Simon Chen
Cynthia Li 1839952
EE 371
May 15, 2021

Lab 2 Report

Procedure

This lab mainly involves two tasks: 1) construct a bit counter that is able to count the number of "1s" in a 8-bit value based on a given ASMD chart; 2) implement a binary search that is able to determine if a given 8-bit value exists in the ram and display the address of the value in the ram if found.

Task 1: we distinguish between the control and datapath from the ASMD chart given, write the code for datapath and control separately and simulate them on ModelSim, process the human input since we are using a fast clock, convert the count in binary form to decimal form for HEX display, and write everything up in the top module. Fig. 1 is the ASMD chart used for the bit counter, Fig. 2 is the block diagram for this task.

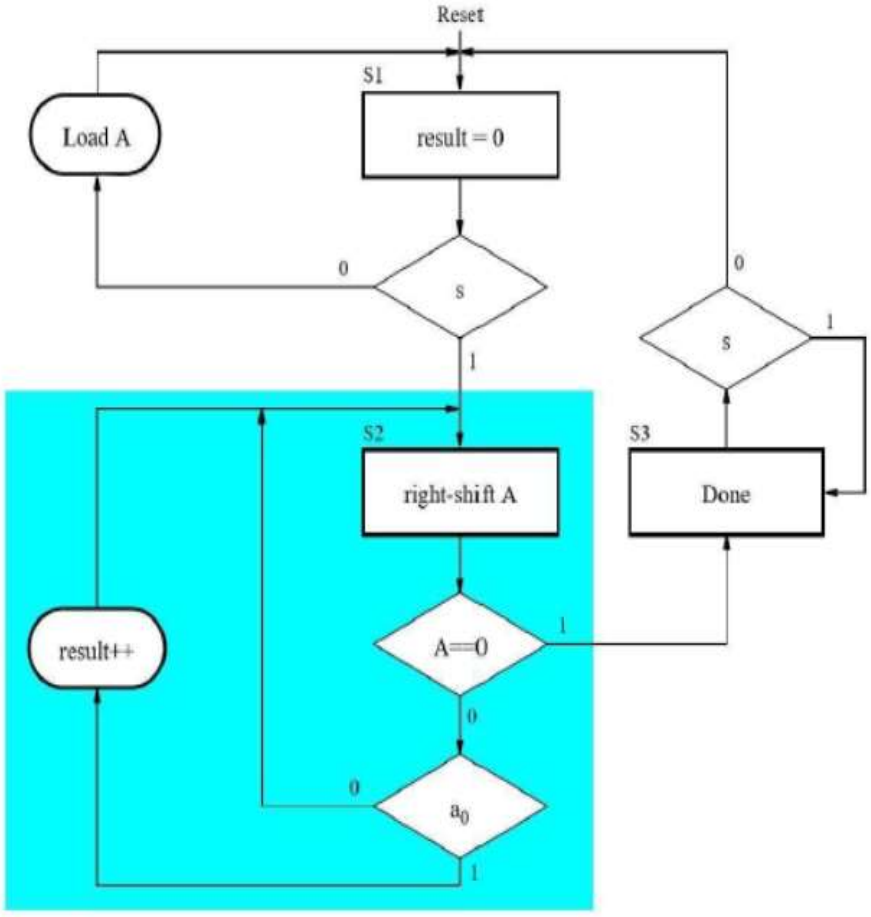

Figure 1. ASMD chart for a bit counting circuit

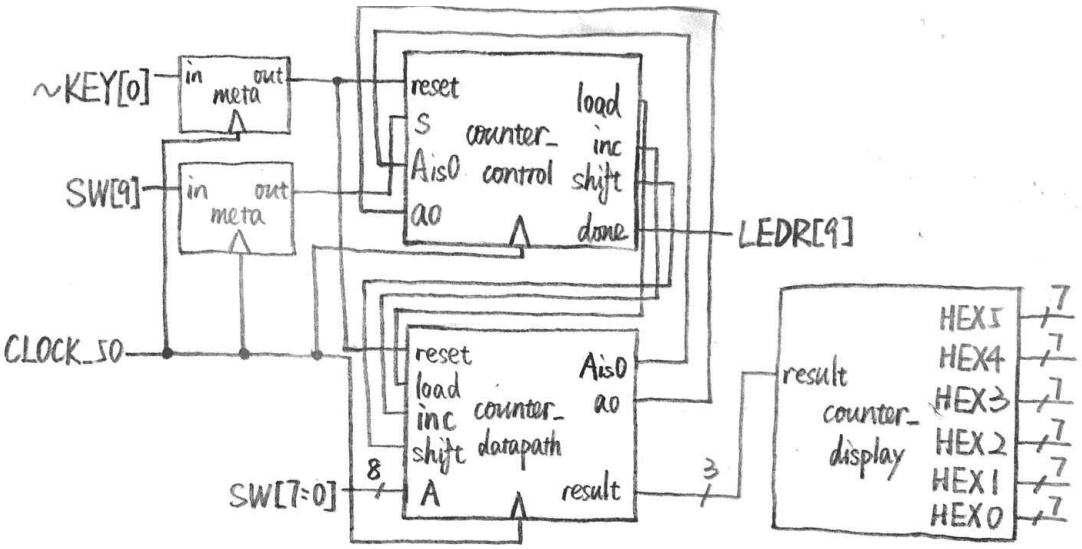

Fig.2 task1 top level block diagram

Task 2: For task 2 we first draw an ASMD chart for the binary search module. With the ASMD chart, it is very easy to implement it using system verilog. Since key0 is used for reset, and we are using a fast clock, we write a module to deal with metastability. We modify the hex display module from task 1 which makes it able to display hexadecimal numbers. Then we create a 32x8 ram module using IP catalog and create a mif for it. Lastly we write the top module.

Fig. 3 ASMD chart

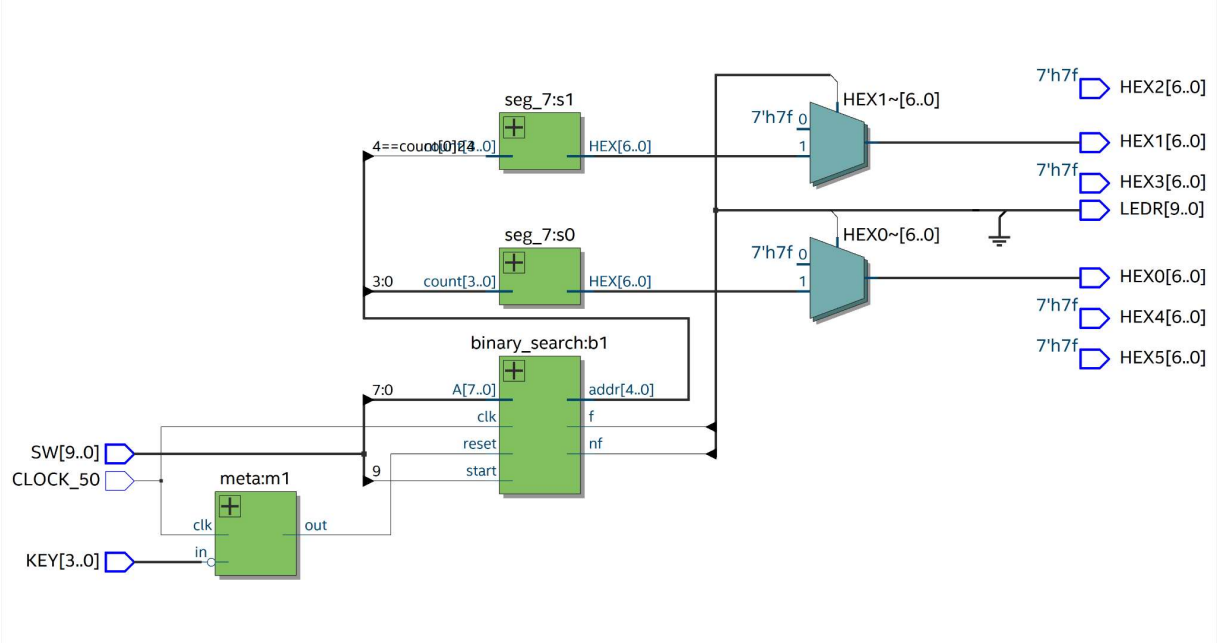

Fig. 4 Block diagram

Result

Task 1:

For task 1, we first analyze the given ASMD, distinguishing the output, conditional variables, states, etc. and extract the information for datapath and control. Then we first completed the code for datapath, which produces the simulation below in Fig. 5.

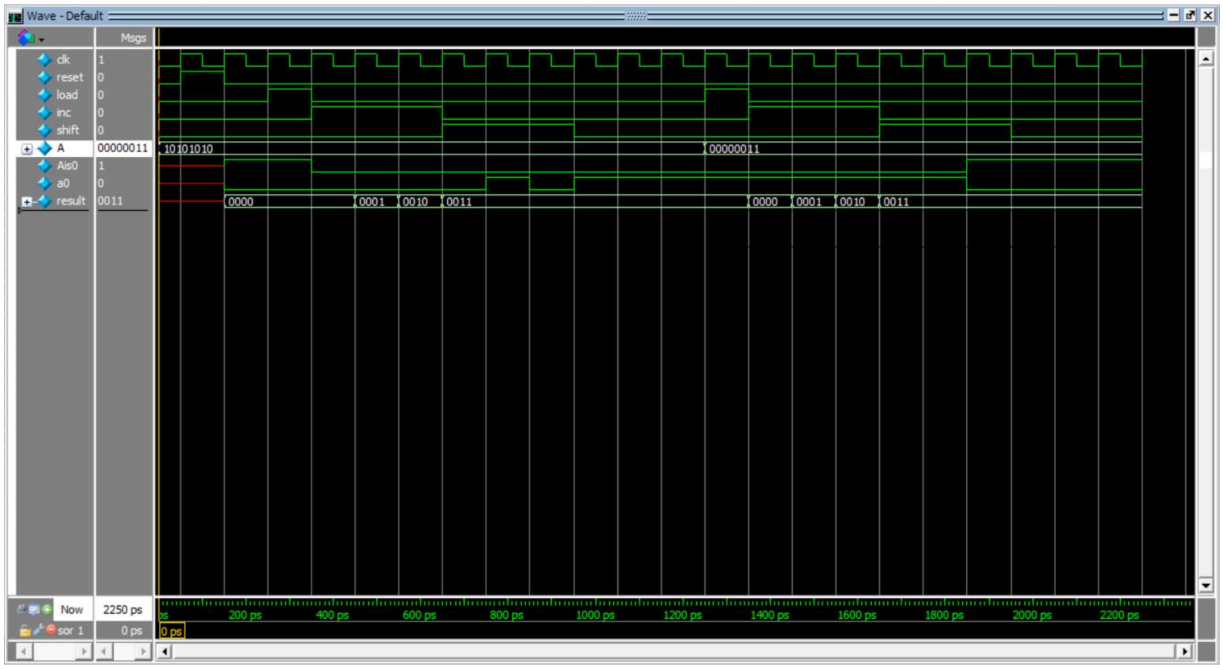

Fig. 5. Counter_datapath_testbench simulation waveform

We tested two different input values A that have different numbers of "1," and changed input reset, load, inc, shift. We expect to see that Ais0 is true before load is true, result increment for each cycle that inc is true with one cycle delay, a0 changes based on shift and A. From Fig. 5 we see that the output are as expected, so the module has correct functionality.

Then we implement the FSM for control, which produces the simulation below in Fig.6.

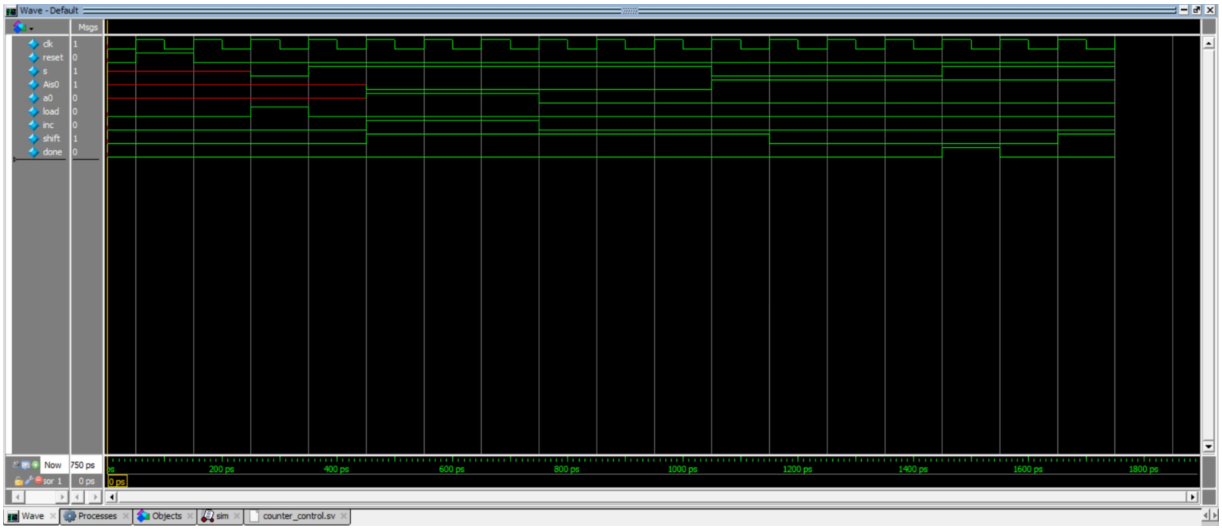

Fig. 6. Counter_control_testbench simulation waveform

We test the transitions between each state, and whether the output load, inc, shift, done to be updated based on the state of FSM. We expect to see that after reset has a pulse, load will have one cycle pulse, then, inc will be true for 3 cycles, and then when Ais0 turns true and s turns true, done is true. So from Fig. 6. We see that the outputs are in the shape expected. So the module should have the expected functionality.

Since we want to display the count in decimal form on HEX0, we wrote a module that converts the count from binary to a corresponding 7-segment display value. It produces the simulation below in Fig. 7.

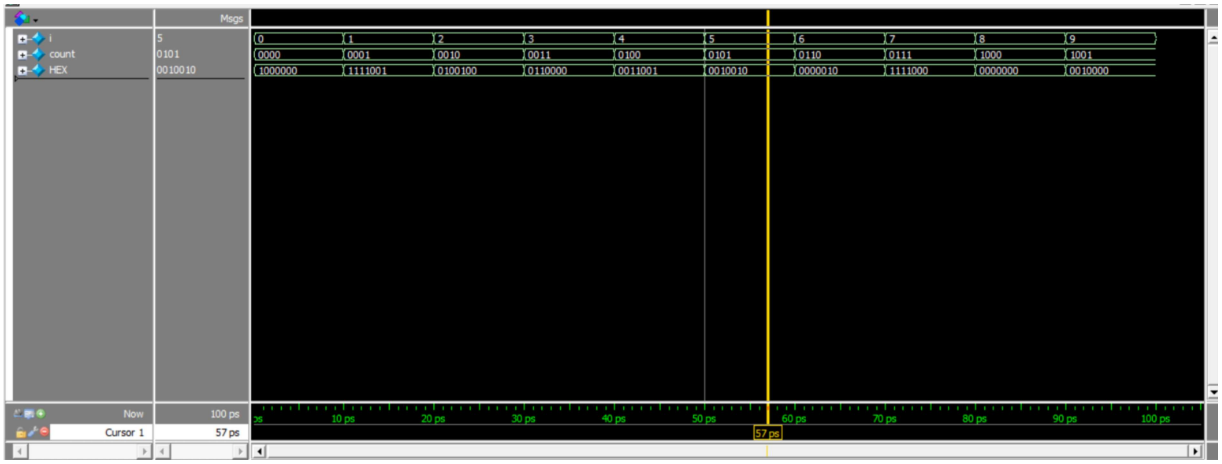

Fig. 7. seg_7_testbench simulation waveform

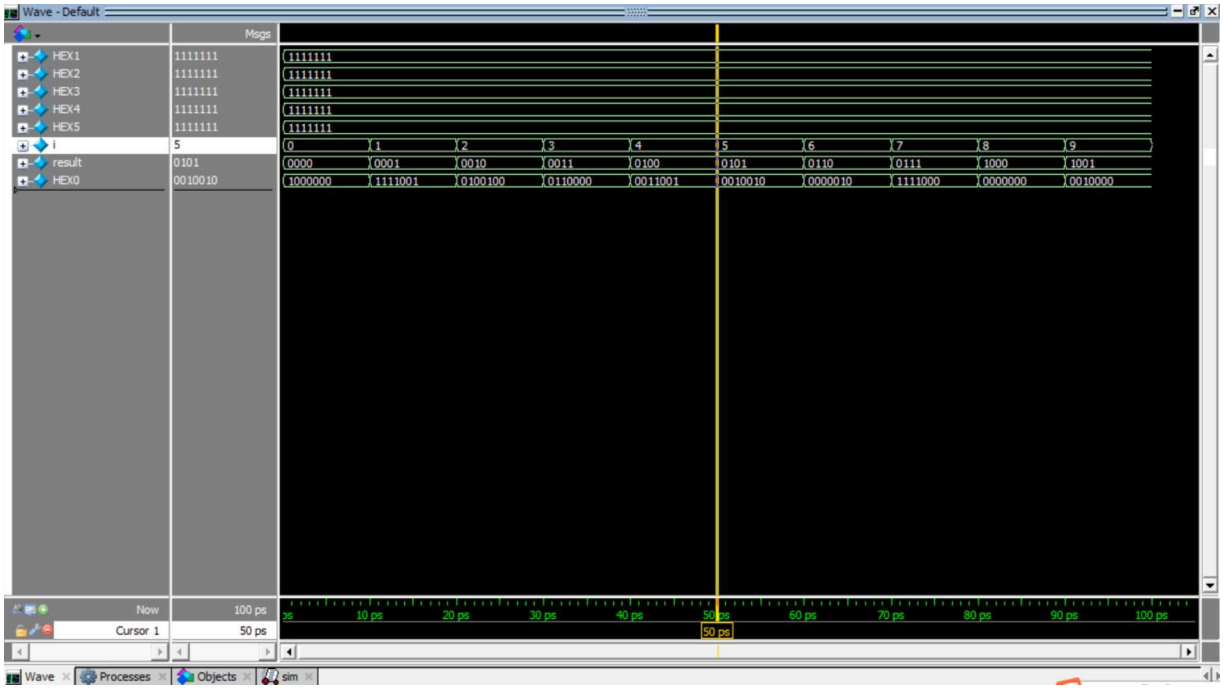

Fig. 8. counter_display_testbench simulation waveform

Seg_7 and counter_display test the same thing, except counter_display also turns off the HEX display for HEX5-1. We expect to see count/result increment from 0 to 9, and HEX/HEX0 shows the corresponding 7-segment display, HEX5-1 are 7'b1. From Fig.7, 8 we can see that we have expected output. So the module works correctly.

Since we are using a fast clock, we want to pass our input through 2 DFFs in order to prevent metastability. We implemented 2 DFF to process the input, which produced the simulation below in Fig. 9.

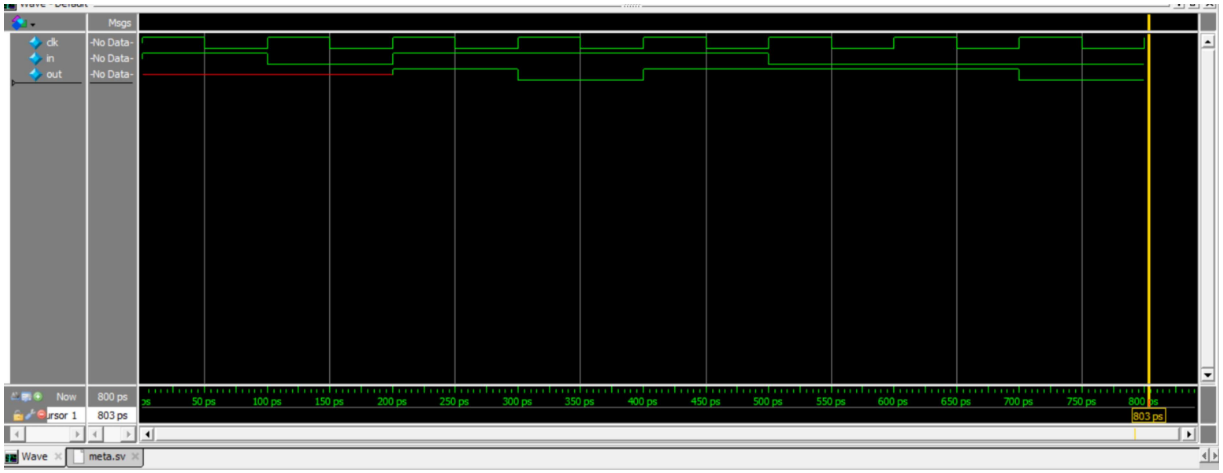

Fig. 9. meta_testbench simulation waveform

Since it simply uses 2 DFFs, we just test some changes in input, and expect to see the changes happen in the output two cycles later. From Fig. 9 we see that this module produces correct output as we expected.

Lastly, we wire everything up in the top module. The output produces the following waveform in Fig. 10.

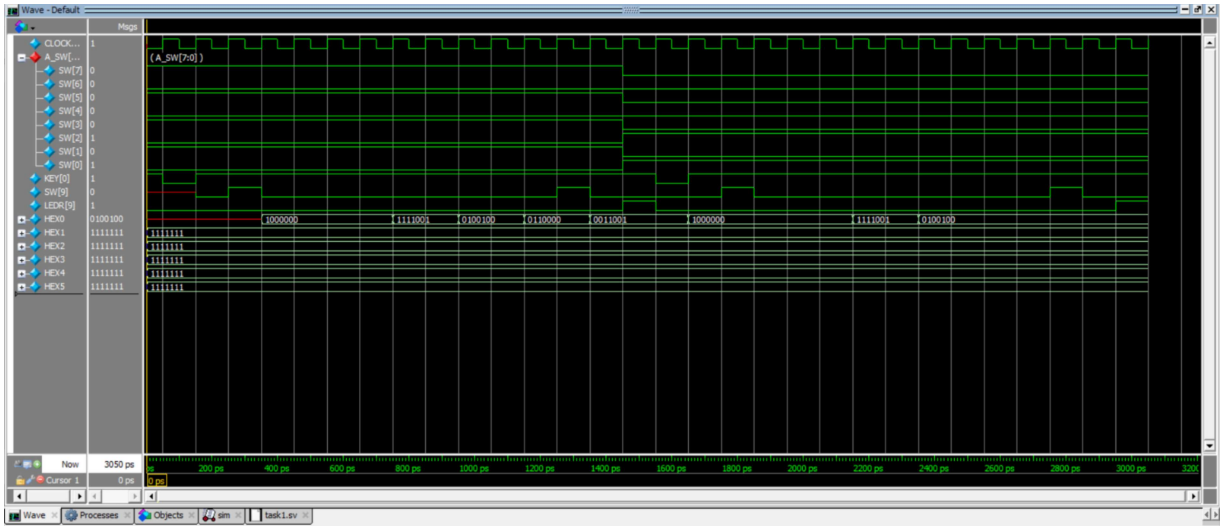

Fig. 10. task1_testbench simulation waveform

For this part, we test whether the reset (KEY[0]) and start signals (SW[9]) is functioning, whether HEX0 can display a proper count based on SW[7:0], and whether done (LEDR[9]) displays properly. We expect to see that after reset and s is initialized, HEX0 starts to have display and increment in the later 10 cycles when it sees a "1," and LEDR[9] turns true if the s is true at this point. Same behavior should happen for the second SW[7:0] input. From Fig. 10. we see that the expected waveform is produced and the module should function properly.

Below is the synthesis report for Task1.

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins
1	|task1	28 (0)	18 (0)	0	0	67	0
1	|counter_control:CTRL|	3 (3)	2 (2)	0	0	0	0
2	|counter_datapath:DATAPATH|	17 (17)	12 (12)	0	0	0	0
3	|counter_display:hex_display|	7 (0)	0 (0)	0	0	0	0
1	|seg_7:s1|	7 (7)	0 (0)	0	0	0	0
4	|meta:m1|	1 (1)	2 (2)	0	0	0	0
5	|meta:m2|	0 (0)	2 (2)	0	0	0	0

Fig. 9. Synthesis Report

Task 2:

For task 2, we first write the meta module which deals with metastability. It is the same as what we used in Task 1.

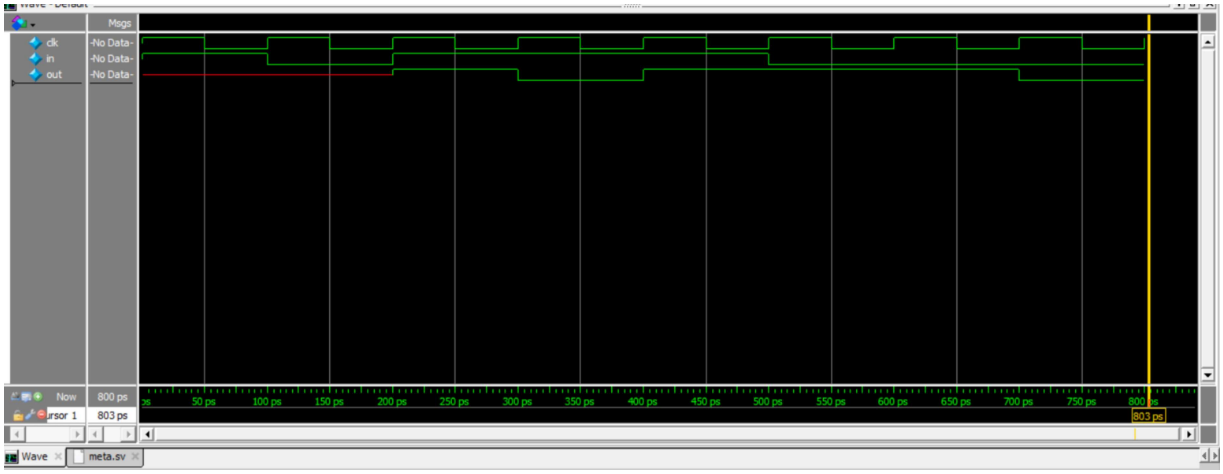

Fig.10. meta_testbench simulation waveform

Since it simply uses 2 DFFs, we just test some changes in input, and expect to see the changes happen in the output two cycles later. From Fig. 9 we see that this module produces correct output as we expected.

Next is the hex display module that displays hexadecimal numbers

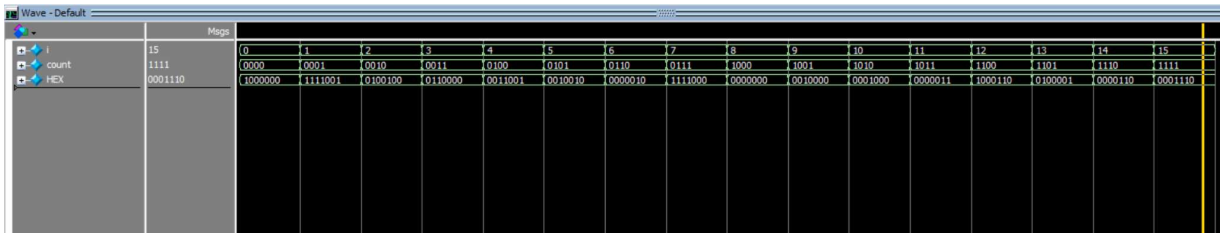

Fig.11. seg_7_testbench simulation waveform

We test all 16 possible input patterns. And we can see that the hex is displaying correctly.

Next is the binary search module and the top level simulation.

We tested two different values that corresponds to a address in the ram, and one that does not, and expect to see address/HEX, and f/nf/LEDR[9][8] update correctly. However, for some reason the addr and f is not updating on the modelsim but it does work on FPGA.

Conclusion:

Though some small problems exists, it does not affect the overall functionality of the program.

APPENDIX:

Task 1:

I. counter_datapath.sv

```
1   // Team: Cynthia Li, Simon Chen
2   // EE 371 Lab 4 Task 1
3
4   // This module is the datapath of the ASMD. It takes in control signals,
5   // reset, load, inc, shift, and a 8 bit value, A, which is the data to load.
6   // It outputs two status, whether local variable A is 0 and whether the
7   // current bit examined is 1, and a 4 bit value 'result' which is the count
8   // of the number of 1 in A.
9   module counter_datapath (clk, reset, A, load, inc, shift, Ais0, a0, result);
10     input logic clk, reset, load, inc, shift;
11     input logic [7:0] A;
12     output logic Ais0, a0;
13     output logic [3:0] result; // need 4-bit value since we want to display 0-8
14
15     // this defines the variable A_reg
16     logic [7:0] A_reg;
17
18     // the following specifies the datapath behavior
19     // it updates the output result and the variable A_reg
20     always_ff @(posedge clk) begin
21        if (reset) begin
22           result <= 4'b0;
23           A_reg <= 8'b0;
24        end
25        if (load) begin
26           result <= 4'b0;
27           A_reg <= A;
28        end
29        if (inc) begin
30           result <= result + 1;
31        end
32        if (shift) begin
33           A_reg <= A_reg >> 1;
34        end
35     end
36
37     // the following defines the output Ais0 and a0's behavior
38     assign Ais0 = (A_reg == 0);
39     assign a0 = A_reg[0];
40
41  endmodule
42
43  module counter_datapath_testbench();
44     logic clk, reset, load, inc, shift;
45     logic [7:0] A;
46     logic Ais0, a0;
47     logic [3:0] result;
48
```

```
49      parameter CLK_Period = 100;
50      initial begin
51          clk <= 1'b0;
52          forever #(CLK_Period/2) clk <= ~clk;
53      end
54
55      counter_datapath dut (.*);
56
57      initial begin
58          A <= 8'b10101010;
59          load <= 0; inc <= 0; shift <= 0;
60          reset <= 0;                              @(posedge clk);
61          reset <= 1;                              @(posedge clk);
62          reset <= 0;                              @(posedge clk);
63          load <= 1;                               @(posedge clk);
64          load <= 0; inc <= 1;          repeat(3) @(posedge clk);
65          inc <= 0; shift <= 1;         repeat(3) @(posedge clk);
66          shift <= 0;                   repeat(3) @(posedge clk);
67          A <= 8'b00000011;
68          load <= 1;                               @(posedge clk);
69          load <= 0; inc <= 1;          repeat(3) @(posedge clk);
70          inc <= 0; shift <= 1;         repeat(3) @(posedge clk);
71          shift <= 0;                   repeat(3) @(posedge clk);
72
73          $stop;
74      end
75  endmodule
```

II. counter_control.sv

```
1   // Team: Cynthia Li, Simon Chen
2   // EE 371 Lab 4 Task 1
3
4   // This module is the control of the ASMD. It takes in a start signal
5   // s, and two conditional variable Ais0 and a0, it outputs control signals,
6   // load, inc, shift, and done that represents the action needed for
7   // counting and the status of the counting action based on the conditions
8   // of s, Ais0, and a0.
9   module counter_control (clk, reset, s, Ais0, a0, load, inc, shift, done);
10      input logic clk, reset, s, Ais0, a0;
11      output logic load, inc, shift, done;
12
13      // the following specifies the states in FSM
14      enum {s1, s2, s3} ps, ns;
15
16      // the following specifies the next state logic
17      always_comb begin
18          case (ps)
19              s1: if (s)
20                      ns = s2;
21                  else // !s
22                      ns = s1;
23              s2: if (Ais0)
24                      ns = s3;
25                  else // !Ais0
26                      ns = s2;
27              s3: if (s)
28                      ns = s1;
29                  else // !s
30                      ns = s3;
31              default: ns = s1;
32          endcase
33      end
34
35      // the following specifies the output logic
36      always_comb begin
37          case (ps)
38              s1: begin
39                      inc = 1'b0;
40                      shift = 1'b0;
41                      done = 1'b0;
42                      if (!s) begin
43                          load = 1'b1;
44                      end else begin
45                          load = 1'b0;
46                      end
47                  end
```

```
48          s2: begin
49                  load = 1'b0;
50                  shift = 1'b1;
51                  done = 1'b0;
52                  if (!Ais0 & a0) begin
53                     inc = 1'b1;
54                  end else begin
55                     inc = 1'b0;
56                  end
57              end
58          s3: begin
59                  load = 1'b0;
60                  inc = 1'b0;
61                  shift = 1'b0;
62                  if (s) begin
63                     done = 1'b1;
64                  end else begin
65                     done = 1'b0;
66                  end
67              end
68          default: begin
69                     load = 1'b0;
70                     inc = 1'b0;
71                     shift = 1'b0;
72                     done = 1'b0;
73                  end
74       endcase
75    end
76
77    always_ff @(posedge clk) begin
78       if (reset) begin
79          ps <= s1;
80       end else begin
81          ps <= ns;
82       end
83    end
84 endmodule
85
86 module counter_control_testbench();
87    logic clk, reset, s, Ais0, a0;
88    logic load, inc, shift, done;
89
90    parameter CLK_Period = 100;
91    initial begin
92       clk <= 1'b0;
93       forever #(CLK_Period/2) clk <= ~clk;
94    end
95
96    counter_control dut (.*);
97
98    initial begin
99       reset <= 0;                              @(posedge clk);
100      reset <= 1;                              @(posedge clk);
101      reset <= 0;                              @(posedge clk);
102      s <= 0;                                  @(posedge clk);
103      s <= 1;                                  @(posedge clk);
104      Ais0 <= 0; a0 <= 1;          repeat(3) @(posedge clk);
105      a0 <= 0;                     repeat(3) @(posedge clk);
106      Ais0 <= 1; s <= 0;                       @(posedge clk);
107      s <= 0;                      repeat(3) @(posedge clk);
108      s <= 1;                      repeat(3) @(posedge clk);
109      $stop;
110   end
111 endmodule
```

III. counter_display.sv

```
1  // Team: Cynthia Li, Simon Chen
2  // EE 371 Lab 4 Task 1
3
4  // This module controls the HEX display of the FPGA board. It takes
5  // in a 4 bit value representing the result of the counting action,
6  // and display the value's decimal form on HEX0; it turns off the
7  // the rest of the displays
8  module counter_display (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, result);
9     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
10    input logic [3:0] result;
11
12    // Default values, turns off the HEX1-HEX5's displays
13    assign HEX1 = 7'b1111111;
14    assign HEX2 = 7'b1111111;
15    assign HEX3 = 7'b1111111;
16    assign HEX4 = 7'b1111111;
17    assign HEX5 = 7'b1111111;
18
19    // Store the 7-bit status of input 4-bit signal
20    seg_7 s1(.count(result), .HEX(HEX0));
21 endmodule
22
23 // This module controls one HEX display on the FPGA board. It takes in
24 // a 4 bit input signal and output a 7 bit value that represents its
25 // corresponding 7 segment display.
26 module seg_7 (count, HEX);
27    input logic [3:0] count;
28    output logic [6:0] HEX;
29
30    // the following specifies each input value's corresponding hex
31    // display value
32    always_comb begin
33       case (count)
34                   // Light: 6543210
35          4'b0000: HEX = 7'b1000000; // 0
36          4'b0001: HEX = 7'b1111001; // 1
37          4'b0010: HEX = 7'b0100100; // 2
38          4'b0011: HEX = 7'b0110000; // 3
39          4'b0100: HEX = 7'b0011001; // 4
40          4'b0101: HEX = 7'b0010010; // 5
41          4'b0110: HEX = 7'b0000010; // 6
42          4'b0111: HEX = 7'b1111000; // 7
43          4'b1000: HEX = 7'b0000000; // 8
44          4'b1001: HEX = 7'b0010000; // 9
45          default: HEX = 7'bX;
46       endcase
47    end
48 endmodule
49
```

```
50  module seg_7_testbench();
51     logic [6:0] HEX;
52     logic [3:0] count;
53
54     seg_7 dut (.*);
55
56     // Try all combinations of input result
57     integer i;
58     initial begin
59        for(i = 0; i < 10; i++) begin
60           count = i;  #10;
61        end
62     end
63  endmodule
64
65  module counter_display_testbench();
66     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
67     logic [3:0] result;
68
69     counter_display dut (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, result);
70
71     // Try all combinations of input result
72     integer i;
73     initial begin
74        for(i = 0; i < 10; i++) begin
75           result = i; #10;
76        end
77     end
78  endmodule
```

IV. meta.sv

```
1   // Team: Cynthia Li, Simon Chen
2   // EE 371 Lab 4 Task 1
3
4   // This module takes an input, pass it through
5   // two DFF to prevent metastability.
6   module meta (clk, in, out);
7      input logic clk, in;
8      output logic out;
9      logic temp;
10
11     // the following passes in through 2 DFF
12     always_ff @(posedge clk)
13           temp <= in;
14     always_ff @(posedge clk)
15           out <= temp;
16  endmodule
17
18  module meta_testbench();
19     logic clk, in;
20     logic out;
21
22     // Set up the clock.
23     parameter CLOCK_PERIOD=100;
24     initial clk=1;
25     always begin
26        #(CLOCK_PERIOD/2);
27        clk = ~clk;
28     end
29
30     meta dut (.*);
31
32     initial begin
33     in <= 1; @(posedge clk);
34     in <= 0; @(posedge clk);
35     in <= 1; repeat(3)@(posedge clk);
36     in <= 0; repeat(3)@(posedge clk);
37     $stop; // End the simulation.
38     end
39  endmodule
```

V. task1.sv

```
// Team: Cynthia Li, Simon Chen
// EE 371 Lab 4 Task 1

// This module is the top level module that controls the bit counter's behavior on the FPGA
// board. It takes in 8 bit intput value from SW[7:0] that represent data to load, KEY[0] for
// reset, SW[9] for start signal, and CLOCK_50 for the counting operation. It outputs a 7 bit
// value to display the count on HEX0 and turns off the rest of HEX display. It also output
// the status of the counting operation on LEDR[9].
module task1 (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
   input logic CLOCK_50; // 50MHz clock.
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;
   input logic [3:0] KEY; // True when not pressed, False when pressed
   input logic [9:0] SW;

   // the following defines the variables needed
   logic [7:0] A;
   logic reset, s;
   logic [3:0] result;
   logic Ais0, a0, load, inc, shift;

   // the following specifies the input
   assign A = SW[7:0];

   // the following instantiates meta to process input to prevent metastability due to fast clock
   meta m1 (.clk(CLOCK_50), .in(~KEY[0]), .out(reset));
   meta m2 (.clk(CLOCK_50), .in(SW[9]), .out(s));

   // the following instantiates the control of bit counter
   counter_control CTRL (.clk(CLOCK_50), .reset, .s, .Ais0, .a0, .load, .inc, .shift, .done(LEDR[9]));
   // the following instantiates the datapath of bit counter
   counter_datapath DATAPATH (.clk(CLOCK_50), .reset, .A, .load, .inc, .shift, .Ais0, .a0, .result);
   // the following instantiates the HEX display for the bit counter
   counter_display hex_display (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .result);
endmodule
```

```
module task1_testbench();
   logic CLOCK_50; // 50MHz clock.
   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic [9:0] LEDR;
   logic [3:0] KEY; // True when not pressed, False when pressed
   logic [9:0] SW;

   parameter CLK_Period = 100;
   initial begin
      CLOCK_50 <= 1'b0;
      forever #(CLK_Period/2) CLOCK_50 <= ~CLOCK_50;
   end

   logic clk;
   assign clk = CLOCK_50;

   task1 dut (.*);

   initial begin
      SW[7:0] <= 8'b10101010;
      KEY[0] <= 1;                              @(posedge clk);
      KEY[0] <= 0;                              @(posedge clk);
      KEY[0] <= 1; SW[9] <= 0;                  @(posedge clk);
      SW[9] <= 1;                  repeat(1) @(posedge clk);
      SW[9] <= 0;                  repeat(9) @(posedge clk);
      SW[9] <= 1;                  repeat(1) @(posedge clk);
      SW[9] <= 0;                  repeat(1) @(posedge clk);

      SW[7:0] <= 8'b00000101;
      KEY[0] <= 1;                              @(posedge clk);
      KEY[0] <= 0;                              @(posedge clk);
      KEY[0] <= 1;                              @(posedge clk);
      SW[9] <= 1;                  repeat(1) @(posedge clk);
      SW[9] <= 0;                  repeat(9) @(posedge clk);
      SW[9] <= 1;                  repeat(1) @(posedge clk);
      SW[9] <= 0;                  repeat(2) @(posedge clk);
      $stop;
   end
endmodule
```

Task 2:

I. task2.sv

```
1   // Team: Cynthia Li, Simon Chen
2   // EE 371 Lab 4 Task 2
3
4   // Top-level module that defines the I/Os for the DE-1 SoC board. It takes a n input clock
5   // of 50mhz clock, a 10-bit Sw, and 4-bit KEY as inputs, and output 6 7-bit hex displays.
6
7   `timescale 1 ps / 1 ps
8   module task2 (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
9      input logic CLOCK_50; // 50MHz clock.
10     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
11     output logic [9:0] LEDR;
12     input logic [3:0] KEY; // True when not pressed, False when pressed
13     input logic [9:0] SW;
14
15     // turn of hex 2 to hex 5
16     assign HEX2 = 7'b1111111;
17     assign HEX3 = 7'b1111111;
18     assign HEX4 = 7'b1111111;
19     assign HEX5 = 7'b1111111;
20
21     logic [4:0] addr;
22     logic f, nf;
23     logic [6:0] HEX0_on, HEX1_on;
24
25     assign LEDR[9] = f;
26     assign LEDR[8] = nf;
27
28     // Instantiation of module meta. Takes in CLOCK_50, ~KEY[0],
29     // as inputs, and outputs key
30     meta m1 (.clk(CLOCK_50), .in(~KEY[0]), .out(key));
31
32     // Instantiation of module binary_search. Inputs: clk, reset, start, A.
33     // Outputs: f, nf, addr
34     binary_search b1 (.clk(CLOCK_50), .reset(key), .start(SW[9]), .A(SW[7:0]),
35                       .f, .nf, .addr);
36
37     // Instantiation of module seg_7. Takes in addr[3:0] as inputs, and
38     // outputs HEX0_on
39     seg_7 s0 (.count(addr[3:0]), .HEX(HEX0_on));
40
41     // Instantiation of module seg_7. Takes in addr[4] as inputs, and
42     // outputs HEX1_on
43     seg_7 s1 (.count(addr[4]), .HEX(HEX1_on));
44
45     // turn on hex0 and hex1 only when A is found
46     always_comb begin
47        if (f) begin
```

```
48          HEX0 = HEX0_on;
49          HEX1 = HEX1_on;
50       end else begin
51          HEX0 = 7'b1111111;
52          HEX1 = 7'b1111111;
53       end
54    end
55 endmodule
56
57 `timescale 1 ps / 1 ps
58 module task2_testbench();
59    logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
60    logic [9:0]  LEDR;
61    logic [3:0]  KEY;
62    logic [9:0]  SW;
63    logic CLOCK_50;
64
65    parameter CLK_Period = 100;
66    initial begin
67       CLOCK_50 <= 1'b0;
68       forever #(CLK_Period/2) CLOCK_50 <= ~CLOCK_50;
69    end
70
71    logic clk;
72    assign clk = CLOCK_50;
73
74    task2 dut (.*);
75
76    initial begin
77       SW[7:0] <= 8'b10101010;
78       KEY[0] <= 1;                          @(posedge clk);
79       KEY[0] <= 0;                          @(posedge clk);
80       KEY[0] <= 1; SW[9] <= 0;              @(posedge clk);
81       SW[9] <= 1;                repeat(1) @(posedge clk);
82       SW[9] <= 0;                repeat(9) @(posedge clk);
83       SW[9] <= 1;                repeat(1) @(posedge clk);
84       SW[9] <= 0;                repeat(1) @(posedge clk);
85
86       SW[7:0] <= 8'b00000101;
87       KEY[0] <= 1;                          @(posedge clk);
88       KEY[0] <= 0;                          @(posedge clk);
89       KEY[0] <= 1;                          @(posedge clk);
90       SW[9] <= 1;                repeat(1) @(posedge clk);
91       SW[9] <= 0;                repeat(9) @(posedge clk);
92       SW[9] <= 1;                repeat(1) @(posedge clk);
93       SW[9] <= 0;                repeat(2) @(posedge clk);
94       $stop;
95       end
96 endmodule
```

II. Binary_search.sv

```
1  // Team: Cynthia Li, Simon Chen
2  // EE 371 Lab 4 Task 2
3
4  // This module implements the binary search that can be used for a sorted ram.
5  // It takes in a clock, 1-bit start and reset signal, and a 8-bit A as inputs.
6  // It outputs 1-bit f and nf signal, and a 5-bit addr.
7
8  `timescale 1 ps / 1 ps
9  module binary_search (
10    input logic start,
11    input logic clk,
12    input logic reset,
13    input logic [7:0] A,
14    output logic f,
15    output logic nf,
16    output logic [4:0] addr,
17    output logic [2:0] ps1
18    );
19
20    // the following are pointers we use to track the addresses
21    logic [5:0] mid, low, up;
22    // the following are holders for the data to compare
23    logic [7:0] data, A_reg;
24
25    // Instantiation of module ram32x8. Input: address, clock, data, wren. Output: q.
26    ram32x8 ram (.address(mid), .clock(clk), .data(), .wren(1'b0), .q(data));
27
28    // Controller logic
29    enum {s_idle, s_wait1, s_wait2, s_compare, s_down, s_up, s_find, s_done} ps, ns;
30
31    always_comb begin
32       case (ps)
33          s_idle: ns = start ? s_wait1 : s_idle;
34
35          s_wait1: ns = s_wait2;
36
37          s_wait2: ns = s_compare;
38
39          s_compare:
40                if (A_reg == data) begin
41                   ns = s_done;
42                end else if (A_reg > data) begin // if A is bigger, search up
43                   ns = s_up;
44                end else begin // if A is smaller, we search down
45                   ns = s_down;
46                end
47
```

```
48          s_up: ns = s_find;
49
50          s_down: ns = s_find;
51
52          s_find: ns = (low > up) ? s_done : s_wait1;
53
54          s_done: ns = start ? s_done : s_idle;
55      endcase
56  end
57
58  // control output logic
59  always_ff @(posedge clk) begin
60      if (reset) begin
61          mid <= 5'b01111;
62          low <= 5'b00000;
63          up <= 5'b11111;
64          A_reg <= A;
65      end else if (ps == s_idle) begin
66          mid <= 5'b01111;
67          low <= 5'b00000;
68          up <= 5'b11111;
69          A_reg <= A;
70      end else if (ps == s_wait1) begin
71          mid <= (low + up) / 2;
72      end else if (ps == s_up) begin
73          low <= mid + 1;
74      end else if (ps == s_down) begin
75          up <= mid - 1;
76      end
77  end
78
79  // DFF
80  always_ff @(posedge clk) begin
81      if (reset)
82          ps <= s_idle;
83      else
84          ps <= ns;
85  end
86
87  // datapath logic
88  always_comb begin
89      if(f) begin
90          addr = mid;
91      end else begin
92          addr = 5'bx;
93      end
94  end
95
```

```
95
96      // the following outputs whether the given data A can be found in the ram
97      assign f= (data == A_reg);
98      assign nf = (data != A_reg) && (ps == s_done);
99      assign ps1 = ps;
100
101   endmodule
102
103   `timescale 1 ps / 1 ps
104
105   module binary_search_testbench();
106    logic clk, reset, start, f, nf;
107    logic [4:0] addr;
108    logic [7:0] A;
109    logic [2:0] ps1;
110
111    binary_search dut (.*);
112
113    // Setting up a simulated clock.
114    parameter CLOCK_PERIOD = 100;
115    initial begin
116     clk <= 0;
117     forever #(CLOCK_PERIOD/2) clk <= ~clk; // Forever toggle the clock
118    end
119
120    initial begin
121                                                  @(posedge clk);
122     reset <= 1;                                     @(posedge clk);
123     reset <= 0;
124     start <= 0; A <= 19;                  repeat(2) @(posedge clk); // location 19
125     start <= 1;                           repeat(8) @(posedge clk);
126     start <= 0; A <= 2;                   repeat(2) @(posedge clk); // location 02
127     start <= 1;                           repeat(8) @(posedge clk);
128     start <= 0; A <= 45;                  repeat(2) @(posedge clk); // not found
129     start <= 1;                           repeat(8) @(posedge clk);
130
131     $stop;
132    end
133   endmodule
```

III. seg_7.sv

```
1   // Team: Cynthia Li, Simon Chen
2   // EE 371 Lab 4 Task 2
3
4   // This module controls one HEX display on the FPGA board. It takes in
5   // a 4 bit input signal and output a 7 bit value that represents its
6   // corresponding 7 segment display.
7   module seg_7 (HEX, count);
8      output logic [6:0] HEX;
9      input logic [3:0] count;
10
11     // the following specifies each input value's corresponding hex
12     // display value
13     always_comb begin
14        case (count)
15                    // Light: 6543210
16           4'b0000: HEX = 7'b1000000; // 0
17           4'b0001: HEX = 7'b1111001; // 1
18           4'b0010: HEX = 7'b0100100; // 2
19           4'b0011: HEX = 7'b0110000; // 3
20           4'b0100: HEX = 7'b0011001; // 4
21           4'b0101: HEX = 7'b0010010; // 5
22           4'b0110: HEX = 7'b0000010; // 6
23           4'b0111: HEX = 7'b1111000; // 7
24           4'b1000: HEX = 7'b0000000; // 8
25           4'b1001: HEX = 7'b0010000; // 9
26           4'b1010: HEX = 7'b0001000; // A
27           4'b1011: HEX = 7'b0000011; // b
28           4'b1100: HEX = 7'b1000110; // C
29           4'b1101: HEX = 7'b0100001; // d
30           4'b1110: HEX = 7'b0000110; // E
31           4'b1111: HEX = 7'b0001110; // F
32           default: HEX = 7'bX;
33        endcase
34     end
35  endmodule
```

```
36
37  module seg_7_testbench();
38     logic [6:0] HEX;
39     logic [3:0] count;
40
41     seg_7 dut (.HEX, .count);
42
43     // Try all combinations of input result
44     integer i;
45     initial begin
46        for(i = 0; i < 16; i++) begin
47           count = i;  #10;
48        end
49     end
50  endmodule
```