UW Student: Brian Chen

EE 371

February 19, 2025

Lab4 Report

I. Procedure

Task 1: Bit-Counting Circuit

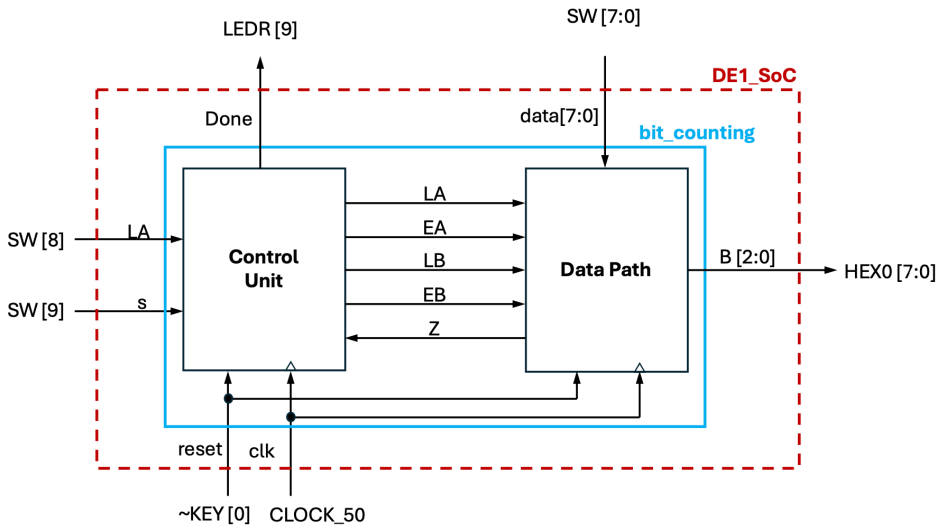

Figure 1: Block diagram of the "DE1_SoC" module of Task 1.

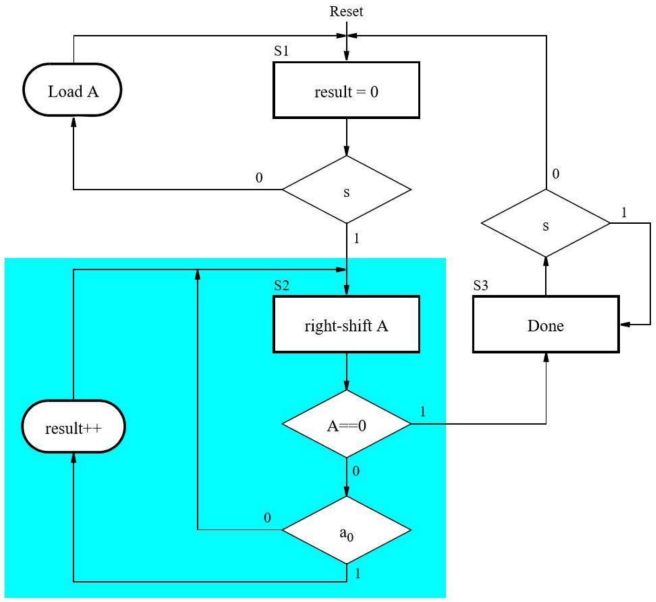

Figure 2: ASMD chart of "bitcounting" module of Task 1.

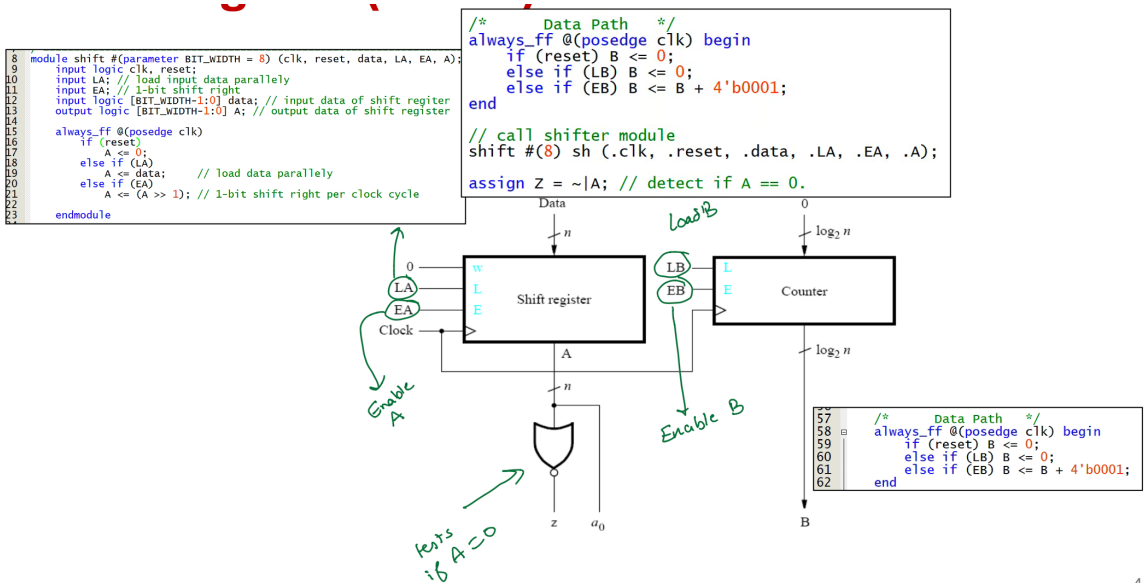

Figure 3: Block diagram of the Data Path portion in "bitcounting" module of Task 1. (This picture is from Prof. Hussein's lecture slide.)

Figure 1 shows the block diagram of the design in Task 1. This diagram contains two portions which are the "Control Unit" and the "Data Path". I designed the Control Unit and the Data Path by writing the SystemVerilog codes in the "bitcounting" module according to the ASMD chart as shown in Figure 2 provided in the handout of LAB4. Figure 3 shows the block diagram of the Data Path. The ASM has three states (S_1, S_2, S_3) and four decision-making nodes which check the logic values of "s", "A == 0 (i.e., Z ==1)" and the A[0], respectively. The Data Path includes two components: one is an 8-bit shift register with the 8-bit output signal A[7:0] which stores the 8-bit data to be counted the number of 1's; the other one is a 4-bit counter with the 4-bit output signal B[4:0] which shows the number of 1's in A[7:0]. The load operation and the shift-right operation of the 8-bit shift register are enabled by the signals "LA" and "EA", respectively. The 4-bit counter is enabled to increase by 1 by the signal "EB". In Task 1, the inputs and outputs of DE1_SoC are arranged as follows. KEY[0] is for the reset signal. SW[7:0] are for specifying the 8-bit data A[7:0] to be counted the number of 1's. SW[8] is for the LA signal which controls the 8-bit shift register to load the bits specified by SW[7:0]. SW[9] is for the start signal. LEDR[9] is for indicating if the process is done. HEX0 is for displaying the number of 1's in A[7:0]. After writing the SystemVerilog codes, I simulated this design on Modelsim-Altera and verifying the functions on LabsLand.

Task 2: Binary Search Circuit

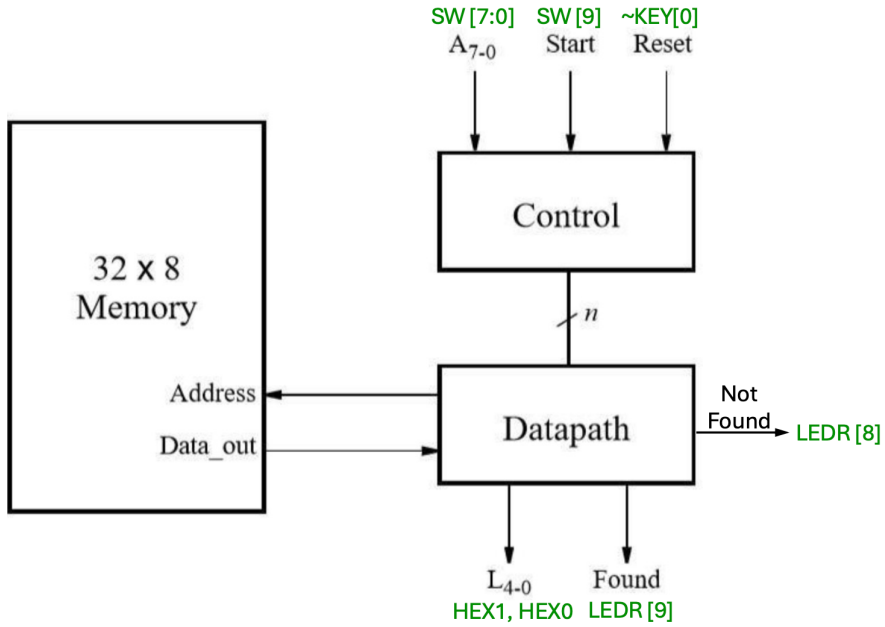

Figure 4: Block diagram (provided in the handout of LAB4) of Task 2.

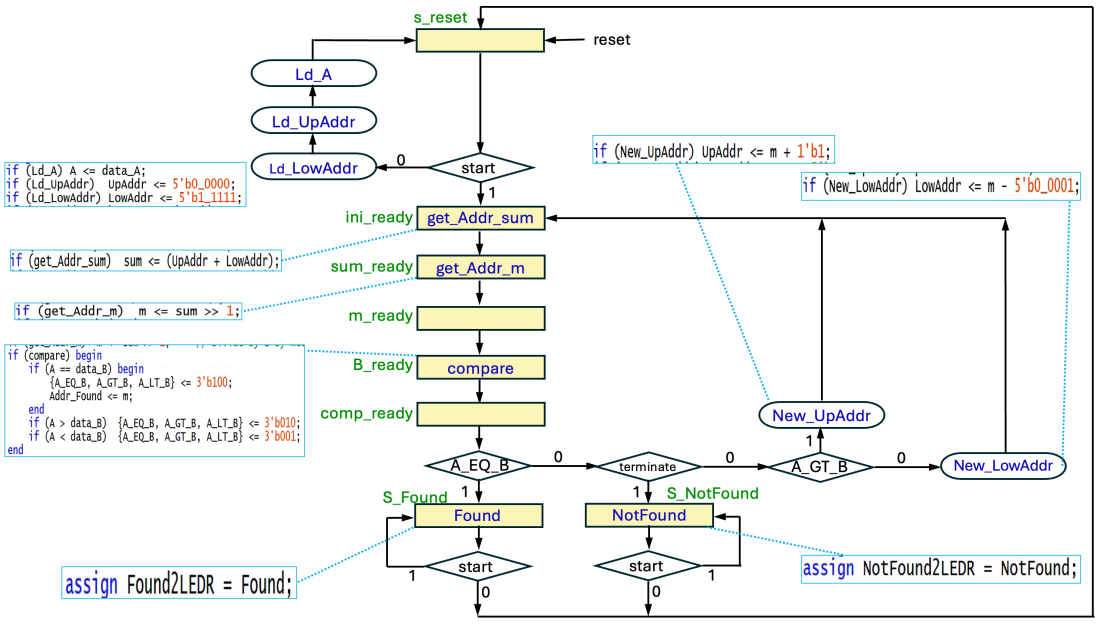

Figure 5: ASMD chart of "binsearch" module of Task 2.

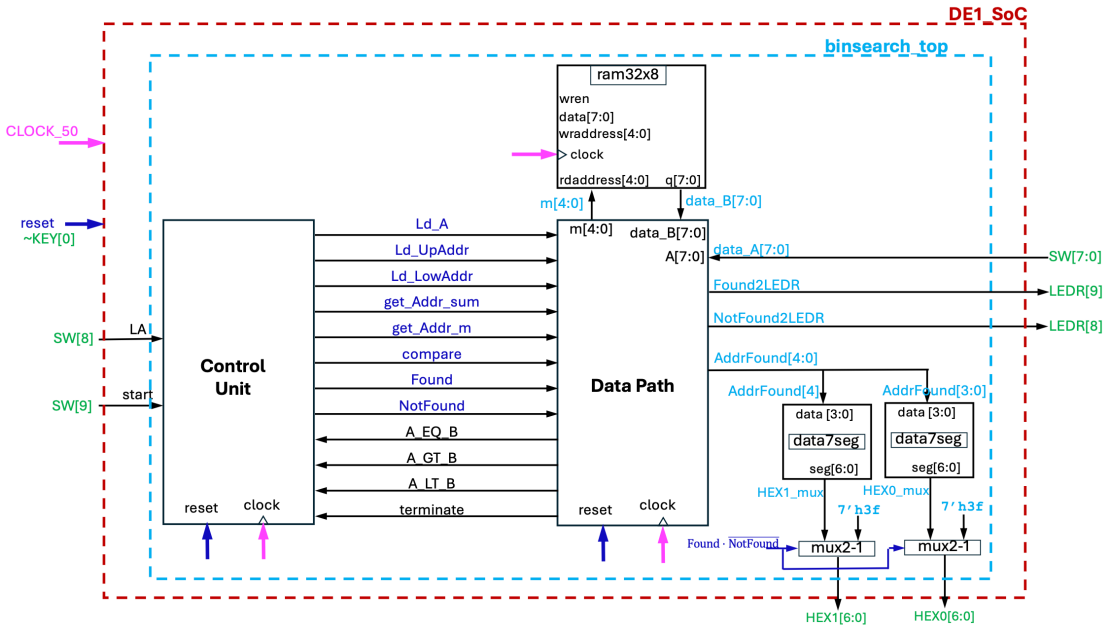

Figure 6: Block diagram of "DE1_SoC" and "binsearch_top" modules of Task 2.

Figure 4 shows the block diagram provided in the handout of the design in Task 2. I derived the ASMD chart as shown in Figure 5. There are 8 states (s_reset, ini_ready, sum_ready, m_ready, B_ready, comp_ready, S_Found, and S_NotFound), 5 output signals (Ld_A, Ld_UpAddr, Ld_LowAddr, New_UpAddr, and New_LowAddr), and 4 input signals (start, A_EQ_B, A_GT_B, A_LT_B, and terminate) in this ASMD chart. Figure 6 is the block diagram of the "DE1_SoC" module and the "binsearch_top" module corresponding to the ASMD chart. As shown in Figure 5, the "binsearch_top" module contains four portions: the Controller Unit, the Data Path, the RAM (ram32x8), and two "data7seg" modules. The Control Unit generates 8 output signals (as printed in blue) to direct the Data Path how the data are processed. The Control Unit also receives 4 input signals sent from the Data Path to do the decision making. The RAM stores 32 words, each word has 8 bits. The two "data7seg" modules convert the 5-bit address AddrFound[4:0] to the signals HEX1[6:0] and HEX0[6:0] for driving the corresponding 7-segment displays to show the address in hexadecimal. I created the files "my_array.mif" to store the data in RAM. The contents of RAM specified in "my_array.mif" are as shown in Figure 7.

```
ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;

CONTENT BEGIN
    00  :   00000001;
    01  :   00000010;
    02  :   00000100;
    03  :   00000110;
    04  :   00001000;
    05  :   00001010;
    06  :   00001100;
    07  :   00001110;
    08  :   00010000;
    09  :   00010010;
    0A  :   00010100;
    0B  :   00010110;
    0C  :   00011000;
    0D  :   00011010;
    0E  :   00011100;
    0F  :   00011110;
    10  :   00100000;
    11  :   00100010;
    12  :   00100100;
    13  :   00100110;
    14  :   00101000;
    15  :   00101010;
    16  :   00101100;
    17  :   00101110;
    18  :   00110000;
    19  :   00110010;
    1A  :   00110100;
    1B  :   00110110;
    1C  :   00111000;
    1D  :   00111010;
    1E  :   00111100;
    1F  :   00111110;
END;
```

Figure 7: The contents of RAM specified in “my_array.mif” (ram32x8).

The Data Path includes four registers. The first register is an 8-bit register with the 8-bit output signal A[7:0] which stores the 8-bit data to be sought if it is in the RAM. The second register is an 8-bit register with the 8-bit output signal B[7:0] which stores the 8-bit data read from the address m[4:0] in the RAM where m[4:0] is the mean of the upper-bound address UpAddr[4:0] and the lower-bound address LowAddr[4:0] of the interval to be sought at each iteration. In order to prevent the occurrence of overflow, the third register is an 6-bit register which stores the 6-bit sum sum[5:0] of UpAddr[4:0] and LowAddr[4:0]. The fouth register is an 5-bit register which stores the 5-bit mean m[4:0] which is obtained by shifting the 6-bit sum[5:0] right by 1 bit to perform the operation of divided by 2. In Task 2, the inputs and outputs of DE1_SoC are arranged as follows. KEY[0] is for the reset signal. SW[7:0] are for specifying the 8-bit data A[7:0] to be sought if it is in the RAM. SW[8] is for the LA signal which controls the 8-bit shift register to load the bits specified by SW[7:0]. SW[9] is for the start signal. LEDR[9] is lighted if the data is found in RAM. LEDR[8] is lighted if the data is not found in RAM. HEX1 and HEX0 are placed for displaying the address where the data is sought in the RAM. After writing the SystemVerilog codes, I simulated this design on Modelsim-Altera and verifying the functions on LabsLand.

II. Result

Task 1: Bit-Counting Circuit

Simulation Result with the Testbench for "DE1_SoC.sv" and Other Modules:

I wrote the testbench to test the top module in Task 1. In this testbench, there is a clock generator. First, I reset this design by pressing the push button (KEY[0]). Then, I applied the 8-bit pattern 10101110 that will be done the bit-counting process by setting the sliding switches (SW[7:0]). I loaded the pattern as mentioned into the 8-bit shift register by turning the sliding switch (SW[8]) on, and tuned off. Finally, I turned the sliding switch (SW[9]) to start. Figures 8 and 9 show the simulation results of the DE1_SoC module and the bitcounting module, respectively. The output of the 4-bit counter is 4'd5 that is the number of 1's in the pattern 10101110. The simulation results show that this design performs well. When I uploaded this design to the LabsLand, I discovered that the 50-MHz clock signal provided by DE1_SoC seems too fast. I placed the clock_divider module on this design to slow down the clock frequency. According to my experiments, I set the clock frequency to $50 \times \frac{10^6}{2^6}$ Hz, that is 390 kHz. During the verification on LabsLand, this design performs well. Figures 10 to 12 show the simulation results of the clock_divider module, the dada7seg module, and the shift module, respectively.

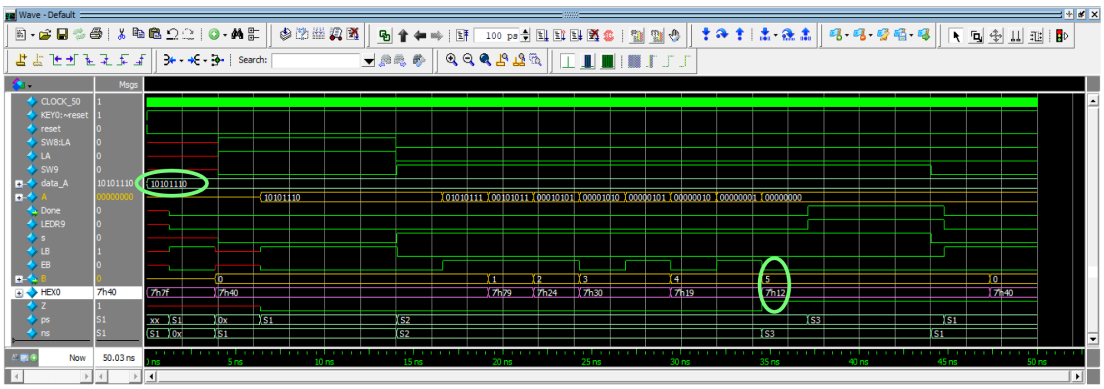

Figure 8: Simulation results of the design "DE1_SoC" in Task 1.

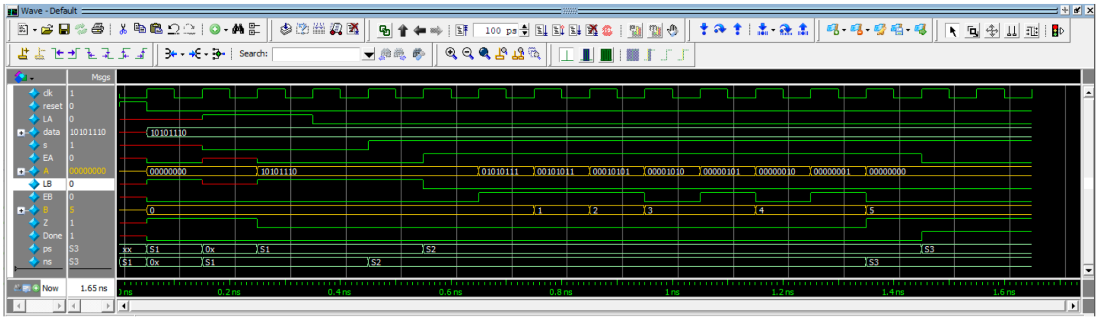

Figure 9: Simulation results of the design “bitcounting” in Task 1.

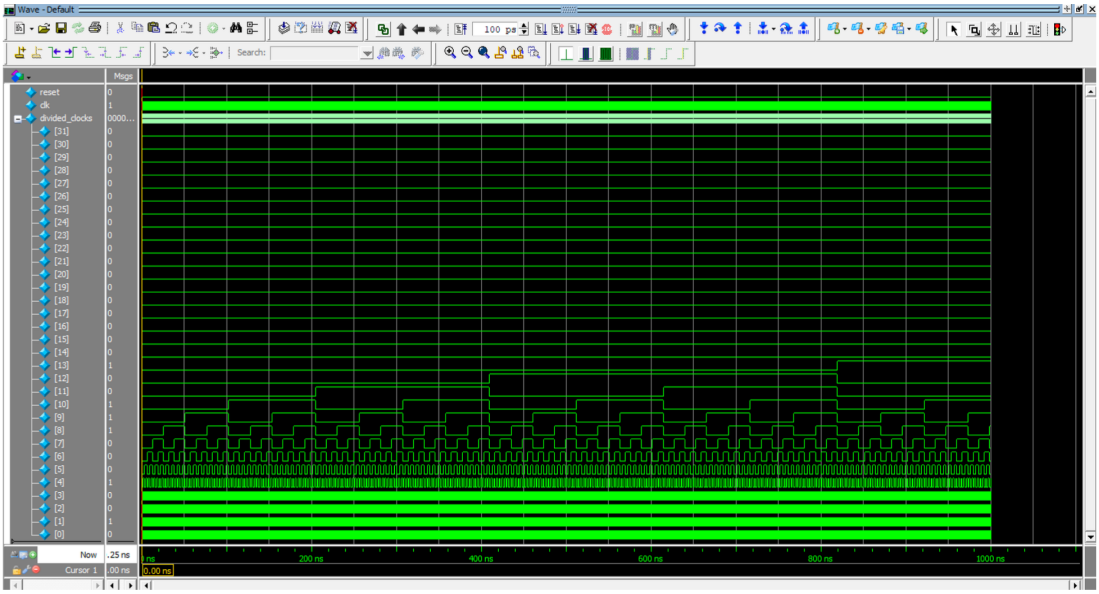

Figure 10: Simulation results of the design “clock_divider” in Task 1.

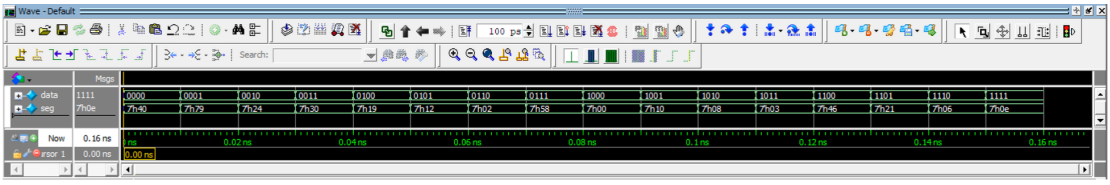

Figure 11: Simulation results of the design “data7seg” in Task 1.

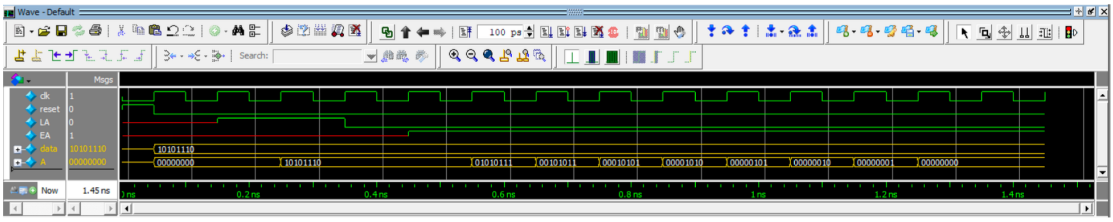

Figure 12: Simulation results of the design “shift” in Task 1.

Task 2: Binary Search Circuit

Simulation Result with the Testbench for "DE1_SoC.sv" and the binsearch_top Module:

I wrote the testbench to test the module in Task 2. First, I reset this design by pressing the push button (KEY[0]). Then, I applied the 8-bit pattern 8'b00001100 (8'd12) that will be sought if it is in the RAM by setting the sliding switches (SW[7:0]). I loaded the pattern into the 8-bit register by turning the sliding switch (SW[8]) on, and then tuned off. Finally, I turned the sliding switch (SW[9]) to start. Figures 13 and 14 show the simulation results of the DE1_SoC module and the bitcounting module, respectively. As we can see in the simulation results, the 8-bit pattern 8'b00001100 (8'd12) was found in the RAM at the address 5'd6 at the fifth step. I applied the second pattern 8'd62 to test this design. As we can see in the simulation results, the 8-bit pattern 8'd62 was not found in the RAM. I applied the third pattern 8'd38 to test this design. As we can see in the simulation results, the 8-bit pattern 8'd38 was found in the RAM at the address 5'd19 at the third step. Finally, I applied the fourth pattern 8'd14 to test this design. As we can see in the simulation results, the 8-bit pattern 8'd14 was found in the RAM at the address 5'd7 at the second step. As a whole, the simulation results show that this design performs well. During the verification on LabsLand, this design also performs well.

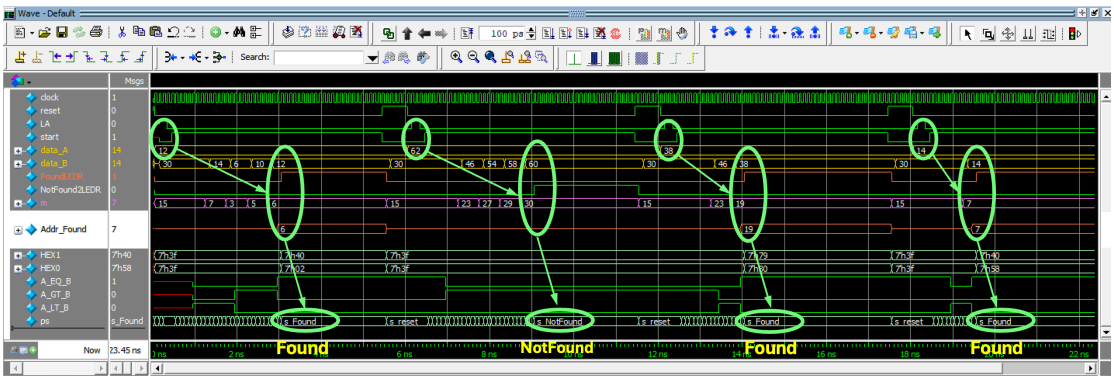

Figure 13: Simulation results of "DE1_SoC" in Task 2.

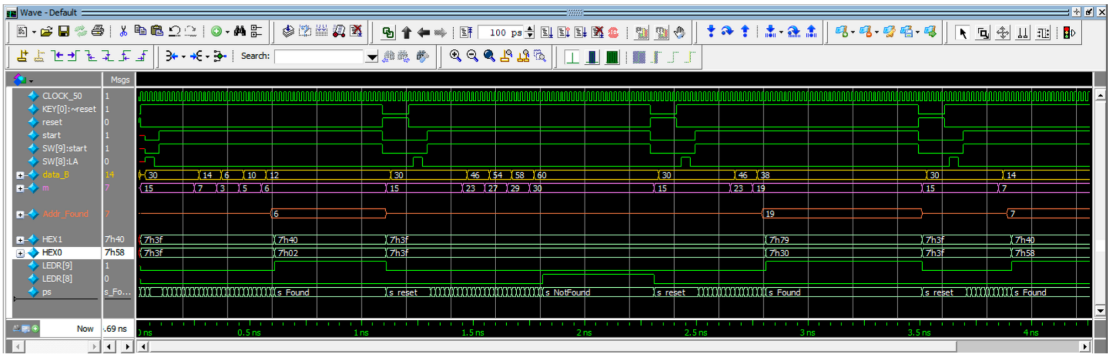

Figure 14: Simulation results of “binsearch_top” in Task 2.

III. Final Product

I finished the design specified in Task 1 and Task 2. I organized each design as shown in the corresponding block diagrams, simulated on the Modelsim with testbenches, and verified the functions with DE1-SoC on the LabsLand. In this Lab, I finished designin the bit-counting circuit and the binary search circuit from specifications to ASMD chart to SystemVerilog codes. These two design were well simulated on Modelsim and well-verified on LabsLand.

IV. Appendix

1. Video for Task 1, https://youtu.be/gZyCoplap5o
2. Video for Task 2, https://youtu.be/e4kpf0vmWpE
3. SystemVerilog codes of Task 1, please see the following pages.
4. SystemVerilog codes of Task 2, please see the following pages.

LAB4-Task1

LAB4-Task1-DE1_SoC.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware  (Task 1)
5   //  Device under Test (dut) --- DE1_SoC
6   //  File Name: DE1_SoC.sv
7   /*=================================================================*/
8   module DE1_SoC (CLOCK_50, SW, KEY, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0, LEDR);
9
10      input logic CLOCK_50;
11      input logic [9:0] SW;
12      input logic [3:0] KEY;
13      output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
14      output logic [9:0] LEDR;
15      logic [3:0] B;
16
17      // Turn OFF LEDR[8]~LEDR9.
18      assign LEDR[8:0] = 9'b0_0000_0000;
19
20      // Set HEX5~HEX1 to display "BLANK".
21      assign HEX5 = 7'h7f;
22      assign HEX4 = 7'h7f;
23      assign HEX3 = 7'h7f;
24      assign HEX2 = 7'h7f;
25      assign HEX1 = 7'h7f;
26
27
28      /*=================================================================*/
29      //      clock_divider (File Name: clock_divider.sv)
30      /*=================================================================*/
31       logic [31:0] div_clk;
32       parameter whichClock = 6; // 0.75 Hz clock  (50 MHz / 2^26 = 0.75 Hz)
33
34       clock_divider cdiv (.clock(CLOCK_50), .reset(~KEY[0]), .divided_clocks(div_clk));
35      /*-----------------------------------------------------------------*/
36      //      Clock selection;
37      //      allows for easy switching between simulation and board clocks
38      /*-----------------------------------------------------------------*/
39       logic clkSelect;
40
41      //Uncomment ONE of the following two lines depending on intention
42      //*****************************************************************
43         //assign clkSelect = CLOCK_50;           // for simulation
44           assign clkSelect = div_clk[whichClock]; // for DE1_SoC  board
45      //*****************************************************************
46      /*=================================================================*/
47
48      // Instantiate bitcounting module
49      bitcounting #(8, 4)  m1
50             (.clk(clkSelect),
51              .reset(~KEY[0]),
52              .LA(SW[8]),       // SW[8]: load data to shift register
53              .s(SW[9]),        // SW[9]: start to count bit 1.
54              .data(SW[7:0]),   // SW[7:0]: data to be loaded
55              .B(B),            // output of counter
56              .Done(LEDR[9])); // to indicate if process is finished.
57
58      // Instantiate data7seg module
59      data7seg m2 (.data(B), .seg(HEX0));
60
61  endmodule
62
63  /*=================================================================*/
64  //  Testbench
65  /*=================================================================*/
66  module DE1_SoC_testbench();
67
68      logic CLOCK_50;
69      logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
70      logic [3:0] KEY;
71      logic [9:0] SW;
72      logic [9:0] LEDR;
73      logic reset;
74      logic [7:0] data;
75      logic s;
76      logic LA;
77      logic clk;
78
```

```
79        assign clk = CLOCK_50;
80        assign KEY[0] = ~reset;
81        assign SW[7:0] = data;
82        assign SW[8] = LA;
83        assign SW[9] = s;
84
85        DE1_SoC dut (.CLOCK_50, .SW, .KEY,
86                     .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0, .LEDR);
87
88        // Set up a simulated clock: 50 MHz
89        parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
90
91        initial begin
92            CLOCK_50 <= 0;
93            forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
94        end
95
96        // Test the design.
97        initial begin
98            reset <= 1;
99            repeat(1)
00            @(posedge clk);
01
02            reset <= 0;
03            data <= 8'b1010_1110;
04            repeat(200)
05            @(posedge clk);
06
07            s <= 0;
08            LA <= 1;
09            repeat(500)
10            @(posedge clk);
11
12            LA <= 0;
13            @(posedge clk);
14
15            s <= 1;
16            repeat(1500)
17            @(posedge clk);
18
19            s <= 0;
20            repeat(300)
21            @(posedge clk);
22            $stop;
23        end
24
25    endmodule
```

LAB4-Task1

. LAB4-Task1-bitcounting.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware (Task 1)
5   //  Device under Test (dut) --- bitcounting
6   //  File Name: bitcounting.sv
7   /*=================================================================*/
8   module bitcounting #(parameter WIDTH_A = 8, WIDTH_B = 4)
9                       (clk, reset, LA, s, data, B, Done);
10      input logic clk, reset;
11      input logic LA; // Load signal of shfit register.
12      input logic s;
13      input logic [WIDTH_A-1:0]data; // input data of shift register
14      output logic [WIDTH_B-1:0]B; // output of counter
15      output logic Done;
16
17      logic EA, LB, EB, Z;
18      logic [WIDTH_A-1:0] A;
19
20      // State variables
21      typedef enum logic [1:0] {S1, S2, S3} state_t;
22      state_t ps, ns;
23
24
25      /*      Combinational_Logic_Here   */
26      // Next State logic
27      always_comb begin
28          case (ps)
29              S1:
30                ns = s ? S2 : S1;
31
32              S2:
33                ns = (Z==1) ? S3 : S2;
34
35              S3:
36                ns = s ? S3 : S1;
37
38              default: ns = S1;
39          endcase
40      end
41
42      /*      State Transition at posedge clk  */
43      always_ff @(posedge clk) begin
44          if (reset)
45              ps <= S1;
46          else
47              ps <= ns;
48      end
49
50
51      /*   Outputs   */
52      assign LB = (ps == S1);
53      assign EA = (ps == S2);
54      assign EB = ((ps == S2) && (A[0] == 1'b1));
55      assign Done = (ps == S3);
56
57      /*      Data Path   */
58      always_ff @(posedge clk) begin
59          if (reset) B <= 0;
60          else if (LB) B <= 0;
61          else if (EB) B <= B + 4'b0001;
62      end
63
64      // call shifter module
65      shift #(8) sh (.clk, .reset, .data, .LA, .EA, .A);
66
67      assign Z = ~|A; // detect if A == 0.
68
69  endmodule
70  /*=================================================================*/
71  //   Testbench for bitcounting
72  /*=================================================================*/
73  module bitcounting_testbench();
74
75      parameter WIDTH_A = 8; // set bit-width of data to be counted
76      parameter WIDTH_B = 4; // set bit-width of counter
77
78      logic clk, reset;
```

```
79        logic LA;
80        logic s;
81        logic [WIDTH_A-1:0]data; //data to be counted
82        logic [WIDTH_B-1:0]B;
83        logic Done;
84
85        bitcounting #(WIDTH_A, WIDTH_B) dut (.*);
86
87        parameter CLK_Period = 100;
88
89        initial begin
90            clk <= 1'b0;
91            forever #(CLK_Period/2) clk <= ~clk;
92        end
93
94        initial begin
95            reset <= 1;
96
97            repeat(1)
98            @(posedge clk);
99            reset <= 0;
00            data <= 8'b1010_1110;
01
02            repeat(1)
03            @(posedge clk);
04            s <= 0;
05            LA <= 1;
06            repeat(2)
07            @(posedge clk);
08            LA <= 0;
09            @(posedge clk);
10            s <= 1;
11
12            repeat(12)
13            @(posedge clk);
14
15            $stop;
16        end
17
18    endmodule
```

LAB4-Task1

I. LAB4-Task1-shift.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware (Task 1)
5   //  Device under Test (dut) --- shift
6   //  File Name: shift.sv
7   /*=================================================================*/
8   module shift #(parameter BIT_WIDTH = 8) (clk, reset, data, LA, EA, A);
9       input logic clk, reset;
10      input LA; // load input data parallely
11      input EA; // 1-bit shift right
12      input logic [BIT_WIDTH-1:0] data; // input data of shift regiter
13      output logic [BIT_WIDTH-1:0] A; // output data of shift register
14
15      always_ff @(posedge clk)
16          if (reset)
17              A <= 0;
18          else if (LA)
19              A <= data;      // load data parallely
20          else if (EA)
21              A <= (A >> 1); // 1-bit shift right per clock cycle
22
23      endmodule
24
25  /*=================================================================*/
26  //   Testbench for shift
27  /*=================================================================*/
28  module shift_testbench();
29
30      parameter BIT_WIDTH = 8; // set bit-width of shift register
31
32      logic clk, reset, LA, EA;
33      logic [BIT_WIDTH-1:0] data;
34      logic [BIT_WIDTH-1:0] A;
35
36      shift #(BIT_WIDTH) dut (.*);
37
38      // set clock signal
39      parameter CLK_Period = 100;
40
41      initial begin
42          clk <= 1'b0;
43          forever #(CLK_Period/2) clk <= ~clk;
44      end
45
46      // set test patterns
47
48      initial begin
49          reset <= 1;     // reset
50          repeat(1)
51          @(posedge clk);
52
53          reset <= 0;
54          data <= 8'b1010_1110;
55          repeat(1)
56          @(posedge clk);
57
58          LA <= 1;        // load dada pararllely
59          repeat(2)
60          @(posedge clk);
61
62          LA <= 0;
63          @(posedge clk);
64
65          EA <= 1;        // 1-bit shift right per clock cycle
66          repeat(10)
67          @(posedge clk);
68
69          $stop;
70      end
71
72  endmodule
```

LAB4-Task1

IV. LAB4-Task1-data2seg.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware  (Task 2)
5   //  Device under Test (dut) --- data7seg
6   //  File Name: data7seg.sv
7   /*=================================================================*/
8   module data7seg(data, seg);
9
10      input logic [3:0] data;
11      output logic [6:0] seg;
12
13      parameter [6:0] Blank = 7'b111_1111;  // 7'h7f
14      parameter [6:0] Chr_0 = 7'b100_0000;  // 7'h40
15      parameter [6:0] Chr_1 = 7'b111_1001;  // 7'h79
16      parameter [6:0] Chr_2 = 7'b010_0100;  // 7'h24
17      parameter [6:0] Chr_3 = 7'b011_0000;  // 7'h30
18      parameter [6:0] Chr_4 = 7'b001_1001;  // 7'h19
19      parameter [6:0] Chr_5 = 7'b001_0010;  // 7'h12
20      parameter [6:0] Chr_6 = 7'b000_0010;  // 7'h02
21      parameter [6:0] Chr_7 = 7'b101_1000;  // 7'h58
22      parameter [6:0] Chr_8 = 7'b000_0000;  // 7'h00
23      parameter [6:0] Chr_9 = 7'b001_0000;  // 7'h10
24      parameter [6:0] Chr_A = 7'b000_1000;  // 7'h08
25      parameter [6:0] Chr_b = 7'b000_0011;  // 7'h03
26      parameter [6:0] Chr_C = 7'b100_0110;  // 7'h46
27      parameter [6:0] Chr_d = 7'b010_0001;  // 7'h21
28      parameter [6:0] Chr_E = 7'b000_0110;  // 7'h06
29      parameter [6:0] Chr_F = 7'b000_1110;  // 7'h0e
30
31      always_comb begin
32          case(data)
33              0:  seg = Chr_0;
34              1:  seg = Chr_1;
35              2:  seg = Chr_2;
36              3:  seg = Chr_3;
37
38              4:  seg = Chr_4;
39              5:  seg = Chr_5;
40              6:  seg = Chr_6;
41              7:  seg = Chr_7;
42
43              8:  seg = Chr_8;
44              9:  seg = Chr_9;
45             10:  seg = Chr_A;
46             11:  seg = Chr_b;
47
48             12:  seg = Chr_C;
49             13:  seg = Chr_d;
50             14:  seg = Chr_E;
51             15:  seg = Chr_F;
52             default: seg = Blank;
53          endcase
54      end
55
56  endmodule
57
58  /*=================================================================*/
59  //   Testbench
60  /*=================================================================*/
61  module data7seg_testbench();
62
63      logic [3:0] data;
64      logic [6:0] seg;
65
66      integer i;
67
68      // instance
69      data7seg dut (.data, .seg);
70
71      // test this module with 16 patterns
72      initial begin
73          for (i = 0; i < 16; i++) begin
74              data = i;
75              #10;
76          end
77          $stop;
78      end
```

```
79    endmodule
```

LAB4-Task1

V. LAB4-Task1-clock_divider.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware (Task 1)
5   //  Device under Test (dut) --- clock_divider
6   //  File Name: clock_divider.sv
7   /*=================================================================*/
8   // divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz,
    ...
9   module clock_divider (clock, reset, divided_clocks);
10
11      input logic reset, clock;
12      output logic [31:0] divided_clocks = 0;
13
14      always_ff @(posedge clock) begin
15          divided_clocks <= divided_clocks + 1;
16      end
17
18  endmodule
19
20  /*=================================================================*/
21  //   Testbench
22  /*=================================================================*/
23  module clock_divider_testbench();
24
25      logic clk, reset;
26      logic [31:0] divided_clocks;
27
28      // instance
29      clock_divider dut (.clock(clk), .reset, .divided_clocks);
30
31      parameter CLK_Period = 100;  // set clock period
32
33      // generate clock signal
34      initial begin
35          clk <= 1'b0;
36          forever #(CLK_Period/2) clk <= ~clk;
37      end
38
39      // reset and test this module
40      initial begin
41      reset <= 1;
42
43      repeat(3)
44      @(posedge clk);
45      reset <= 0;
46
47      repeat(10000)
48      @(posedge clk);
49
50      $stop;
51  end
52
53  endmodule
```

LAB4-Task2

LAB4-Task2-DE1_SoC.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware  (Task 2)
5   //  Device under Test (dut) --- DE1_SoC
6   //  File Name: DE1_SoC.sv
7   /*=================================================================*/
8   module DE1_SoC (CLOCK_50, SW, KEY, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0, LEDR);
9
10      input logic CLOCK_50;
11      input logic [9:0] SW;
12      input logic [3:0] KEY;
13      output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
14      output logic [9:0] LEDR;
15
16      // Turn OFF LEDR[7]~LEDR[0].
17      assign LEDR[7:0] = 8'b0000_0000;
18
19      // Set HEX5~HEX2 to display "BLANK".
20      assign HEX5 = 7'h7f;
21      assign HEX4 = 7'h7f;
22      assign HEX3 = 7'h7f;
23      assign HEX2 = 7'h7f;
24
25      // Instantiate binsearch_top module
26      binsearch_top #(8, 5) m1 (
27      .clock (CLOCK_50),
28      .reset (~KEY[0]),
29      .LA (SW[8]),
30      .start (SW[9]),
31      .data_A (SW[7:0]),
32      .Found2LEDR (LEDR[9]),          // Use LEDR[9] to indicate "Found".
33      .NotFound2LEDR (LEDR[8]),       // Use LEDR[8] to indicate "NotFound".
34      .HEX1 (HEX1),                // use HEX1, HEX0 to display the address.
35      .HEX0 (HEX0));
36
37  endmodule
38
39  /*=================================================================*/
40  //  Testbench for DE1_SoC
41  /*=================================================================*/
42  `timescale 1 ps / 1 ps
43  module DE1_SoC_testbench();
44
45      logic CLOCK_50;
46      logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
47      logic [3:0] KEY;
48      logic [9:0] SW;
49      logic [9:0] LEDR;
50
51      // Declare the following signal names for readability.
52      logic reset;
53      logic [7:0] data_A;
54      logic start;
55      logic LA;
56      logic clock;
57
58      assign clock = CLOCK_50;
59      assign KEY[0] = ~reset;
60      assign SW[7:0] = data_A;
61      assign SW[8] = LA;
62      assign SW[9] = start;
63
64      // Instantiate DE1_SoC module
65      DE1_SoC dut (.CLOCK_50, .SW, .KEY,
66                   .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0, .LEDR);
67
68      // Set up a simulated clock: 50 MHz
69      parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
70
71      initial begin
72          CLOCK_50 <= 0;
73          forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
74      end
75
76      // Test the design.
77      initial begin
78          reset <= 1;
```

```
79          repeat(1)
80          @(posedge clock);
81
82          reset <= 0;
83          data_A <= 8'b0000_1100;  // Prepare the 1st data to be sought.
84          repeat(1)
85          @(posedge clock);
86
87          start <= 0;
88          LA <= 1;           // Load the 1st data to be sought.
89          repeat(2)
90          @(posedge clock);
91
92          LA <= 0;
93          @(posedge clock);
94
95          start <= 1;
96          repeat(50)
97          @(posedge clock);
98
99          start <= 0;
00          reset <= 1;
01          repeat(6)
02          @(posedge clock);
03
04          reset <= 0;
05          data_A <= 8'b011_1110;  // Prepare the 2nd data to be sought.
06          repeat(1)
07          @(posedge clock);
08
09          start <= 0;
10          LA <= 1;           // Load the 2nd data to be sought.
11          repeat(2)
12          @(posedge clock);
13
14          LA <= 0;
15          @(posedge clock);
16
17          start <= 1;
18          repeat(50)
19          @(posedge clock);
20
21
22          start <= 0;
23          reset <= 1;
24          repeat(6)
25          @(posedge clock);
26
27          reset <= 0;
28          data_A <= 8'b0010_0110;  // Prepare the 3rd data to be sought.
29          repeat(1)
30          @(posedge clock);
31
32          start <= 0;
33          LA <= 1;           // Load the 3rd data to be sought.
34          repeat(2)
35          @(posedge clock);
36
37          LA <= 0;
38          @(posedge clock);
39
40          start <= 1;
41          repeat(50)
42          @(posedge clock);
43
44          start <= 0;
45          reset <= 1;
46          repeat(6)
47          @(posedge clock);
48
49          reset <= 0;
50          data_A <= 8'b0000_1110;  // Prepare the 4th data to be sought.
51          repeat(1)
52          @(posedge clock);
53
54          start <= 0;
55          LA <= 1;           // Load the 4th data to be sought.
56          repeat(2)
```

```
57            @(posedge clock);
58
59            LA <= 0;
60            @(posedge clock);
61
62            start <= 1;
63            repeat(50)
64            @(posedge clock);
65
66            start <= 0;
67            $stop;
68        end
69
70    endmodule
```

LAB4-Task2

. LAB4-Task2-binsearch_top.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware (Task 2)
5   //  Device under Test (dut) --- binsearch_top
6   //  File Name: binsearch_top.sv
7   /*=================================================================*/
8   module binsearch_top
9       #(parameter BIT_WIDTH = 8, parameter ADDR_WIDTH = 5)(
10      input logic clock,
11      input logic reset,
12      input logic LA,
13      input logic start,
14      input logic [BIT_WIDTH-1:0] data_A,
15      output logic Found2LEDR,
16      output logic NotFound2LEDR,
17      output logic [6:0] HEX1,
18      output logic [6:0] HEX0);
19
20      // Declare the internal signals bewteen the Controller and the Datapath.
21      logic A_EQ_B, A_GT_B, A_LT_B, terminate, Ld_A, Ld_UpAddr, Ld_LowAddr;
22      logic get_Addr_sum, get_Addr_m, read_mem, compare, Found, NotFound;
23      logic New_UpAddr,New_LowAddr;
24
25      logic [BIT_WIDTH-1:0] data_B;
26      logic [ADDR_WIDTH-1:0] m, Addr_Found;
27
28      logic [6:0] HEX1_mux, HEX0_mux;  // The bridge-signals between datapath and HEX1,
    HEX0.
29
30      // Instantiate binsearch_ctrl (Controller) module
31      binsearch_ctrl m1 (
32      .clock,
33      .reset,
34      .LA,
35      .start,
36      .A_EQ_B,
37      .A_GT_B,
38      .A_LT_B,
39      .terminate,
40      .Ld_A,
41      .Ld_UpAddr,
42      .Ld_LowAddr,
43      .get_Addr_sum,
44      .get_Addr_m,
45      .compare,
46      .Found,
47      .NotFound,
48      .New_UpAddr,
49      .New_LowAddr);
50
51      // Instantiate binsearch_datapath module
52      binsearch_datapath #(8,5) m2 (
53      .clock,
54      .reset,
55      .A_EQ_B,
56      .A_GT_B,
57      .A_LT_B,
58      .terminate,
59      .Ld_A,
60      .Ld_UpAddr,
61      .Ld_LowAddr,
62      .get_Addr_sum,
63      .get_Addr_m,
64      .compare,
65      .Found,
66      .NotFound,
67      .New_UpAddr,
68      .New_LowAddr,
69      .data_A,
70      .data_B,
71      .m,
72      .Addr_Found,
73      .Found2LEDR,
74      .NotFound2LEDR);
75
76      // Instantiate ram32x8 module
77      ram32x8 m3 (.clock, .data(8'b0), .rdaddress(m), .wraddress(5'b0),
```

```
78                  .wren(1'b0), .q(data_B));
79
80      // Instantiate data7seg module to display addresses on HEX1 and HEX0.
81      data7seg m4 (.data({3'b000,Addr_Found[4]}), .seg(HEX1_mux));
82      data7seg m5 (.data(Addr_Found[3:0]), .seg(HEX0_mux));
83
84      // Selecting the proper signal to HEX1 and HEX0 depends on Found or NotFound.
85      assign HEX1 = (Found && ~NotFound) ? HEX1_mux : 7'h3f;
86      assign HEX0 = (Found && ~NotFound) ? HEX0_mux : 7'h3f;
87
88  endmodule
89
90  /*=================================================================*/
91  //   Testbench for binsearch_top
92  /*=================================================================*/
93  `timescale 1 ps / 1 ps
94  module binsearch_top_testbench();
95
96      parameter BIT_WIDTH = 8;
97      parameter ADDR_WIDTH = 5;
98
99      logic clock, reset;
00      logic LA;
01      logic start;
02      logic [BIT_WIDTH-1:0] data_A;
03      logic Found2LEDR;
04      logic NotFound2LEDR;
05      logic [6:0] HEX1;
06      logic [6:0] HEX0;
07
08      // Instantiate binsearch_top module
09      binsearch_top #(BIT_WIDTH, ADDR_WIDTH) dut (.*);
10
11
12      // Generate clock signal
13      parameter CLK_Period = 100;
14
15      initial begin
16          clock <= 1'b0;
17          forever #(CLK_Period/2) clock <= ~clock;
18      end
19
20      // Test the design.
21      initial begin
22          reset <= 1;
23          repeat(1)
24          @(posedge clock);
25
26          reset <= 0;
27          data_A <= 8'b0000_1100;  // Prepare the 1st data to be sought.
28          repeat(1)
29          @(posedge clock);
30
31          start <= 0;
32          LA <= 1;            // Load the 1st data to be sought.
33          repeat(2)
34          @(posedge clock);
35
36          LA <= 0;
37          @(posedge clock);
38
39          start <= 1;
40          repeat(50)
41          @(posedge clock);
42
43          start <= 0;
44          reset <= 1;
45          repeat(6)
46          @(posedge clock);
47
48          reset <= 0;
49          data_A <= 8'b011_1110;  // Prepare the 2nd data to be sought.
50          repeat(1)
51          @(posedge clock);
52
53          start <= 0;
54          LA <= 1;            // Load the 2nd data to be sought.
55          repeat(2)
```

```
56            @(posedge clock);
57
58            LA <= 0;
59            @(posedge clock);
60
61            start <= 1;
62            repeat(50)
63            @(posedge clock);
64
65
66            start <= 0;
67            reset <= 1;
68            repeat(6)
69            @(posedge clock);
70
71            reset <= 0;
72            data_A <= 8'b0010_0110;  // Prepare the 3rd data to be sought.
73            repeat(1)
74            @(posedge clock);
75
76            start <= 0;
77            LA <= 1;            // Load the 3rd data to be sought.
78            repeat(2)
79            @(posedge clock);
80
81            LA <= 0;
82            @(posedge clock);
83
84            start <= 1;
85            repeat(50)
86            @(posedge clock);
87
88            start <= 0;
89            reset <= 1;
90            repeat(6)
91            @(posedge clock);
92
93            reset <= 0;
94            data_A <= 8'b0000_1110;  // Prepare the 4th data to be sought.
95            repeat(1)
96            @(posedge clock);
97
98            start <= 0;
99            LA <= 1;            // Load the 4th data to be sought.
00            repeat(2)
01            @(posedge clock);
02
03            LA <= 0;
04            @(posedge clock);
05
06            start <= 1;
07            repeat(50)
08            @(posedge clock);
09
10            start <= 0;
11            $stop;
12        end
13
14    endmodule
```

LAB4-Task2

I. LAB4-Task2-binsearch_ctrl.sv


```
1   /*=====================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware (Task 2)
5   //  Device under Test (dut) --- binsearch_ctrl
6   //  File Name: binsearch_ctrl.sv
7   /*=====================================================================*/
8   module binsearch_ctrl (
9       input logic clock,
10      input logic reset,
11      input logic LA,
12      input logic start,
13      input logic A_EQ_B,
14      input logic A_GT_B,
15      input logic A_LT_B,
16      input logic terminate,
17      output logic Ld_A,
18      output logic Ld_UpAddr,
19      output logic Ld_LowAddr,
20      output logic get_Addr_sum,
21      output logic get_Addr_m,
22      output logic compare,
23      output logic Found,
24      output logic NotFound,
25      output logic New_UpAddr,
26      output logic New_LowAddr);
27
28      // State variables
29      typedef enum logic [2:0] {s_reset, ini_ready, sum_ready, m_ready, B_ready,
30                                s_Found, s_NotFound, comp_ready} state_t;
31      state_t ps, ns;
32
33      // Next State logic
34      always_comb begin
35          ns = s_reset; // Default to avoid latches
36          case (ps)
37              s_reset:
38                      ns = start ? ini_ready : s_reset;
39
40              ini_ready:
41                      ns = sum_ready;
42
43              sum_ready:
44                      ns = m_ready;
45
46              m_ready:
47                      ns = B_ready;
48
49              B_ready:
50                      ns = comp_ready;
51
52              comp_ready:
53                      if (A_EQ_B) ns =s_Found;
54                      else if (terminate) ns = s_NotFound;
55                      else ns = ini_ready;
56
57              s_Found:
58                      ns = start ? s_Found : s_reset;
59
60              s_NotFound:
61                      ns = start ? s_NotFound : s_reset;
62
63              default: ns = s_reset;
64          endcase
65      end // always_comb
66
67      /*      State Transition at posedge clk  */
68      always_ff @(posedge clock) begin
69          if (reset)
70              ps <= s_reset;
71          else
72              ps <= ns;
73      end
74
75      /*   Outputs   */
76      assign Ld_A         = (ps == s_reset) && LA && (!start);
77      assign Ld_UpAddr    = ((ps == s_reset) && (!start)) ? 1 : 0;
78      assign Ld_LowAddr   = ((ps == s_reset) && (!start)) ? 1 : 0;
```

LAB4-Task2

I. LAB4-Task2-binsearch_ctrl.sv


```
79        assign get_Addr_sum = (ps == ini_ready) ? 1 : 0;
80        assign get_Addr_m   = (ps == sum_ready)  ? 1 : 0;
81        assign compare      = (ps == B_ready) ? 1 : 0;
82        assign Found        = (ps == s_Found)  ? 1 : 0;
83        assign NotFound     = (ps == s_NotFound)  ? 1 : 0;
84        assign New_UpAddr   = (ps == comp_ready) && (!A_EQ_B) && (!terminate) && (A_GT_B)  ? 1
      : 0;
85        assign New_LowAddr  = (ps == comp_ready) && (!A_EQ_B) && (!terminate) && (A_LT_B)  ? 1
      : 0;
86
87    endmodule
```

LAB4-Task2

V. LAB4-Task2-binsearch_datapath.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware (Task 2)
5   //  Device under Test (dut) --- binsearch_datapath
6   //  File Name: binsearch_datapath.sv
7   /*=================================================================*/
8   module binsearch_datapath
9       #(parameter BIT_WIDTH = 8, parameter ADDR_WIDTH = 5)(
10      input logic clock, reset,
11      output logic A_EQ_B,         // A == B
12      output logic A_GT_B,         // A > B
13      output logic A_LT_B,         // A < B
14      output logic terminate,      // terminate searching
15      input logic Ld_A,            // Load data to be sought
16      input logic Ld_UpAddr,       // Set the initial address (0) of the top word.
17      input logic Ld_LowAddr,      // Set the initial address (31) of the bottom word.
18      input logic get_Addr_sum,    // Calculate the sum of the top and bottom addresses.
19      input logic get_Addr_m,      // Calculate the mean of the top and bottom
    addresses.
20      input logic compare,         // Compare data_A and data_B.
21      input logic Found,           // Indicate data_A was found in ram32x8.
22      input logic NotFound,        // Indicate data_A was not found in ram32x9
23      input logic New_UpAddr,      // Update the address of the NEW top word.
24      input logic New_LowAddr,     // Update the address of the NEW bottom word.
25      input logic [BIT_WIDTH-1:0] data_A,
26      input logic [BIT_WIDTH-1:0] data_B,
27      output logic [ADDR_WIDTH-1:0] m,           // mean of the top and bottom addresses.
28      output logic [ADDR_WIDTH-1:0] Addr_Found, // The address of the word that is the same
    as data_A.
29      output logic Found2LEDR,       // Turn on LEDR[9] when data_A was found in ram32x8.
30      output logic NotFound2LEDR);  // Turn on LEDR[8] when data_A was not found in ram32x8.
31
32      logic [BIT_WIDTH:0] A;
33      logic [ADDR_WIDTH-1:0] UpAddr, LowAddr; // The addresses of the top/bottom words.
34      logic [BIT_WIDTH:0] sum;
35
36      assign Found2LEDR = Found;        // Send "Found" from Controller to Datapth to turn
    on LEDR[9].
37      assign NotFound2LEDR = NotFound; // Send "NotFound" from Controller to Datapth to
    turn on LEDR[8].
38
39  /*    Data Path   */
40      always_ff @(posedge clock) begin
41          if (reset) begin
42              UpAddr <= 5'b0_0000;      // Set initial address of the top word to be 0.
43              LowAddr <= 5'b1_1111;     // Set initial address of the bottom word to be 31.
44              m <= 5'b0_1111;           // Set initial m to 15.
45              Addr_Found <= 5'bz_zzzz; // Set initial Addr_Found to be high impedance.
46          end
47          else begin
48              if (Ld_A) A <= data_A;
49              if (Ld_UpAddr)  UpAddr <= 5'b0_0000;
50              if (Ld_LowAddr) LowAddr <= 5'b1_1111;
51              if (get_Addr_sum)  sum <= (UpAddr + LowAddr);  // Calculate sum
52              if (get_Addr_m)  m <= sum >> 1;     // Divide by 2 by means of shift right 1
    bit.
53              if (compare) begin
54                  if (A == data_B) begin
55                      {A_EQ_B, A_GT_B, A_LT_B} <= 3'b100;
56                      Addr_Found <= m;
57                  end
58                  if (A > data_B)  {A_EQ_B, A_GT_B, A_LT_B} <= 3'b010;
59                  if (A < data_B)  {A_EQ_B, A_GT_B, A_LT_B} <= 3'b001;
60              end
61              if ((UpAddr == m) || (LowAddr == m))    terminate <= 1'b1;  // Set terminated
    condition
62              else terminate <= 1'b0;
63
64              if (New_UpAddr) UpAddr <= m + 1'b1;         // Update the address of the NEW
    top word.
65              if (New_LowAddr) LowAddr <= m - 5'b0_0001;// Update the address of the NEW
    bottom word.
66          end
67      end // always_ff
68  endmodule
69
```

LAB4-Task2

7. LAB4-Task2-data7seg.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-19-2025
4   //  EE/CSE371 LAB4--- Implementing Algorithm in Hardware  (Task 2)
5   //  Device under Test (dut) --- data7seg
6   //  File Name: data7seg.sv
7   /*=================================================================*/
8   module data7seg(data, seg);
9
10      input logic [3:0] data;
11      output logic [6:0] seg;
12
13      parameter [6:0] Blank = 7'b111_1111;  // 7'h7f
14      parameter [6:0] Chr_0 = 7'b100_0000;  // 7'h40
15      parameter [6:0] Chr_1 = 7'b111_1001;  // 7'h79
16      parameter [6:0] Chr_2 = 7'b010_0100;  // 7'h24
17      parameter [6:0] Chr_3 = 7'b011_0000;  // 7'h30
18      parameter [6:0] Chr_4 = 7'b001_1001;  // 7'h19
19      parameter [6:0] Chr_5 = 7'b001_0010;  // 7'h12
20      parameter [6:0] Chr_6 = 7'b000_0010;  // 7'h02
21      parameter [6:0] Chr_7 = 7'b101_1000;  // 7'h58
22      parameter [6:0] Chr_8 = 7'b000_0000;  // 7'h00
23      parameter [6:0] Chr_9 = 7'b001_0000;  // 7'h10
24      parameter [6:0] Chr_A = 7'b000_1000;  // 7'h08
25      parameter [6:0] Chr_b = 7'b000_0011;  // 7'h03
26      parameter [6:0] Chr_C = 7'b100_0110;  // 7'h46
27      parameter [6:0] Chr_d = 7'b010_0001;  // 7'h21
28      parameter [6:0] Chr_E = 7'b000_0110;  // 7'h06
29      parameter [6:0] Chr_F = 7'b000_1110;  // 7'h0e
30
31      always_comb begin
32          case(data)
33              0:  seg = Chr_0;
34              1:  seg = Chr_1;
35              2:  seg = Chr_2;
36              3:  seg = Chr_3;
37
38              4:  seg = Chr_4;
39              5:  seg = Chr_5;
40              6:  seg = Chr_6;
41              7:  seg = Chr_7;
42
43              8:  seg = Chr_8;
44              9:  seg = Chr_9;
45             10:  seg = Chr_A;
46             11:  seg = Chr_b;
47
48             12:  seg = Chr_C;
49             13:  seg = Chr_d;
50             14:  seg = Chr_E;
51             15:  seg = Chr_F;
52            default: seg = Blank;
53          endcase
54      end
55
56  endmodule
57
58  /*=================================================================*/
59  //   Testbench
60  /*=================================================================*/
61  module data7seg_testbench();
62
63      logic [3:0] data;
64      logic [6:0] seg;
65
66      integer i;
67
68      // instance
69      data7seg dut (.data, .seg);
70
71      // test this module with 16 patterns
72      initial begin
73          for (i = 0; i < 16; i++) begin
74              data = i;
75              #10;
76          end
77          $stop;
78      end
```

```
79      endmodule
```

LAB4-Task2

VI. LAB4-Task2-ram32x8.v

```
1   // megafunction wizard: %RAM: 2-PORT%
2   // GENERATION: STANDARD
3   // VERSION: WM1.0
4   // MODULE: altsyncram
5
6   // ============================================================
7   // File Name: ram32x8.v
8   // Megafunction Name(s):
9   //          altsyncram
10  //
11  // Simulation Library Files(s):
12  //          altera_mf
13  // ============================================================
14  // ************************************************************
15  // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
16  //
17  // 17.0.0 Build 595 04/25/2017 SJ Lite Edition
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 ram32x8 (
41      clock,
42      data,
43      rdaddress,
44      wraddress,
45      wren,
46      q);
47
48      input     clock;
49      input   [7:0]  data;
50      input   [4:0]  rdaddress;
51      input   [4:0]  wraddress;
52      input     wren;
53      output  [7:0]  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 [7:0] sub_wire0;
64      wire [7:0] q = sub_wire0[7:0];
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),
```

LAB4-Task2

VI. LAB4-Task2-ram32x8.v

```
74                  .aclr1 (1'b0),
75                  .addressstall_a (1'b0),
76                  .addressstall_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 ({8{1'b1}}),
85                  .eccstatus (),
86                  .q_a (),
87                  .rden_a (1'b1),
88                  .rden_b (1'b1),
89                  .wren_b (1'b0));
90      defparam
91          altsyncram_component.address_aclr_b = "NONE",
92          altsyncram_component.address_reg_b = "CLOCK0",
93          altsyncram_component.clock_enable_input_a = "BYPASS",
94          altsyncram_component.clock_enable_input_b = "BYPASS",
95          altsyncram_component.clock_enable_output_b = "BYPASS",
96          altsyncram_component.init_file = "my_array.mif",
97          altsyncram_component.intended_device_family = "Cyclone V",
98          altsyncram_component.lpm_type = "altsyncram",
99          altsyncram_component.numwords_a = 32,
100         altsyncram_component.numwords_b = 32,
101         altsyncram_component.operation_mode = "DUAL_PORT",
102         altsyncram_component.outdata_aclr_b = "NONE",
103         altsyncram_component.outdata_reg_b = "UNREGISTERED",
104         altsyncram_component.power_up_uninitialized = "FALSE",
105         altsyncram_component.ram_block_type = "M10K",
106         altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
107         altsyncram_component.widthad_a = 5,
108         altsyncram_component.widthad_b = 5,
109         altsyncram_component.width_a = 8,
110         altsyncram_component.width_b = 8,
111         altsyncram_component.width_byteena_a = 1;
112
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: CLRrdaddress NUMERIC "0"
134 // Retrieval info: PRIVATE: CLRrren NUMERIC "0"
135 // Retrieval info: PRIVATE: CLRwraddress 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_LES 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"
```

LAB4-Task2

VI. LAB4-Task2-ram32x8.v

```
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MEMSIZE NUMERIC "256"
// Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "ram32x8.mif"
// Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"
// Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "0"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "2"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_B NUMERIC "3"
// Retrieval info: PRIVATE: REGdata NUMERIC "1"
// Retrieval info: PRIVATE: REGq NUMERIC "1"
// Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
// Retrieval info: PRIVATE: REGrren NUMERIC "1"
// Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
// Retrieval info: PRIVATE: REGwren NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: USE_DIFF_CLKEN NUMERIC "0"
// Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
// Retrieval info: PRIVATE: VarWidth NUMERIC "0"
// Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "8"
// Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "8"
// Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "8"
// Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "8"
// Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "0"
// Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: enable NUMERIC "0"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADDRESS_ACLR_B STRING "NONE"
// Retrieval info: CONSTANT: ADDRESS_REG_B STRING "CLOCK0"
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_B STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_B STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "ram32x8.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "32"
// Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "32"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "DUAL_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_B STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_B STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_MIXED_PORTS STRING "DONT_CARE"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "5"
// Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "5"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
// Retrieval info: CONSTANT: WIDTH_B NUMERIC "8"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL "data[7..0]"
// Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
// Retrieval info: USED_PORT: rdaddress 0 0 5 0 INPUT NODEFVAL "rdaddress[4..0]"
// Retrieval info: USED_PORT: wraddress 0 0 5 0 INPUT NODEFVAL "wraddress[4..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT GND "wren"
// Retrieval info: CONNECT: @address_a 0 0 5 0 wraddress 0 0 5 0
// Retrieval info: CONNECT: @address_b 0 0 5 0 rdaddress 0 0 5 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 8 0 data 0 0 8 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 8 0 @q_b 0 0 8 0
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x8.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x8.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x8.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x8.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x8_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x8_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf

```

LAB4-Task2

VII. LAB4-Task2-my_array.mif

```
-- Copyright (C) 2017  Intel Corporation. All rights reserved.
-- Your use of Intel Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Intel Program License
-- Subscription Agreement, the Intel Quartus Prime License Agreement,
-- the Intel MegaCore Function License Agreement, or other
-- applicable license agreement, including, without limitation,
-- that your use is for the sole purpose of programming logic
-- devices manufactured by Intel and sold by Intel or its
-- authorized distributors.  Please refer to the applicable
-- agreement for further details.

-- Quartus Prime generated Memory Initialization File (.mif)

WIDTH=8;
DEPTH=32;

ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;

CONTENT BEGIN
    00  :   00000001;
    01  :   00000010;
    02  :   00000100;
    03  :   00000110;
    04  :   00001000;
    05  :   00001010;
    06  :   00001100;
    07  :   00001110;
    08  :   00010000;
    09  :   00010010;
    0A  :   00010100;
    0B  :   00010110;
    0C  :   00011000;
    0D  :   00011010;
    0E  :   00011100;
    0F  :   00011110;
    10  :   00100000;
    11  :   00100010;
    12  :   00100100;
    13  :   00100110;
    14  :   00101000;
    15  :   00101010;
    16  :   00101100;
    17  :   00101110;
    18  :   00110000;
    19  :   00110010;
    1A  :   00110100;
    1B  :   00110110;
    1C  :   00111000;
    1D  :   00111010;
    1E  :   00111100;
    1F  :   00111110;
END;
```