
69   end //initial
70 endmodule           //M10K_32_4_testbench

```

```
305      count = 5'b00101;      @(posedge clk);
306      count = 5'b00110;      @(posedge clk);
307      count = 5'b11001;      @(posedge clk);
308
309
310
311      $stop; // end simulation
312
313 endmodule          //display_31digit_testbench
```

```

1  /*
2   * Justin Sim and Mina Gao
3   * 1/23/2024
4   * EE 371 Hussein
5   * Lab 2
6
7   * This module will create a Hexadecimal rendering of input data for
8   * numbers between 0 and 31
9   * It takes in any 5'b input and returns the corresponding number for the
10  * DE1_SoC 7'b HEX displays
11
12  inputs:
13    clk          clock
14    [4:0] count  user input (SW)
15
16  outputs:
17    [6:0] HEX:    7'b HEX display on DE1_SoC
18 */
19 module display_31digit (clk, count, HEX0, HEX1);
20   input logic clk;
21   input logic [4:0] count;
22   output logic [6:0] HEX0, HEX1;
23
24   //for 0-9
25   logic [6:0] h0, h1, h2, h3, h4, h5, h6, h7, h8, h9, hoff;
// hC, hL, hE, hA, hR, hF, hU;
27
28   assign h0 = 7'b1000000; // 0
29   assign h1 = 7'b1111001; // 1
30   assign h2 = 7'b0100010; // 2
31   assign h3 = 7'b0110000; // 3
32   assign h4 = 7'b0011001; // 4
33   assign h5 = 7'b00100010; // 5
34   assign h6 = 7'b00000010; // 6
35   assign h7 = 7'b1111000; // 7
36   assign h8 = 7'b00000000; // 8
37   assign h9 = 7'b00100000; // 9
38   assign hoff = 7'b1111111; // no display
39
40   //for letters (clear, full)
41   // assign hC = 7'b10000110; // C
42   // assign hL = 7'b10000111; // L
43   // assign hE = 7'b00000110; // E
44   // assign hA = 7'b00001000; // A
45   // assign hR = 7'b01011111; // R
46   // assign hF = 7'b00001110; // F
47   // assign hU = 7'b10000001; // U
48
49   always_comb begin
50     case(count)
51       // 0
52       5'b00000:
53         begin
54           HEX1 = hoff;
55           HEX0 = h0;
56         end
57
58       // 1
59       5'b00001:
60         begin
61           HEX1 = hoff;
62           HEX0 = h1;
63         end
64
65       // 2
66       5'b00010:
67         begin
68           HEX1 = hoff;
69           HEX0 = h2;
70         end
71
72       // 3
73       5'b00011:
74         begin
75           HEX1 = hoff;
76           HEX0 = h3;
77         end
78     endcase
79   end
80 endmodule

```

```
77          end
78
79      // 4
80      5'b00100:
81      begin
82          HEX1 = h0ff;
83          HEX0 = h4;
84      end
85
86      // 5
87      5'b00101:
88      begin
89          HEX1 = h0ff;
90          HEX0 = h5;
91      end
92
93      // 6
94      5'b00110:
95      begin
96          HEX1 = h0ff;
97          HEX0 = h6;
98      end
99
100     // 7
101    5'b00111:
102    begin
103        HEX1 = h0ff;
104        HEX0 = h7;
105    end
106
107     // 8
108    5'b01000:
109    begin
110        HEX1 = h0ff;
111        HEX0 = h8;
112    end
113
114     // 9
115    5'b01001:
116    begin
117        HEX1 = h0ff;
118        HEX0 = h9;
119    end
120
121     // 10
122    5'b01010:
123    begin
124        HEX1 = h1;
125        HEX0 = h0;
126    end
127
128     // 11
129    5'b01011:
130    begin
131        HEX1 = h1;
132        HEX0 = h1;
133    end
134
135     // 12
136    5'b01100:
137    begin
138        HEX1 = h2;
139        HEX0 = h1;
140    end
141
142     // 13
143    5'b01101:
144    begin
145        HEX1 = h3;
146        HEX0 = h1;
147    end
148
149     // 14
150    5'b01110:
151    begin
152        HEX1 = h4;
```

```
153           HEX0 = h1;
154       end
155
156 // 15
157 5'b01111:
158 begin
159     HEX1 = h5;
160     HEX0 = h1;
161 end
162
163 // 16
164 5'b10000:
165 begin
166     HEX1 = h6;
167     HEX0 = h1;
168 end
169
170 // 17
171 5'b10001:
172 begin
173     HEX1 = h7;
174     HEX0 = h1;
175 end
176
177 // 18
178 5'b10010:
179 begin
180     HEX1 = h8;
181     HEX0 = h1;
182 end
183
184 // 19
185 5'b10011:
186 begin
187     HEX1 = h9;
188     HEX0 = h1;
189 end
190
191 // 20
192 5'b10100:
193 begin
194     HEX1 = h0;
195     HEX0 = h2;
196 end
197
198 // 21
199 5'b10101:
200 begin
201     HEX1 = h1;
202     HEX0 = h2;
203 end
204
205 // 22
206 5'b10110:
207 begin
208     HEX1 = h2;
209     HEX0 = h2;
210 end
211
212 // 23
213 5'b10111:
214 begin
215     HEX1 = h3;
216     HEX0 = h2;
217 end
218
219 // 24
220 5'b11000:
221 begin
222     HEX1 = h4;
223     HEX0 = h2;
224 end
225
226 // 25
227 5'b11001:
228 begin
```

```

229                     HEX1 = h5;
230                     HEX0 = h2;
231                 end
232
233             // 26
234             5'b11010:
235             begin
236                 HEX1 = h6;
237                 HEX0 = h2;
238             end
239
240             // 27
241             5'b11011:
242             begin
243                 HEX1 = h7;
244                 HEX0 = h2;
245             end
246             // 28
247             5'b11100:
248             begin
249                 HEX1 = h8;
250                 HEX0 = h2;
251             end
252             // 29
253             5'b11101:
254             begin
255                 HEX1 = h9;
256                 HEX0 = h2;
257             end
258             // 30
259             5'b11110:
260             begin
261                 HEX1 = h0;
262                 HEX0 = h3;
263             end
264             // 31
265             5'b11111:
266             begin
267                 HEX1 = h1;
268                 HEX0 = h3;
269             end
270         endcase
271     end
272 endmodule          //display_31digit
273
274 //-----// display_31digit_testbench tests the expected and unexpected behaviors,
275 //-----// reset behaviors, and random input behaviors
276
277 module display_31digit_testbench ();
278     logic clk;
279     logic [6:0] HEX0, HEX1;
280     logic [4:0] count;
281
282     display_31digit dut (.clk(clk), .count(count), .HEX0(HEX0), .HEX1(HEX1));
283
284     // clock setup
285     parameter clock_period = 100;
286
287     initial begin
288         clk <= 0;
289         forever #(clock_period / 2) clk <= ~clk;
290     end
291
292     initial begin
293         count = 5'b00000;           @(posedge clk);
294         count = 5'b00001;           @(posedge clk);
295         count = 5'b00010;           @(posedge clk);
296         count = 5'b00011;           @(posedge clk);
297         count = 5'b00100;           @(posedge clk);
298         count = 5'b00101;           @(posedge clk);
299         count = 5'b00110;           @(posedge clk);
300         count = 5'b00111;           @(posedge clk);
301         count = 5'b01000;           @(posedge clk);
302         count = 5'b01001;           @(posedge clk);
303         count = 5'b01010;           @(posedge clk);
304     end

```

```

1  /*
2   * Justin Sim and Mina Gao
3   * 1/23/2024
4   * EE 371 Hussein
5   * Lab 2
6
7   * This module converts 4'b binary input to 7'b Hexadecimal for the DE1_SoC
8
9   * inputs:
10  *     clk          clock
11  *     [3:0] binary_input    user input (SW)
12  * outputs:
13  *     [7:0] hexadecimal_output:      7'b HEX display on DE1_SoC
14
15 */
16 module HexadecimalConverter (clk, binary_input, hexadecimal_output);
17     input logic [3:0] binary_input;
18     input logic clk;
19     output logic [6:0] hexadecimal_output;
20     logic [6:0] h0, h1, h2, h3, h4, h5, h6, h7, h8, h9,
21             hA, hB, hC, hD, hE, hF, hoff;
22
23     assign h0 = 7'b1000000; // 0
24     assign h1 = 7'b1111001; // 1
25     assign h2 = 7'b0100100; // 2
26     assign h3 = 7'b0110000; // 3
27     assign h4 = 7'b0011001; // 4
28     assign h5 = 7'b0010010; // 5
29     assign h6 = 7'b0000010; // 6
30     assign h7 = 7'b1111100; // 7
31     assign h8 = 7'b0000000; // 8
32     assign h9 = 7'b0010000; // 9
33     assign hA = 7'b0001000; // A
34     assign hB = 7'b0000011; // B
35     assign hC = 7'b1000110; // C
36     assign hD = 7'b0100001; // D
37     assign hE = 7'b0000110; // E
38     assign hF = 7'b0001110; // F
39     assign hoff = 7'b1111111; // no display
40
41     always_comb begin
42         case (binary_input)
43             4'b0000: hexadecimal_output = h0;
44             4'b0001: hexadecimal_output = h1;
45             4'b0010: hexadecimal_output = h2;
46             4'b0011: hexadecimal_output = h3;
47             4'b0100: hexadecimal_output = h4;
48             4'b0101: hexadecimal_output = h5;
49             4'b0110: hexadecimal_output = h6;
50             4'b0111: hexadecimal_output = h7;
51             4'b1000: hexadecimal_output = h8;
52             4'b1001: hexadecimal_output = h9;
53             4'b1010: hexadecimal_output = hA;
54             4'b1011: hexadecimal_output = hB;
55             4'b1100: hexadecimal_output = hC;
56             4'b1101: hexadecimal_output = hD;
57             4'b1110: hexadecimal_output = hE;
58             4'b1111: hexadecimal_output = hF;
59             4'bXXXX: hexadecimal_output = hoff; // Handle other cases as needed
60         endcase
61     end
62
63     endmodule // HexadecimalConverter
64
65 //-----//
66 //HexadecimalConverter_testbench tests the expected and unexpected behaviors,
67 // reset behaviors, and random input behaviors
68
69 module HexadecimalConverter_testbench ;
70     logic [3:0] binary_input;
71     logic [7:0] hexadecimal_output;
72     logic clock;
73
74     // Instantiate HexadecimalConverter module
75     HexadecimalConverter dut (
76         .binary_input(binary_input),

```

```
77      .hexadecimal_output(hexadecimal_output)
78  );
79
80 // clock setup
81 parameter clock_period = 100;
82 initial begin
83   clock <= 0;
84   forever #(clock_period / 2) clock <= ~clock;
85 end
86
87 // Test stimulus
88 initial begin
89   binary_input = 4'b0000;      @(posedge clock);
90
91   // Apply test vectors
92   binary_input = 4'b1010;      @(posedge clock); // Binary input: 1010 (Decimal: 10)
93   binary_input = 4'b1101;      @(posedge clock); // Binary input: 1101 (Decimal: 13)
94   binary_input = 4'b0111;      @(posedge clock); // Binary input: 0111 (Decimal: 7)
95
96   // Add more test vectors as needed
97
98 $stop; // Stop simulation after a certain duration
99 end
100
101 endmodule          // HexadecimalConverter_testbench
```

```

1  /*
2   * Justin Sim and Mina Gao
3   * 1/23/2024
4   * EE 371 Hussein
5   * Lab 2, Task 2
6
7   * This module will keep track of how many cars have entered
8   * or exited the garage.
9   * It uses two sensors, a and b, to detect vehicle
10  * movement in and out of the garage, implements an fsm and counter.
11  * The output will be on the HEX displaying the number of cars
12  * currently occupying the garage, and will say FULL when
13  * the maximum capacity (editable parameter) has been reached
14
15  * inputs:
16  *     CLOCK_50
17  *     [35:0] V_GPIO:      virtual GPIO for DE1_SoC
18  * in/outputs:
19  *     [6:0] HEX:          7'b HEX display on DE1_SoC
20
21 */
22 module DE1_SoC(CLOCK_50, KEY, SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);
23   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
24   input logic [9:0] SW;
25   input logic [3:0] KEY;
26   input logic CLOCK_50;
27
28   logic [4:0] rdaddress;
29   logic [3:0] dout;
30   logic reset;
31   assign reset = ~KEY[0];
32
33   logic [31:0] div_clk;
34   parameter whichClock = 25; // 0.75 Hz clock
35   clock_divider cdiv (.clock(CLOCK_50),
36   .reset(reset),
37   .divided_clocks(div_clk));
38   // Clock selection; allows for easy switching between simulation and boardclocks
39   logic clkSelect;
40
41   // Uncomment ONE of the following two lines depending on intention
42   assign clkSelect = CLOCK_50; // for simulation
43   // assign clkSelect = div_clk[whichClock]; // for board
44
45   // Instantiate the memory with MIF
46   ram32x4 ram (
47     .clock(CLOCK_50),
48     .data(SW[3:0]),
49     .rdaddress(rdaddress),
50     .wraddress(SW[8:4]),
51     .wren(~KEY[3]),
52     .q(dout));
53
54   /* Increment rdaddress by 1 every second, using clkselect (1 sec)*/
55   counter ct (.clk(clkSelect), .reset(reset), .rdaddress(rdaddress));
56
57   /* Displays the write address on HEX 4-5 */
58   display_31digit disp_wr_addr (.clk(CLOCK_50), .count(SW[8:4]), .HEX0(HEX5), .HEX1(HEX4));
59
60   /* Displays the read address on HEX 2-3 */
61   display_31digit disp_rd_addr (.clk(CLOCK_50), .count(rdaddress), .HEX0(HEX3), .HEX1(HEX2));
62
63   /* Displays the user input on HEX 1*/
64   HexadecimalConverter disp_din (.clk(CLOCK_50), .binary_input(SW[3:0]), .
65   hexadecimal_output(HEX1));
66
67   /*Displays the data at specified address*/
68   HexadecimalConverter disp_dout (.clk(CLOCK_50), .binary_input(dout), .hexadecimal_output
69   (HEX0));
70
71 endmodule // DE1_SoC
72 //-----
73 //DE1_SoC_testbench tests the expected and unexpected behaviors, reset behaviors
74 //specifically, it tests a full cycle of scrolling through addresses

```

```

73 // and shows a case of the RAM's write function
74 `timescale 1 ps / 1 ps
75 module DE1_SoC_testbench();
76   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
77   logic [9:0] SW;
78   logic [3:0] KEY;
79   logic CLOCK_50;
80
81   DE1_SoC dut(.CLOCK_50(CLOCK_50), .HEX0(HEX0), .HEX1(HEX1),
82   .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5), .KEY(KEY), .SW(SW));
83
84 // clock setup
85 parameter clock_period = 100;
86 initial begin
87   CLOCK_50 <= 0;
88   forever #(clock_period / 2) CLOCK_50 <= ~CLOCK_50;
89 end
90
91   integer i;
92   integer n;
93
94 initial begin
95   // count up
96   KEY[0] = 0; @(posedge CLOCK_50);
97   KEY[0] = 1; @(posedge CLOCK_50);
98   for (n=1; n<=31; n++) begin
99     @(posedge CLOCK_50);
100   end
101
102   // write "0" to address 4
103
104   SW[8:4] = 5'b00100;
105   SW[3:0] = 4'b0000; @(posedge CLOCK_50); //write enable
106   KEY[3] = 0; @(posedge CLOCK_50); @(posedge CLOCK_50);
107   KEY[3] = 1; @(posedge CLOCK_50);
108   @(posedge CLOCK_50);
109   @(posedge CLOCK_50);
110   @(posedge CLOCK_50);
111   @(posedge CLOCK_50);
112   @(posedge CLOCK_50);
113   $stop;
114 end //initial
115 endmodule // DE1_SoC_testbench

```

```
1  /*
2   * Justin Sim and Mina Gao
3   * 1/23/2024
4   * EE 371 Hussein
5   * Lab 2, Task 2
6
7   * This module divides the input clock such that it can slow down the cycle:
8   * divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz,[25] = 0.75Hz, ...
9
10  inputs:
11    clock
12  in/outputs:
13    [31:0] divided_clocks          array of varying clock speeds
14
15 */
16 module clock_divider (clock, reset, divided_clocks);
17   input logic reset, clock;
18   output logic [31:0] divided_clocks = 0;
19   always_ff @(posedge clock) begin
20     divided_clocks <= divided_clocks + 1;
21   end
22 endmodule // clock_divider
23 //-----
24 // clock_divider_testbench tests the expected behaviors of the clock_divider module
25
26 module clock_divider_testbench ();
27   logic reset, clock;
28   logic [31:0] divided_clocks;
29
30   clock_divider dut (.clock, .reset, .divided_clocks);
31
32 // clock setup
33 parameter clock_period = 100;
34 initial begin
35   clock <= 0;
36   forever #(clock_period / 2) clock <= ~clock;
37 end
38 integer n;
39 initial begin
40   for (n=1; n<=31; n++) begin
41     @(posedge clock);
42   end
43   reset = 1;
44   $stop;
45 end
46
47 endmodule // clock_divider_testbench
```

```

1  /*
2  Justin Sim and Mina Gao
3  1/23/2024
4  EE 371 Hussein
5  Lab 2, Task 2
6
7  This module will provide a count up from 0 to 31
8  starts the count back at 0 when reset
9  will automatically cycle back to 0 after reaching 31
10
11  inputs:
12      clk          clock
13      reset
14  in/outputs:
15      [4:0] rdaddress    read address
16
17 */
18 module counter (clk, reset, rdaddress);
19     input logic clk, reset;
20     output logic [4:0] rdaddress;
21
22     always_ff @(posedge clk) begin
23         if (reset) rdaddress <= 5'b00000;
24
25         else begin
26             if (rdaddress == 5'b11111) // edge case
27                 rdaddress <= 5'b00000;
28             else
29                 rdaddress <= rdaddress + 5'd00001;
30         end
31     end
32 endmodule // counter
33
34 //-----//
35 // counter_testbench tests the expected and unexpected behaviors, reset behaviors
36 // specifically, it tests a full cycle from 0 - 31
37
38 module counter_testbench ();
39     logic clk, reset;
40     logic [4:0] rdaddress;
41
42     counter dut (.clk, .reset, .rdaddress);
43
44     // clock setup
45     parameter clock_period = 100;
46
47     initial begin
48         clk <= 0;
49         forever #(clock_period / 2) clk <= ~clk;
50     end
51
52     integer i;
53
54     initial begin
55         reset <= 0;          @(posedge clk);
56         reset <= 1;          @(posedge clk);
57         reset <= 0; repeat(34) @(posedge clk);
58         reset <= 1; @(posedge clk);
59         reset <= 0; repeat(5)  @(posedge clk);
60         $stop;
61     end
62 endmodule // counter_testbench

```

Date: January 23, 2024

ram32x4.mif

Project: DE1\_SoC

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
<b>0</b>	0	1	2	3	4	5	6	7	.....
<b>8</b>	0	2	4	6	8	10	12	14	.....
<b>16</b>	0	1	3	5	7	9	11	13	.....
<b>24</b>	7	6	5	4	3	2	1	0	.....

```

153 // Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
154 // Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "0"
155 // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
156 // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "2"
157 // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
158 // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_B NUMERIC "3"
159 // Retrieval info: PRIVATE: REGdata NUMERIC "1"
160 // Retrieval info: PRIVATE: REGq NUMERIC "1"
161 // Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
162 // Retrieval info: PRIVATE: REGrren NUMERIC "1"
163 // Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
164 // Retrieval info: PRIVATE: REGwren NUMERIC "1"
165 // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
166 // Retrieval info: PRIVATE: USE_DIFF_CLKEN NUMERIC "0"
167 // Retrieval info: PRIVATE: UseDRAM NUMERIC "1"
168 // Retrieval info: PRIVATE: Varwidth NUMERIC "0"
169 // Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "4"
170 // Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "4"
171 // Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "4"
172 // Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "4"
173 // Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "0"
174 // Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "0"
175 // Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "0"
176 // Retrieval info: PRIVATE: enable NUMERIC "0"
177 // Retrieval info: PRIVATE: rden NUMERIC "0"
178 // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
179 // Retrieval info: CONSTANT: ADDRESS_ACLR_B STRING "NONE"
180 // Retrieval info: CONSTANT: ADDRESS_REG_B STRING "CLOCK0"
181 // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
182 // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_B STRING "BYPASS"
183 // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_B STRING "BYPASS"
184 // Retrieval info: CONSTANT: INIT_FILE STRING "ram32x4.mif"
185 // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
186 // Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
187 // Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "32"
188 // Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "32"
189 // Retrieval info: CONSTANT: OPERATION_MODE STRING "DUAL_PORT"
190 // Retrieval info: CONSTANT: OUTDATA_ACLR_B STRING "NONE"
191 // Retrieval info: CONSTANT: OUTDATA_REG_B STRING "UNREGISTERED"
192 // Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
193 // Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
194 // Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_MIXED_PORTS STRING "DONT_CARE"
195 // Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "5"
196 // Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "5"
197 // Retrieval info: CONSTANT: WIDTH_A NUMERIC "4"
198 // Retrieval info: CONSTANT: WIDTH_B NUMERIC "4"
199 // Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
200 // Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
201 // Retrieval info: USED_PORT: data 0 0 4 0 INPUT NODEFVAL "data[3..0]"
202 // Retrieval info: USED_PORT: q 0 0 4 0 OUTPUT NODEFVAL "q[3..0]"
203 // Retrieval info: USED_PORT: rdaddress 0 0 5 0 INPUT NODEFVAL "rdaddress[4..0]"
204 // Retrieval info: USED_PORT: wraddress 0 0 5 0 INPUT NODEFVAL "wraddress[4..0]"
205 // Retrieval info: USED_PORT: wren 0 0 0 0 INPUT GND "wren"
206 // Retrieval info: CONNECT: @address_a 0 0 5 0 wraddress 0 0 5 0
207 // Retrieval info: CONNECT: @address_b 0 0 5 0 rdaddress 0 0 5 0
208 // Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
209 // Retrieval info: CONNECT: @data_a 0 0 4 0 data 0 0 4 0
210 // Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
211 // Retrieval info: CONNECT: q 0 0 4 0 @q_b 0 0 4 0
212 // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.v TRUE
213 // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.inc FALSE
214 // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.cmp FALSE
215 // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.bsf FALSE
216 // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_inst.v FALSE
217 // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_bb.v TRUE
218 // Retrieval info: LIB_FILE: altera_mf
219

```

```

1 // megafunction wizard: %RAM: 2-PORT%
2 // GENERATION: STANDARD
3 // VERSION: WM1.0
4 // MODULE: altsyncram
5
6 // =====
7 // File Name: ram32x4.v
8 // Megafunction Name(s):
9 //      altsyncram
10 //
11 // Simulation Library Files(s):
12 //      altera_mf
13 // =====
14 // **** THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
15 //
16 // 17.0.0 Build 595 04/25/2017 SJ Lite Edition
17 // ****
18 //
19 //
20
21 //Copyright (C) 2017 Intel Corporation. All rights reserved.
22 //Your use of Intel Corporation's design tools, logic functions
23 //and other software and tools, and its AMPP partner logic
24 //functions, and any output files from any of the foregoing
25 // (including device programming or simulation files), and any
26 //associated documentation or information are expressly subject
27 //to the terms and conditions of the Intel Program License
28 //Subscription Agreement, the Intel Quartus Prime License Agreement,
29 //the Intel MegaCore Function License Agreement, or other
30 //applicable license agreement, including, without limitation,
31 //that your use is for the sole purpose of programming logic
32 //devices manufactured by Intel and sold by Intel or its
33 //authorized distributors. Please refer to the applicable
34 //agreement for further details.
35
36
37 // synopsys translate_off
38 timescale 1 ps / 1 ps
39 // synopsys translate_on
40 module ram32x4 (
41     clock,
42     data,
43     rdaddress,
44     wraddress,
45     wren,
46     q);
47
48     input    clock;
49     input [3:0] data;
50     input [4:0] rdaddress;
51     input [4:0] wraddress;
52     input    wren;
53     output   [3:0] q;
54 `ifndef ALTERA_RESERVED_QIS
55 // synopsys translate_off
56 endif
57     tri1    clock;
58     tri0    wren;
59 `ifndef ALTERA_RESERVED_QIS
60 // synopsys translate_on
61 endif
62
63     wire [3:0] sub_wire0;
64     wire [3:0] q = sub_wire0[3:0];
65
66     altsyncram altsyncram_component (
67         .address_a (wraddress),
68         .address_b (rdaddress),
69         .clock0 (clock),
70         .data_a (data),
71         .wren_a (wren),
72         .q_b (sub_wire0),
73         .aclr0 (1'b0),
74         .aclr1 (1'b0),
75         .addressesall_a (1'b0),
76         .addressesall_b (1'b0),

```

```

77      .byteena_a (1'b1),
78      .byteena_b (1'b1),
79      .clock1 (1'b1),
80      .clocken0 (1'b1),
81      .clocken1 (1'b1),
82      .clocken2 (1'b1),
83      .clocken3 (1'b1),
84      .data_b ({4{1'b1}}),
85      .eccstatus (),
86      .q_a (),
87      .rden_a (1'b1),
88      .rden_b (1'b1),
89      .wren_b (1'b0));
90
91      defparam
92          altsyncram_component.address_aclr_b = "NONE",
93          altsyncram_component.address_reg_b = "CLOCK0",
94          altsyncram_component.clock_enable_input_a = "BYPASS",
95          altsyncram_component.clock_enable_input_b = "BYPASS",
96          altsyncram_component.clock_enable_output_b = "BYPASS",
97          altsyncram_component.init_file = "ram32x4.mif",
98          altsyncram_component.intended_device_family = "Cyclone V",
99          altsyncram_component.lpm_type = "altsyncram",
100         altsyncram_component.numwords_a = 32,
101         altsyncram_component.numwords_b = 32,
102         altsyncram_component.operation_mode = "DUAL_PORT",
103         altsyncram_component.outdata_aclr_b = "NONE",
104         altsyncram_component.outdata_reg_b = "UNREGISTERED",
105         altsyncram_component.power_up_uninitialized = "FALSE",
106         altsyncram_component.ram_block_type = "M10K",
107         altsyncram_component.read_during_write_mode_mixed_ports = "DONT CARE",
108         altsyncram_component.widthad_a = 5,
109         altsyncram_component.widthad_b = 5,
110         altsyncram_component.width_a = 4,
111         altsyncram_component.width_b = 4,
112         altsyncram_component.width_byteena_a = 1;
113
114     endmodule
115
116 // =====
117 // CNX file retrieval info
118 // =====
119 // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
120 // Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
121 // Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
122 // Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
123 // Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
124 // Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
125 // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
126 // Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
127 // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
128 // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
129 // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
130 // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
131 // Retrieval info: PRIVATE: CLRdata NUMERIC "0"
132 // Retrieval info: PRIVATE: CLRq NUMERIC "0"
133 // Retrieval info: PRIVATE: CLRraddress NUMERIC "0"
134 // Retrieval info: PRIVATE: CLRrren NUMERIC "0"
135 // Retrieval info: PRIVATE: CLRwaddress NUMERIC "0"
136 // Retrieval info: PRIVATE: CLRwren NUMERIC "0"
137 // Retrieval info: PRIVATE: Clock NUMERIC "0"
138 // Retrieval info: PRIVATE: Clock_A NUMERIC "0"
139 // Retrieval info: PRIVATE: Clock_B NUMERIC "0"
140 // Retrieval info: PRIVATE: IMPLEMENT_IN_LLES NUMERIC "0"
141 // Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
142 // Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "0"
143 // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_B"
144 // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
145 // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
146 // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
147 // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
148 // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
149 // Retrieval info: PRIVATE: MEMSIZE NUMERIC "128"
150 // Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
151 // Retrieval info: PRIVATE: MIFFilename STRING "ram32x4.mif"
152 // Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"

```

```

1  /*
2   * Justin Sim and Mina Gao
3   * 1/23/2024
4   * EE 371 Hussein
5   * Lab 2, Task 1
6
7   * This module implements a memory-unit-like design which demonstrates
8   * a 32x4 RAM chip
9   * It uses switches on the DE1_SoC as input data for the RAM-like design
10  * It outputs the user-input and stored memory values on the DE1_SoC HEXs
11
12  inputs:
13    All SW           switches on DE1_SoC
14    All KEY          keys on DE1_SoC
15  outputs:
16    [6:0] HEX:       7'b HEX display on DE1_SoC
17
18 */
19 module DE1_SoC(HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
20                 KEY, SW);
21   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
22   input logic [9:0] SW;
23   input logic [3:0] KEY;
24
25   logic [3:0] dout;
26   // Turn off Unused Displays
27   assign HEX1 = 7'b11111111;
28   assign HEX3 = 7'b11111111;
29
30   /* Reads din, and writes to the corresponding address
31      into internal memory, reads/returns the data stored
32      on given address when write is disabled */
33   M10K_32_4 mem (.clk(KEY[0]), .addr(sw[8:4]), .din(sw[3:0]),
34                  .write(SW[9]), .dout(dout));
35
36   /* Displays the given address on HEX 4-5 */
37   display_31digit disp_addr (.clk(KEY[0]), .count(sw[8:4]), .HEX0(HEX5), .HEX1(HEX4));
38
39   /* Displays the user input on HEX 2 */
40   HexadecimalConverter disp_din (.clk(KEY[0]), .binary_input(SW[3:0]), .hexadecimal_output
41                                 (HEX2));
42
43   /*Displays the data at specified address*/
44   HexadecimalConverter disp_dout (.clk(KEY[0]), .binary_input(dout), .hexadecimal_output (HEX0));
45
46 endmodule           //De1_SoC
47 //-----
48 //DE1_SoC_testbench tests the expected and unexpected behaviors,
49 // reset behaviors, and random input behaviors
50
51 module DE1_SoC_testbench();
52   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
53   logic [9:0] SW;
54   logic [3:0] KEY;
55
56   DE1_SoC dut(HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
57               KEY, SW);
58
59   // clock setup
60   parameter clock_period = 100;
61   initial begin
62     KEY[0] <= 0;
63     forever #(clock_period / 2) KEY[0] <= ~KEY[0];
64   end
65
66   initial begin
67     /*Display Address 15*/
68     SW[8:4] = 15; @(posedge KEY[0]);
69     /*Write "0" to Address 15*/
70     SW[3:0] = 4'b0000;
71     SW[9] <= 1;   @(posedge KEY[0]);
72     SW[9] <= 0;   @(posedge KEY[0]);
73     /*Display Address 21*/
74     SW[8:4] = 21; @(posedge KEY[0]);

```