UW Student: Brian Chen

EE 371

January 28, 2025

Lab2 Report

I. Procedure

Task 1:

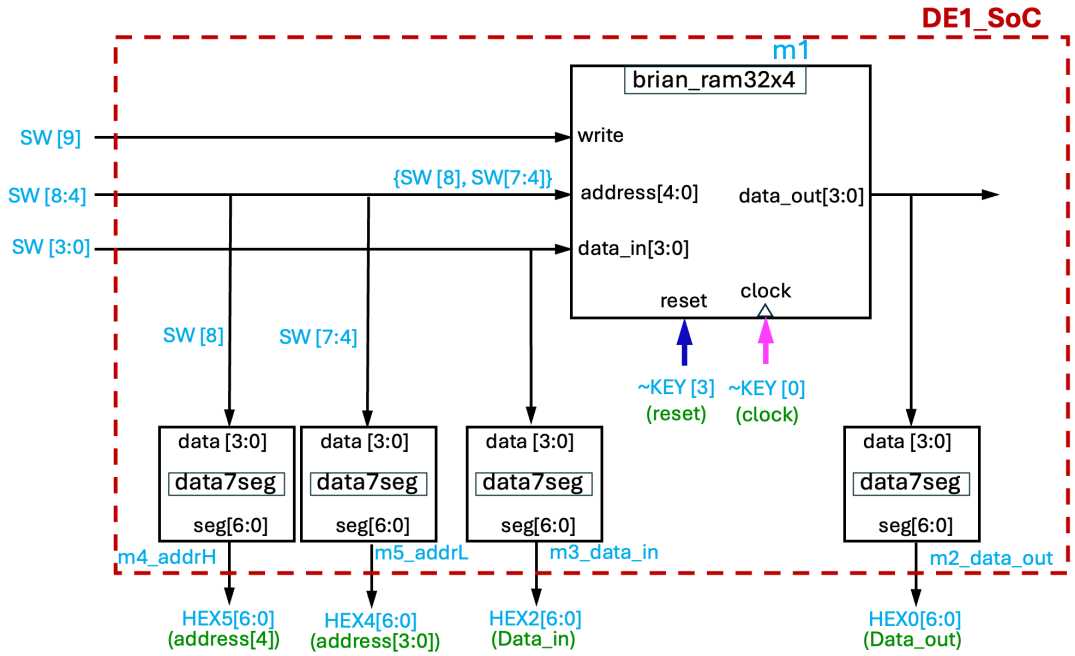

Figure 1: Block diagram of the top module of Task 1.

Figure 1 shows the block diagram of the design in Task 1. First, I wrote "brian_ram32x4" module, which contains the RAM structure with 32x4 array. It constructed with input variable clock, reset, write, address, data_in, and output variable data_out. I used the SystemVerilog statement "logic [3:0] meory_array [31:0];" as suggested in the course handout to create the array for this RAM. After this, I constructed the write operation by the SystemVerilog statement "always_ff @ (posedge clock)". Second, I constructed the module "data7seg" which is similar to "bcd7seg" module I used in the Lab 1. It's for the HEX display. Third, I used DE1_SoC as the top module. I collected "brian_ram32x4.sv" and "data7seg.sv" here, and I assigned SW[3:0] as the input data values, SW[8:4] as the specific address that for inputting data, KEY[0] as the clock input, SW[9] as the Write signal, HEX5 and HEX4 as the address value display that

data being input, HEX2 as the data value being input to the memory, and HEX0 as the data read out of the memory.

Task 2:

Figure 2 shows the block diagram of the design in Task 2. First, I created the environment for IP Catalog and MIF by following the instructions. Second, I constructed the "counter5b.sv" for sending out signal every clock cycle for later implement displaying every address and data value in address. Third, I built "data7seg.sv" and "address7seg.sv". The "data7seg.sv" is basically the same as the Task1, and the "address7seg.sv" is similar with it, but it got one more output for displaying the full address on HEX5 and HEX4. Fourth, "clock_divider.sv", I pull it from the previous lab project in EE271 in previous quarter. It's for dividing clock and could easier the simulation being observed. Fifth, I used "DE1_SoC.sv" as the top module which collects all others module including "ram32x4" which built by IP Catalog. I assigned SW[8:4] and SW[3:0] for the write address and corresponding data, respectively. I assigned the write 8-bit address and the 4-bit write data to be shown at HEX 5, HEX4, and HEX1, respectively. I assigned HEX0 for displaying the data value that in the word at the address shown on HEX3 and HEX2.

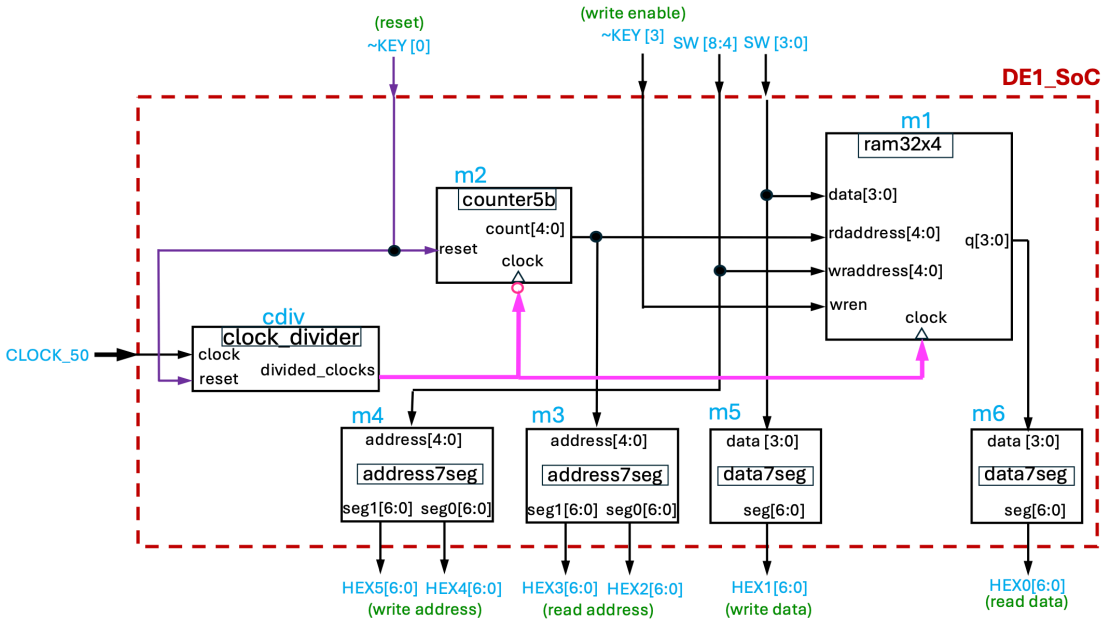

Figure 2: Block diagram of the top module of Task 2.

Task 3:

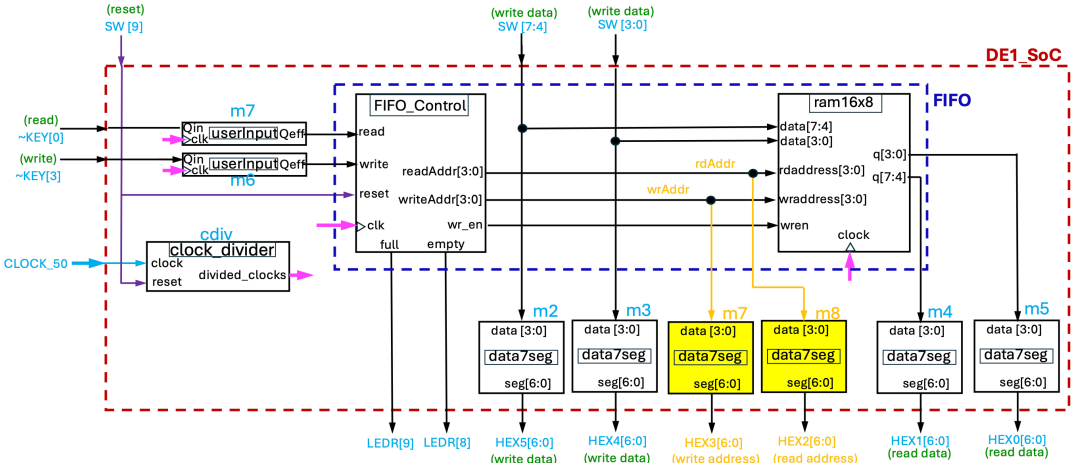

Figure 3: Block diagram of the top module of Task 3

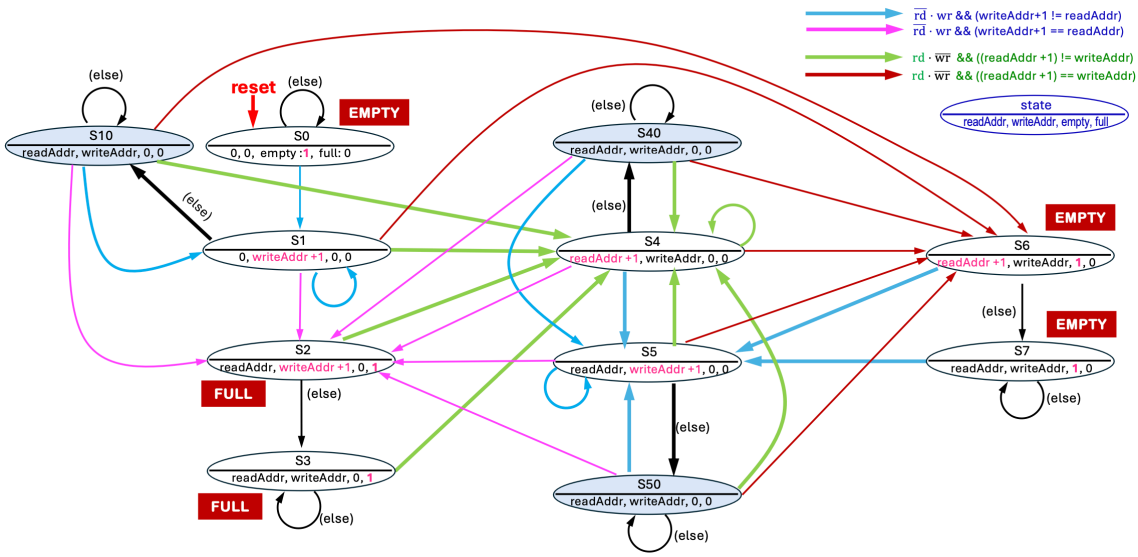

Figure 4: State diagram of finite-state machine for the FIFO control unit.

Figure 3 shows the block diagram of the design in Task 3. As indicated in yellow in this figure which are not specified in the course handout, I applied HEX3 and HEX2 for displaying the write address and read address, respectively. First, I chose IP Catalog to create “ram16x8.v” and put “data7seg.sv” and “clock_divider.sv” from earlier Task in Task 3 project. Second, I drew a finite-state machine (FSM) for FIFO control unit. Figure 4 shows the state diagram of the FSM. I wrote the SystemVerilog code for this unit as the FIFO_Control module in “FIFO_Control.sv”. Third, in FIFO.sv, I instantiated the dual port RAM by calling the module in “ram16x8.v” and collected it with FIFO_Control module. Fourth, I created “userInput.sv”, which is similar to Lab 1.

It is for not overcounting the signal of the KEY press input. Fifth, I used “DE1_SoC.sv” as the top module, and I collected all the modules. I assigned SW[7:0] as the data input; HEX5 and HEX4 as showing the value of the data inputting the FIFO. I assigned the current data output of the FIFO on HEX1 and HEX0. I assigned LEDR9 and LEDR8 to indicate “full” and “empty”, respectively. I assigned SW[9] as reset, KEY[3] as write, KEY[0] as read.

II. Result

Task 1:

Simulation Result with the Testbench for “DE1_SoC.sv”:

I wrote the testbench to test the top module in Task 1. In this testbench, I applied the data in the form of “running 1’s” for writing into the RAM at the specified addresses. After the write phase, I read the data from each address to verify if the RAM performs well. Figure 5 shows the simulation results on Modelsim. The upper portion shows the write phase, and the lower portion shows the read phase. Figures 6 and 7 show the simulation results of the modules “data7seg” and “brian_ram32x4”, respectively. As indicated by the yellow arrows in Figure 5, this RAM performs the functions as expected.

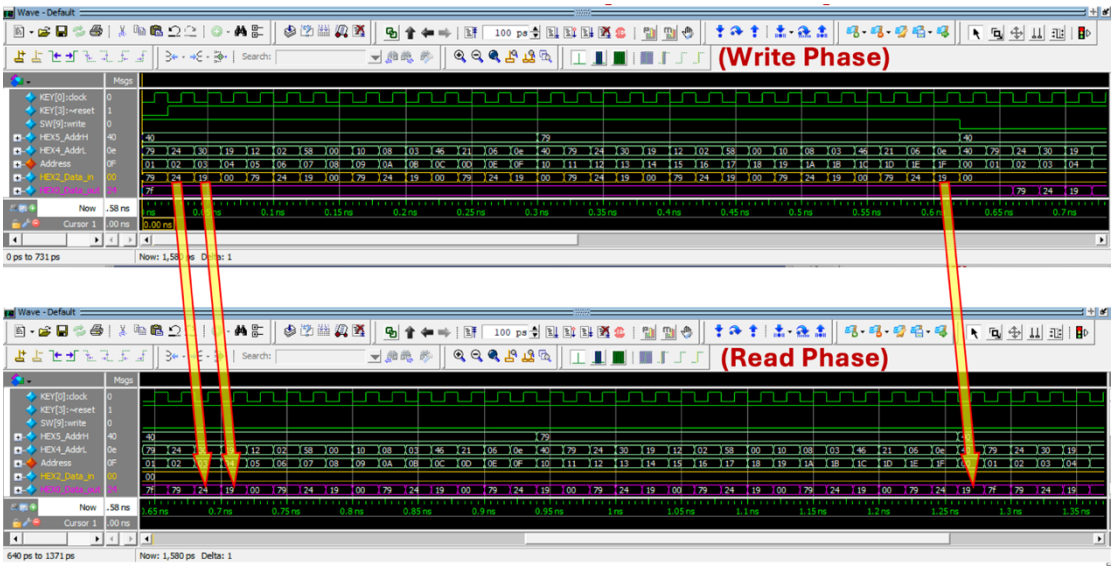

Figure 5: Simulation results of the design “DE1_SoC” in Task 1.

Simulation Result with the Testbench for “data7seg.sv”:

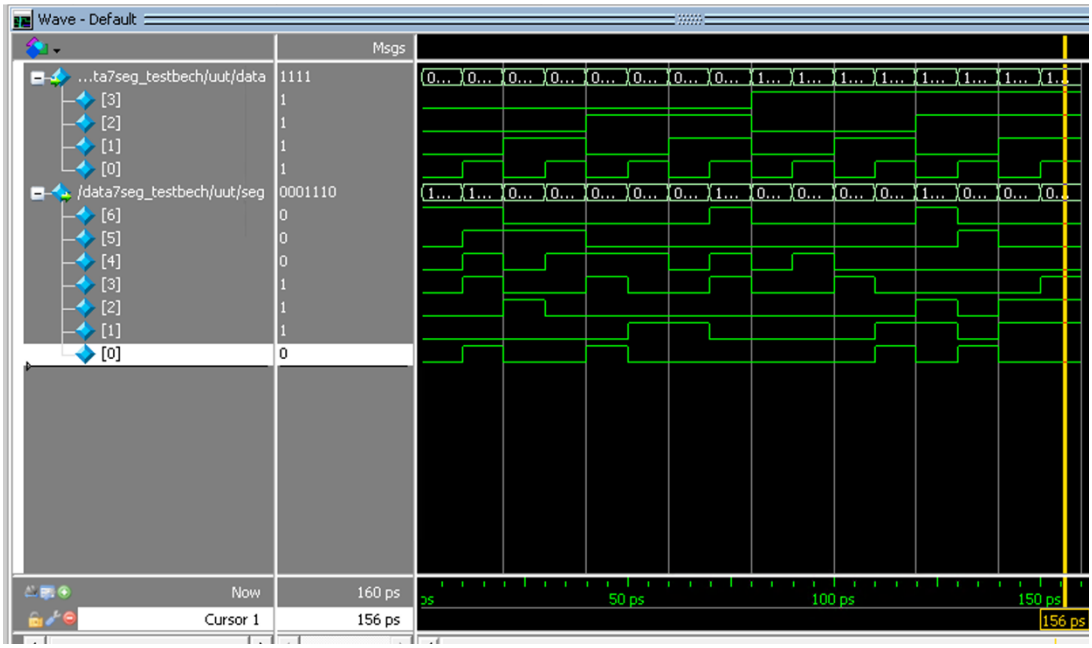

Figure 6: Simulation results of the design “data7seg” in Task 1.

Simulation Result with the Testbench for “brian_ram32x4.sv”:

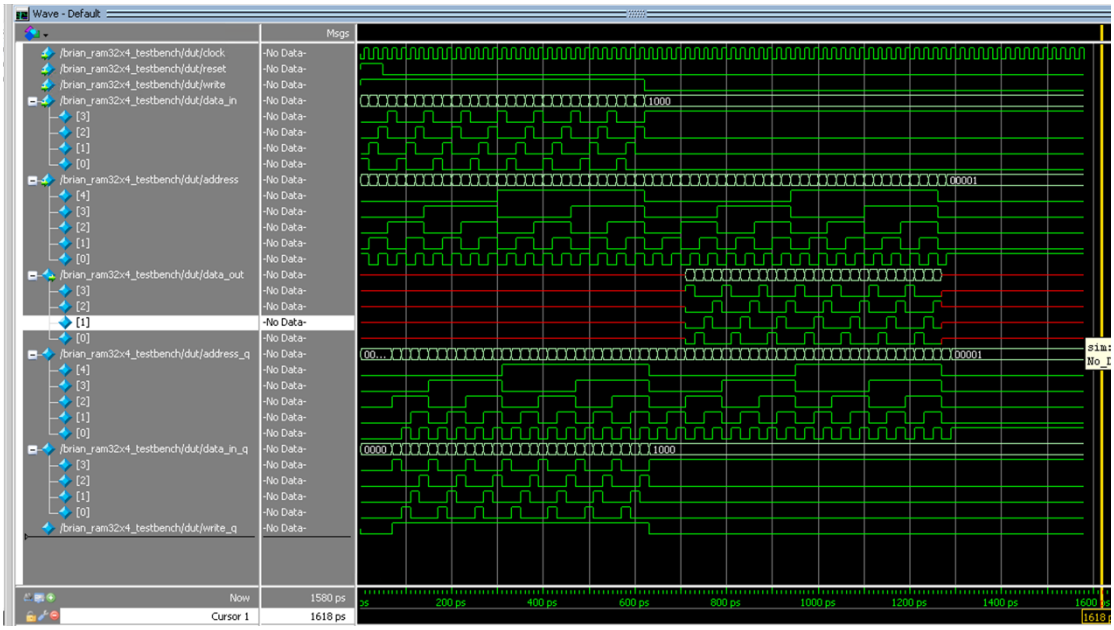

Figure 7: Simulation results of the design “brian_ram32x4” in Task 1.

Task 2:

Simulation Result with the Testbench for "DE1_SoC.sv":

I wrote the testbench to test the top module in Task 2. The simulation of the design in Task 2 includes two portions.

In the first portion, I created the file "ram32x4.mif" to initialize the contents of the RAM. After the initialization, I wrote the testbench to simulate the RAM. As the simulation results shown in Figure 8, the data read from the RAM at each address and the data specified in the file "ram32x4.mif" are matched.

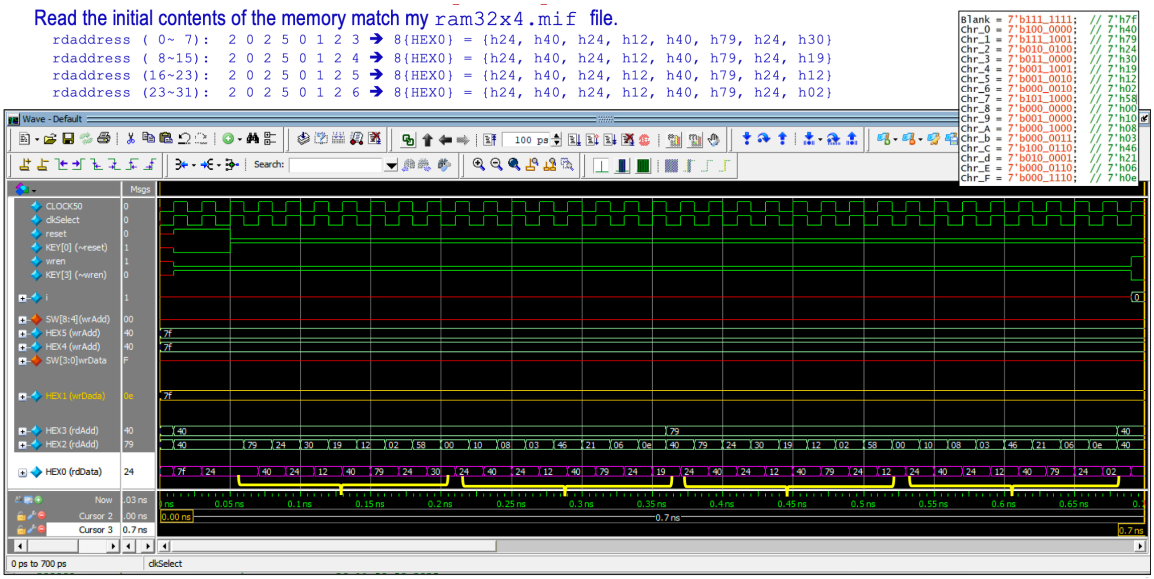

Figure 8: Simulation results of the design "DE1_SoC" in Task 2 with initial contents specified in the file "ram32x4.mif".

In the second portion, I specified the data to be written into the RAM in the testbench as shown in Figure 9. As the simulation results shown in Figure 10, the data read from the RAM at each address and the data specified in the testbench are matched.

Figures 11 to 14 show the simulation results of the modules "address7seg", "data7seg", "counter5b", and "clock_divider", respectively. Apparently, the simulation results indicate all the modules as mentioned in Task 2 perform well.

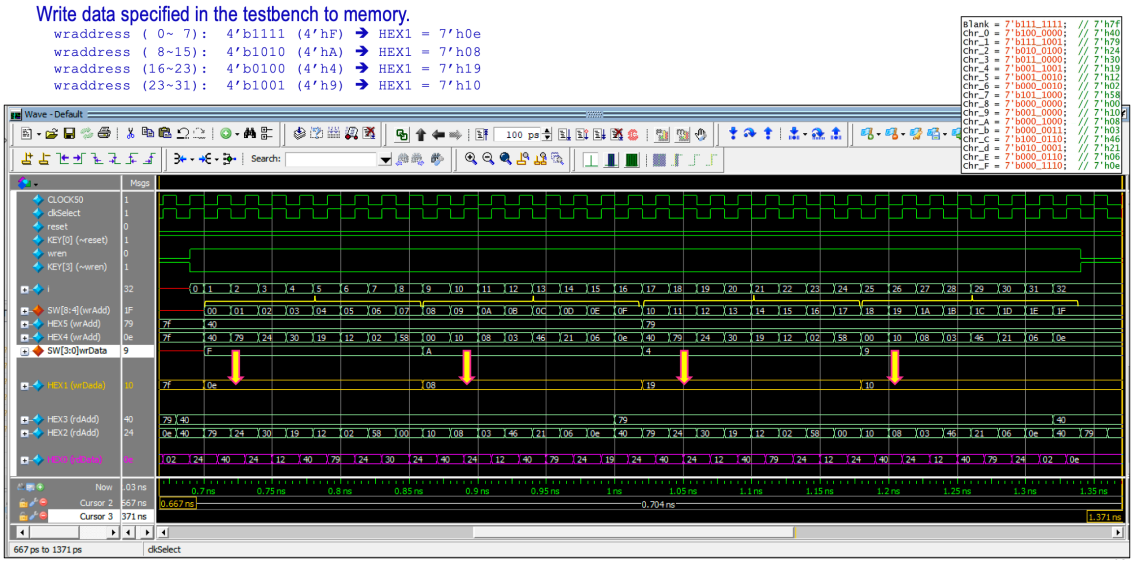

Figure 9: Simulation results of the design "DE1_SoC" in Task 2 with the data specified in the testbench (write phase).

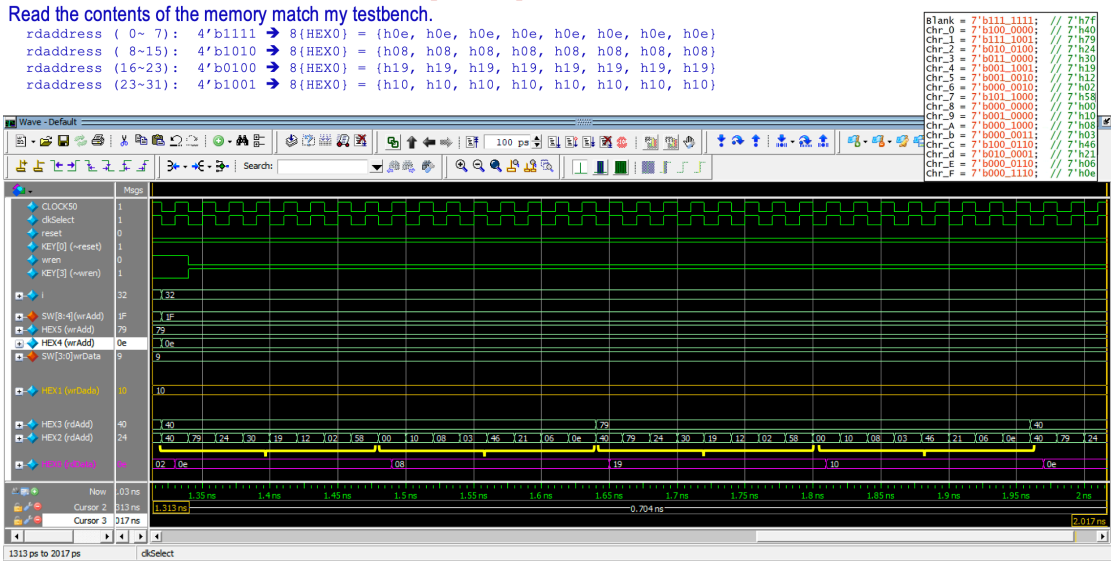

Figure 10: Simulation results of the design "DE1_SoC" in Task 2 with the data specified in the testbench (read phase).

Simulation Result with the Testbench for “address7seg.sv”:

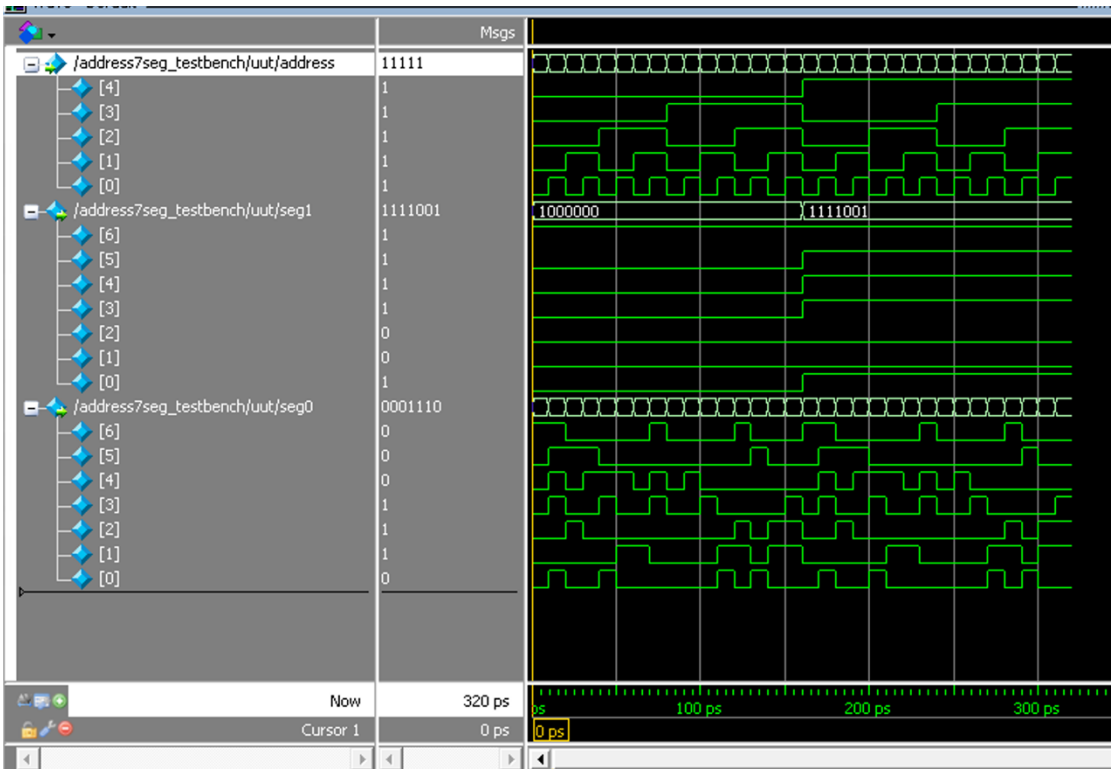

Figure 11: Simulation results of the design “address7seg” in Task 2.

Simulation Result with the Testbench for “data7seg.sv”:

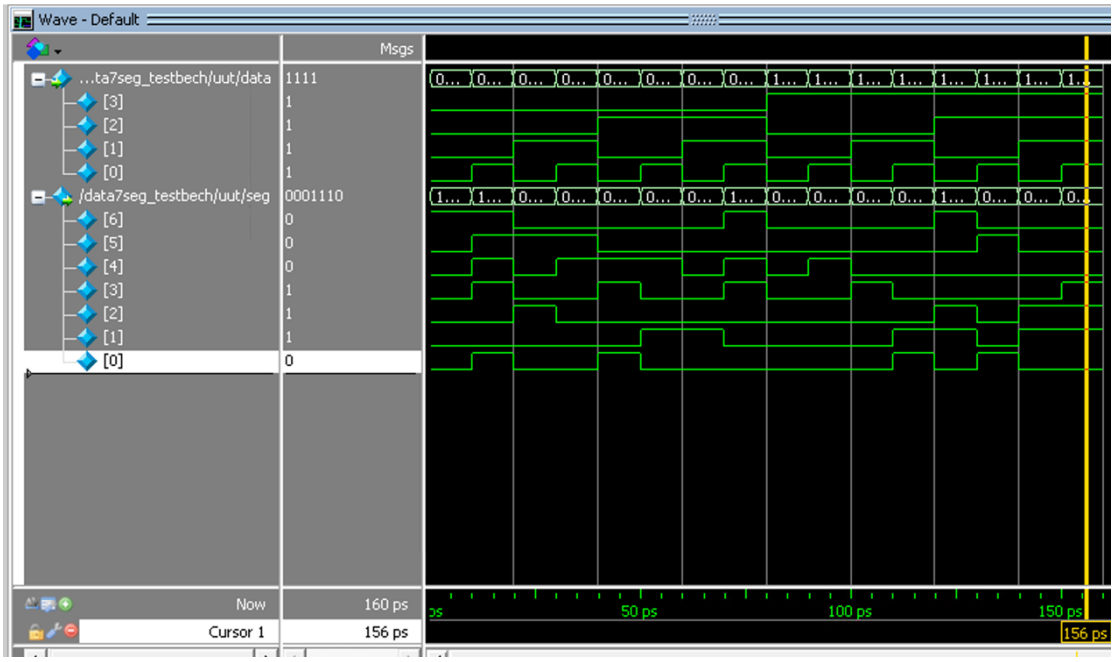

Figure 11: Simulation results of the design “data7seg” in Task 2.

Simulation Result with the Testbench for “counter5b.sv”:

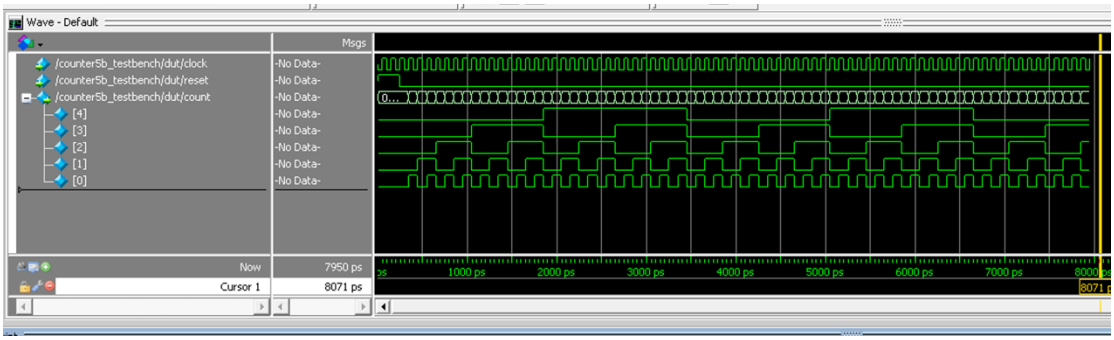

Figure 13: Simulation results of the design “counter5b” in Task 2.

Simulation Result with the Testbench for “clock_divider.sv”:

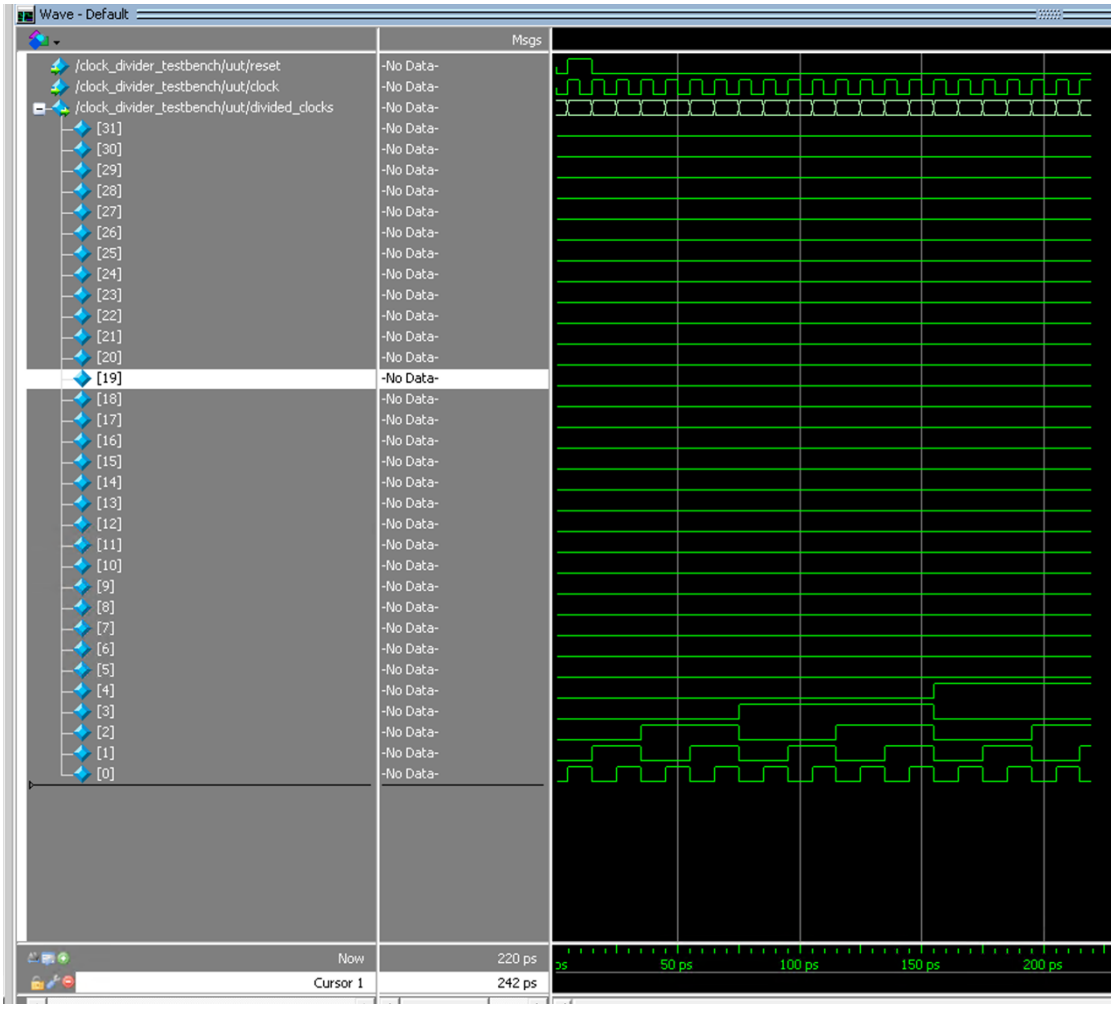

Figure 14: Simulation results of the design “clock_divider” in Task 2.

Task 3:

I wrote the testbench to test the top module in Task 3. Figure 15 shows the simulation results ranging from 0 ns to 10.5 ns. As indicated by the yellow arrows, the simulation results include the FIFO status at either full, empty, or neither. For more clear consideration, I zoomed in the waveforms in the intervals [0 ns, 3.5 ns], [3.5 ns, 7 ns], and [7 ns, 10.5 ns] as shown in Figures 16, 17, and 18, respectively.

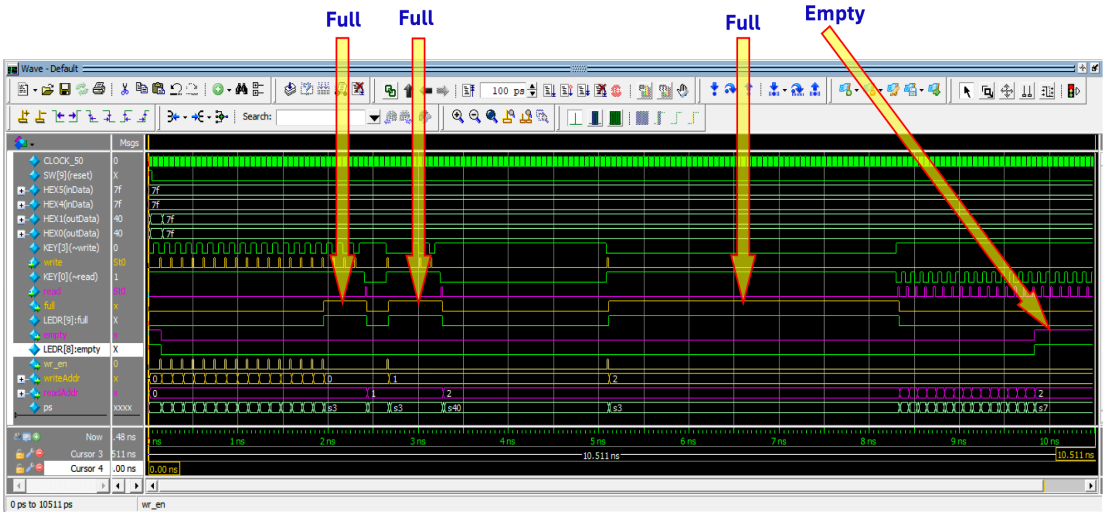

Figure 15: Simulation results of the design “DE1_SoC” in Task 3 (0 ns to 10.5 ns).

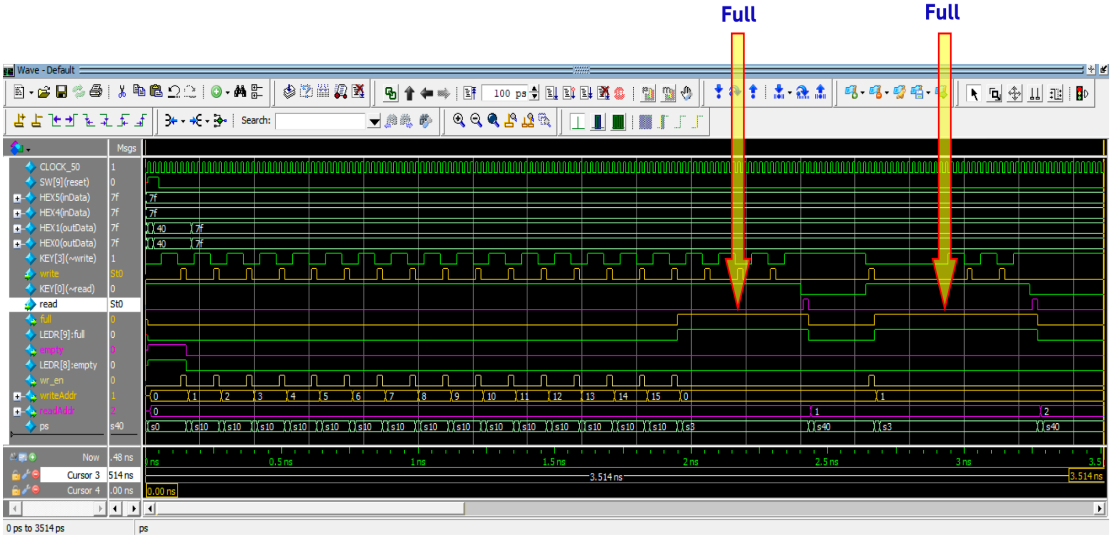

Figure 16: Simulation results of the design “DE1_SoC” in Task 3 (0 ns to 3.5 ns).

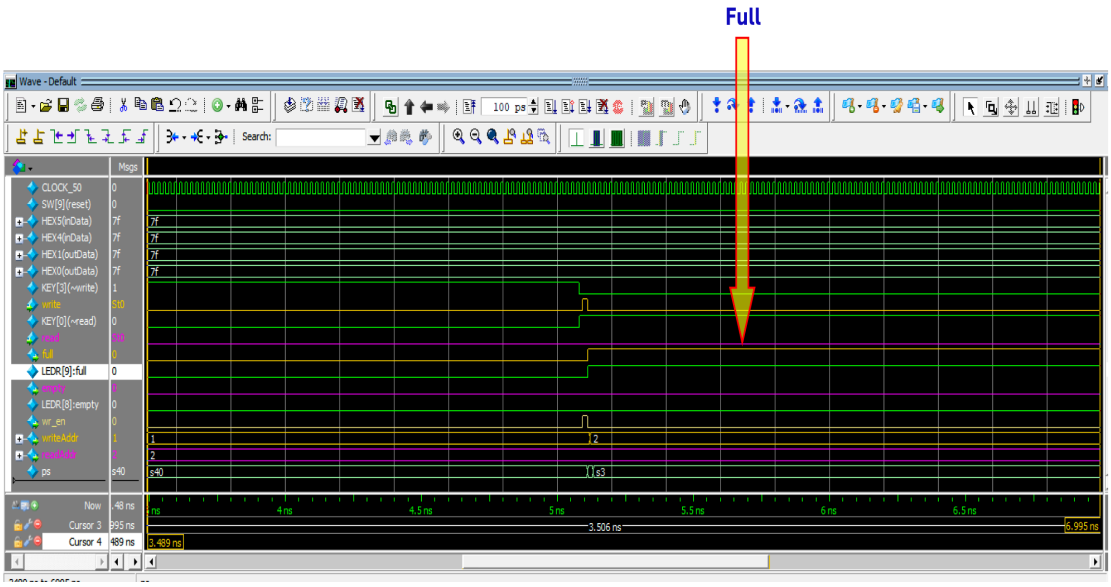

Figure 17: Simulation results of the design “DE1_SoC” in Task 3 (3.5 ns to 7 ns).

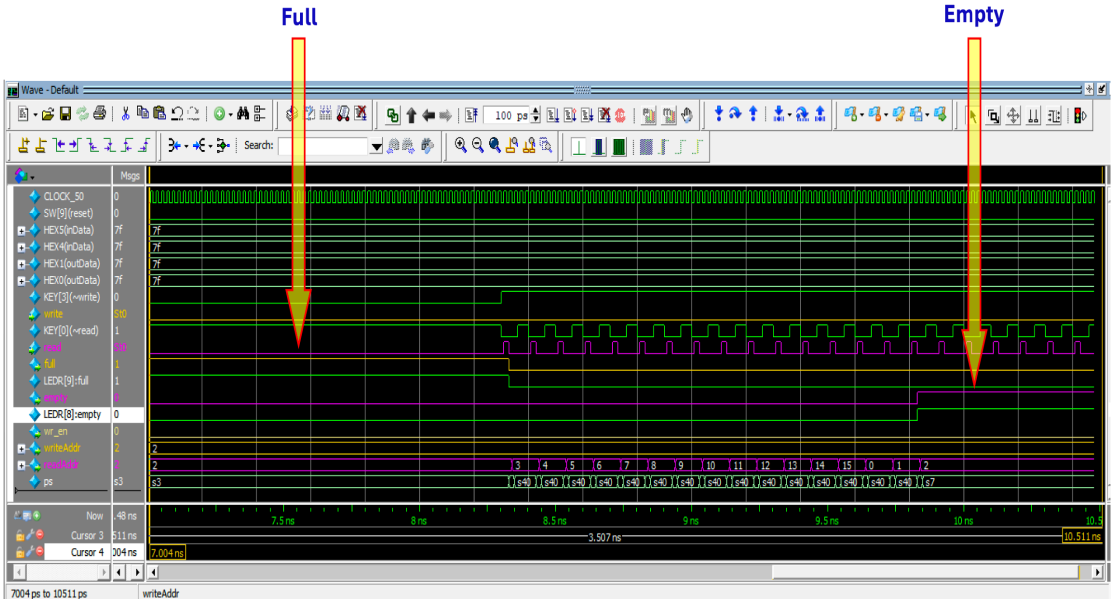

Figure 18: Simulation results of the design “DE1_SoC” in Task 3 (7 ns to 10.5 ns).

Figures 19 to 23 show the simulation results of the modules “clock_divider”, “data7seg”, “userInput”, “FIFO_Control”, and “FIFO”, respectively. Apparently, the simulation results indicate all the modules as mentioned in Task 3 perform well.

Simulation Result with the Testbench for “clock_divider.sv”:

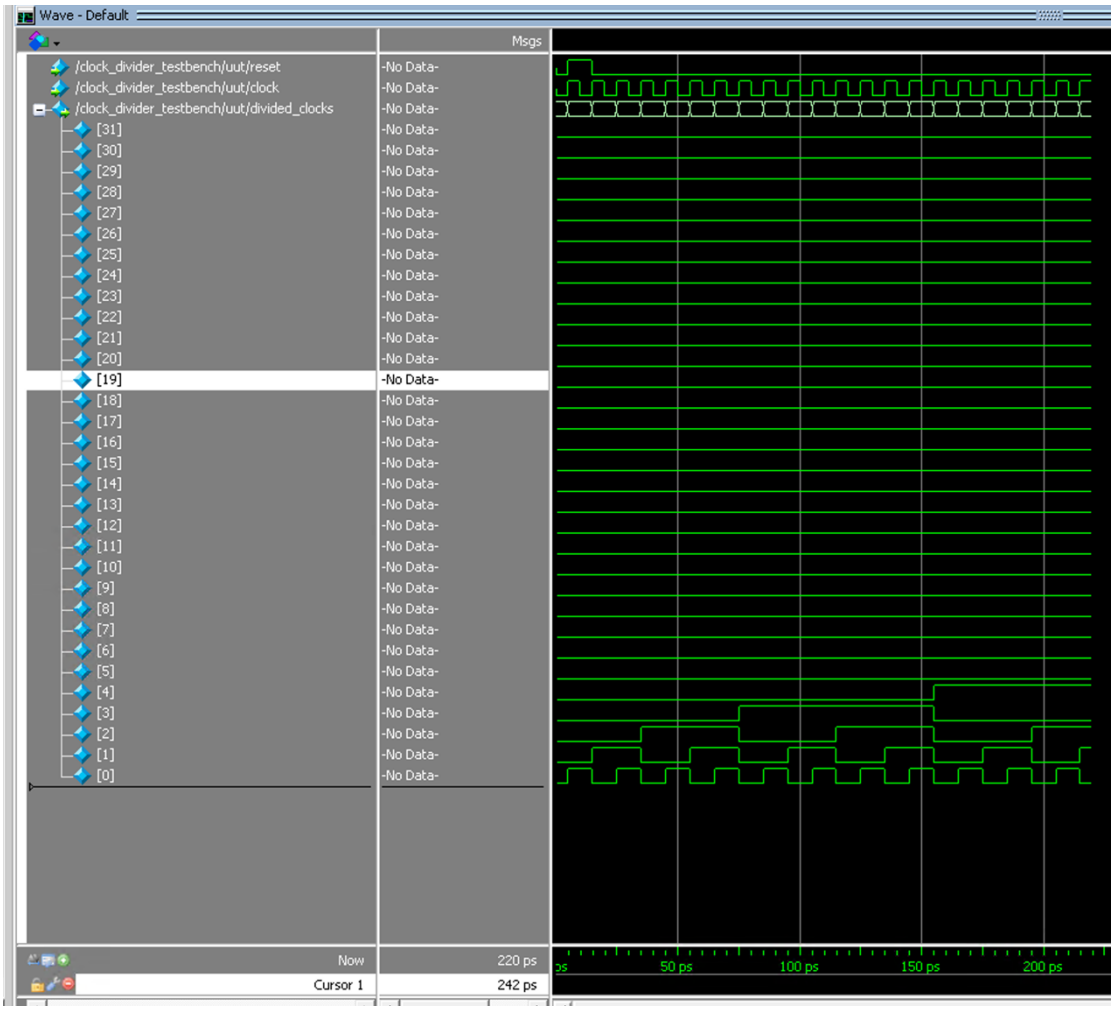

Figure 19: Simulation results of the design “clock_divider” in Task 3.

Simulation Result with the Testbench for “data7seg.sv”:

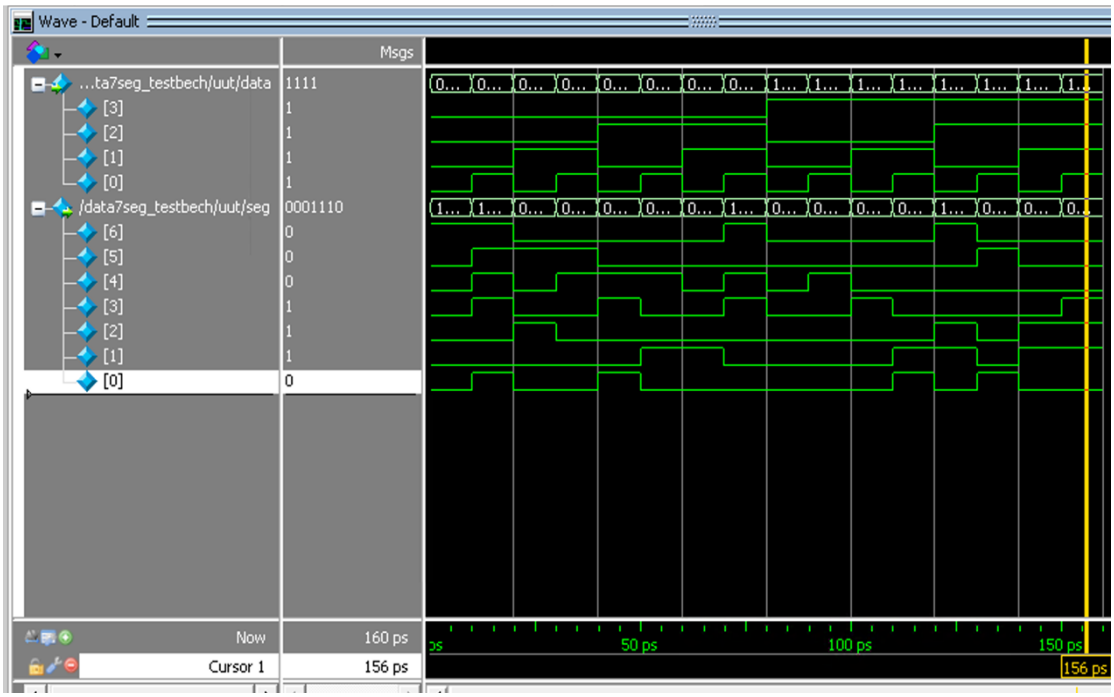

Figure 20: Simulation results of the design “data7seg” in Task 3.

Simulation Result with the Testbench for “userInput.sv”:

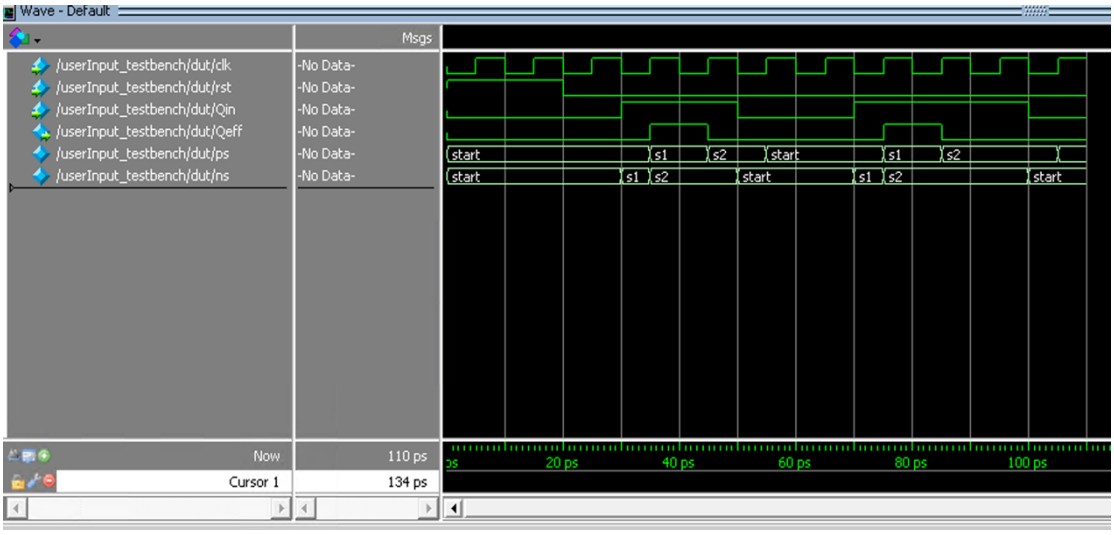

Figure 21: Simulation results of the design “userInput” in Task 3.

Simulation Result with the Testbench for "FIFO_Control.sv":

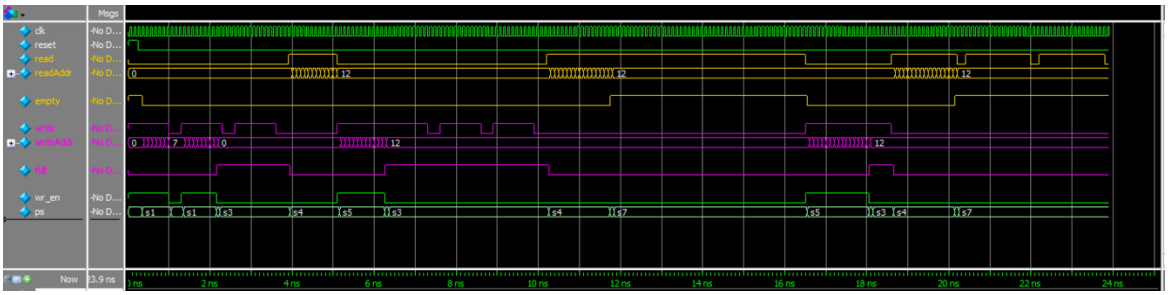

Figure 22: Simulation results of the design "FIFO_Control" in Task 3.

Simulation Result with the Testbench for "FIFO.sv":

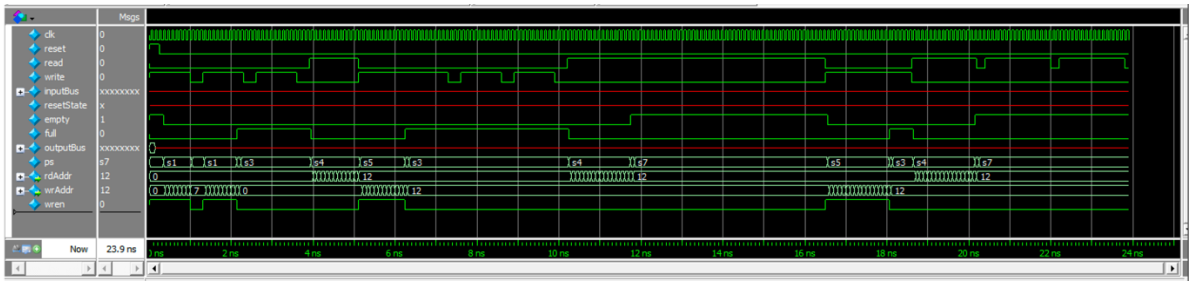

Figure 23: Simulation results of the design "FIFO" in Task 3.

III. Final Product

I finished the design specified in Task 1, Task 2, and Task 3. 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 addition to the specifications of Task 3 in the course handout, I applied HEX3 and HEX2 for displaying the write address and read address, respectively, as indicated in yellow in Figure 3. This arrangement enhances the readability of this design because I can track the current write address and current read address easily. As I know, monitoring the current write address and current read address is the key to the design in Task 3.

IV. Appendix

1. Video for Task 1, https://youtu.be/n9HCN49CvJw
2. Video for Task 2, https://youtu.be/knoLh0ORA6Y
3. Video for Task 3, https://youtu.be/V4jeYKumBSg
4. SystemVerilog codes of Task 1, please see the following pages.
5. SystemVerilog codes of Task 2, please see the following pages.
6. SystemVerilog codes of Task 3, please see the following pages.

TASK 1 DE1_SoC.sv:


```
1   /*=================================================================*/
2   //  Brian Chen
3   //  01/28/2025
4   //  LAB2 -- Memory Blocks (Task 1)
5   //  Device under Test (dut) --- DE1_SoC
6   //  File Name: DE1_SoC.sv
7   /*=================================================================*/
8   module DE1_SoC (SW, KEY, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
9
10  input logic [9:0] SW;
11  input logic [3:0] KEY;
12  output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
13  logic [3:0] data_out;
14
15  assign HEX1 = 7'h7f;
16  assign HEX3 = 7'h7f;
17
18   // Instantiate RAM module
19  brian_ram32x4 m1       (.clock(~KEY[0]),
20                          .reset(~KEY[3]),
21                          .write(SW[9]),
22                          .address(SW[8:4]),
23                          .data_in(SW[3:0]),
24                          .data_out);
25
26  data7seg      m2_data_out (.data(data_out),.seg(HEX0));
27  data7seg      m3_data_in  (.data(SW[3:0]), .seg(HEX2));
28
29  data7seg      m4_addrH    (.data(SW[8]),   .seg(HEX5));
30  data7seg      m5_addrL    (.data(SW[7:4]), .seg(HEX4));
31
32  endmodule
33
34  /*=================================================================*/
35  /*=================================================================*/
36  //  Testbench-----------------
    DE1_SoC_testbench

37  /*=================================================================*/
38  module DE1_SoC_testbench ();
39
40  logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
41  logic [3:0] KEY;
42  logic [9:0] SW;
43
44  integer i;
45
46  DE1_SoC dut (.SW, .KEY, .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);
47
48
49  parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
50
51  initial begin
52      KEY[0] <= 0;
53      forever #(CLOCK_PERIOD/2) KEY[0] <= ~KEY[0];
54  end
55
56
57  initial begin
58      KEY[3] <= 1'b0;
59      #20
60      KEY[3] <= 1'b1;
61  end
62
63  initial begin
64      SW[3:0] <= 4'b1000;
65        SW[9] <= 1'b1;      // write operation
66      SW[8:4] <= 5'b0_0000;
67
68      for (i = 0; i < 32; i = i + 1) begin
69          @(negedge KEY[0])   // <---- update data_in @ negedge clock
70          if (SW[3:0] == 4'b1000)
71              SW[3:0] <= 4'b0001;
72          else
73              SW[3:0] <= (SW[3:0] << 1);
```

```
74
75              SW[8:4] <= SW[8:4] + 1;  // <-- update address @ negedge clock
76          end
77
78          SW[3:0] <= 4'b1000;
79            SW[9] <= 1'b0;     // read operation
80          SW[8:4] <= 5'b0_0000;
81
82          for (i = 32; i < 80; i = i + 1) begin
83              @(negedge KEY[0])
84              SW[8:4] <= SW[8:4] + 1;
85          end
86
87          $stop;
88      end
89
90      endmodule
```

Task1 brian_ram32x4.sv:


```
1   /*=================================================================*/
2   //  Brian Chen
3   //  01/28/2025
4   //  LAB2 -- Memory Blocks (Task 1)
5   //  Device under Test (dut) --- brain_ram32x4
6   //  File Name: brian_ram32x4.sv
7   /*=================================================================*/
8   module brian_ram32x4 (clock, reset, write, address, data_in, data_out);
9
10  input logic clock, reset, write;
11  input logic [3:0] data_in;
12  input logic [4:0] address;
13  output logic [3:0] data_out;
14
15  logic [4:0] address_q;
16  logic [3:0] data_in_q;
17  logic write_q;
18
19  logic [3:0] memory_array [31:0];
20
21  always_ff @  (posedge clock or posedge reset)
22      if (reset) begin
23          address_q <= 5'b0_0000;
24          data_in_q <= 4'b0000;
25          write_q   <= 1'b0;
26      end
27      else begin
28          address_q <= address;
29          data_in_q <= data_in;
30          write_q   <= write;
31      end
32
33  always_ff @ (posedge clock) // write operation performed @ posedge clock
34      if (write_q)
35          memory_array [address_q] <= data_in_q;
36
37  assign data_out = memory_array[address_q];
38
39  endmodule
40
41  /*=================================================================*/
42  //  Testbench
43  /*=================================================================*/
44  module brian_ram32x4_testbench ();
45
46  logic clock, reset, write;
47  logic [3:0] data_in;
48  logic [4:0] address;
49  logic [3:0] data_out;
50
51  integer i;
52
53  brian_ram32x4 dut (.clock, .reset, .write, .address, .data_in, .data_out);
54
55  parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
56
57  initial begin
58      clock <= 0;
59      forever #(CLOCK_PERIOD/2) clock <= ~clock;
60  end
61
62  initial begin
63      reset <= 1'b1;
64
65      repeat(3)
66      @ (posedge clock);
67      reset <= 1'b0;
68  end
69
70  initial begin
71      data_in <= 4'b1000;
72        write <= 1'b1;      // write operation
73      address <= 5'b0_0000;
74
75      for (i = 0; i < 32; i = i + 1) begin
```

```
76            @(negedge clock)   // <---- update data_in @ negedge clock
77            if (data_in == 4'b1000)
78                data_in <= 4'b0001;
79            else
80                data_in <= (data_in << 1);
81
82            address <= address + 1;  // <-- update address @ negedge clock
83        end
84
85        data_in <= 4'b1000;
86          write <= 1'b0;   // set this for read operation
87        address <= 5'b0_0000;
88
89        for (i = 32; i < 65; i = i + 1) begin
90            @(negedge clock)
91            address <= address + 1;
92        end
93
94        $stop;
95    end
96
97    endmodule
```

Task1 data7seg.sv:


```
1   /*=================================================================*/
2   //  Brian Chen
3   //  01/28/2025
4   //  LAB2 -- Memory Blocks (Task 1)
5   //  Device under Test (dut) --- data7seg
6   //  File Name: data7seg.sv
7   /*=================================================================*/
8   module data7seg(data, seg);
9
10  input logic [3:0] data; // 4-bit input to represent a hexadecimal digit (0 to 15)
11  output logic [6:0] seg; // 7-bit output for 7-segment display encoding
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
32    // Always block to assign the correct segment pattern based on the input data
33  always_comb begin
34
35  // Using case statement to select the appropriate 7-segment encoding based on 'data'
36      case(data)
37          0:  seg = Chr_0;
38          1:  seg = Chr_1;
39          2:  seg = Chr_2;
40          3:  seg = Chr_3;
41
42          4:  seg = Chr_4;
43          5:  seg = Chr_5;
44          6:  seg = Chr_6;
45          7:  seg = Chr_7;
46
47          8:  seg = Chr_8;
48          9:  seg = Chr_9;
49         10:  seg = Chr_A;
50         11:  seg = Chr_b;
51
52         12:  seg = Chr_C;
53         13:  seg = Chr_d;
54         14:  seg = Chr_E;
55         15:  seg = Chr_F;
56        default: seg = Blank; // Default case: If input is out of expected range (0-15),
    display blank
57      endcase
58
59  end
60
61  endmodule
62
63
64  ///
65
66  module data7seg_testbech();
67  // Testbench signals
68      logic [3:0] data;
69      logic [6:0] seg;
70
71      // Instantiate the data7seg module
72      data7seg uut (
73          .data(data),
74          .seg(seg)
```

```
75        );
76
77        // Testbench stimulus generation
78        initial begin
79            // Display headers
80            $display("Time\tdata\tseg");
81
82            // Apply test vectors (0 to 15)
83            for (int i = 0; i < 16; i++) begin
84                data = i;           // Apply value to data
85                #10;                // Wait 10 time units (simulation time)
86                $display("%0t\t%0d\t%b", $time, data, seg);  // Display time, data, and seg
87            end
88
89            // End the simulation
90            $finish;
91        end
92
93    endmodule
94
```

```
75        );
76
77        // Testbench stimulus generation
78        initial begin
79            // Display headers
80            $display("Time\tdata\tseg");
81
82            // Apply test vectors (0 to 15)
83            for (int i = 0; i < 16; i++) begin
84                data = i;           // Apply value to data
85                #10;                // Wait 10 time units (simulation time)
86                $display("%0t\t%0d\t%b", $time, data, seg);  // Display time, data, and seg
87            end
88
89            // End the simulation
90            $finish;
91        end
92
93    endmodule
94
```

TASK2 DE1_SoC.sv:


```
1   /*=================================================================*/
2   // Brian Chen
3   // 01/28/2025
4   //  LAB2 -- Memory Blocks
5   //  Device under Test (dut) --- DE1_SoC
6   //  File Name: DE1_SoC.sv
7   /*=================================================================*/
8   `timescale 1ps/ 1ps
9
10  module DE1_SoC (CLOCK_50, SW, KEY, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
11
12  input logic CLOCK_50;
13  input logic [9:0] SW;
14  input logic [3:0] KEY;
15  output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
16  logic [4:0] rdaddress;
17  logic [3:0] q;
18
19      // Generate clk off of CLOCK_50, whichClock picks rate.
20  logic rst;
21  logic [31:0] div_clk;
22  logic [1:0] result;
23
24  parameter whichClock = 25; // 0.75 Hz clock  (50 MHz / 2^26 = 0.745 Hz)
25  /*=================================================================*/
26  //     clock_divider (File Name: clock_divider.sv)
27  /*=================================================================*/
28      clock_divider cdiv (.clock(CLOCK_50), .reset(~KEY[0]), .divided_clocks(div_clk));
29  /*-----------------------------------------------------------------*/
30  //     Clock selection;
31  //     allows for easy switching between simulation and board clocks
32  /*-----------------------------------------------------------------*/
33  logic clkSelect;
34
35      //Uncomment ONE of the following two lines depending on intention
36  //*****************************************************************
37     //assign clkSelect = CLOCK_50;           // for simulation
38     assign clkSelect = div_clk[whichClock]; // for DE1_SoC  board
39  //*****************************************************************
40
41   // Instantiate RAM module
42  ram32x4           m1 (.clock(clkSelect), .data(SW[3:0]), .rdaddress(rdaddress),
43                        .wraddress(SW[8:4]), .wren(~KEY[3]), .q);
44  counter5b         m2 (.clock(~clkSelect), .reset(~KEY[0]), .count(rdaddress));
45  address7seg       m3 (.address(rdaddress), .seg1(HEX3), .seg0(HEX2));
46  address7seg       m4 (.address(SW[8:4]), .seg1(HEX5), .seg0(HEX4));
47  data7seg          m5 (.data(SW[3:0]), .seg(HEX1));
48  data7seg          m6 (.data(q), .seg(HEX0));
49
50  endmodule
51
52  /*=================================================================*/
53  /*=================================================================*/
54  //  Testbench------------------
    DE1_SoC_testbench

55  /*=================================================================*/
56  module DE1_SoC_testbench ();
57
58  logic CLOCK_50;
59  logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
60  logic [3:0] KEY;
61  logic [9:0] SW;
62  logic wren, reset;
63
64  integer i;
65
66  DE1_SoC dut (.CLOCK_50, .SW, .KEY, .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);
67
68  assign KEY[3] = ~wren;
69  assign KEY[0] = ~reset;
70
71  // Set up a simulated clock: 50 MHz
72  parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
73
```

```
74   initial begin
75       CLOCK_50 <= 0;
76       forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
77   end
78
79   // Test the design.
80   initial begin
81       repeat(1)
82       @(posedge CLOCK_50);
83       reset <= 1;
84       wren  <= 0;
85       repeat(2)
86       @(posedge CLOCK_50);
87       reset <= 0;
88   end
89
90   initial begin
91       repeat(35)
92       @(posedge CLOCK_50);
93
94       wren <= 1;
95
96       for (i = 0; i < 8; i++) begin
97           @(negedge CLOCK_50);
98           SW[8:4] <= i;
99           SW[3:0] <= 4'b1111;
100      end
101
102      for (i = 8; i < 16; i++) begin
103          @(negedge CLOCK_50);
104          SW[8:4] <= i;
105          SW[3:0] <= 4'b1010;
106      end
107
108      for (i = 16; i < 24; i++) begin
109          @(negedge CLOCK_50);
110          SW[8:4] <= i;
111          SW[3:0] <= 4'b0100;
112      end
113
114      for (i = 24; i < 32; i++) begin
115          @(negedge CLOCK_50);
116          SW[8:4] <= i;
117          SW[3:0] <= 4'b1001;
118      end
119
120      @(negedge CLOCK_50);
121      wren  <= 0;
122
123      repeat(35)
124      @(posedge CLOCK_50);
125
126      $stop; // End the simulation.
127  end
128
129  endmodule
```

TASK2 ram32x4.sv:

```
// megafunction wizard: %RAM: 2-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: ram32x4.v
// Megafunction Name(s):
//          altsyncram
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.0.0 Build 595 04/25/2017 SJ Lite Edition
// ************************************************************


//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.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module ram32x4 (
    clock,
    data,
    rdaddress,
    wraddress,
    wren,
    q);

    input     clock;
    input   [3:0]  data;
    input   [4:0]  rdaddress;
    input   [4:0]  wraddress;
    input     wren;
    output  [3:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri1      clock;
    tri0      wren;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

    wire [3:0] sub_wire0;
    wire [3:0] q = sub_wire0[3:0];

    altsyncram  altsyncram_component (
                .address_a (wraddress),
                .address_b (rdaddress),
                .clock0 (clock),
                .data_a (data),
                .wren_a (wren),
                .q_b (sub_wire0),
                .aclr0 (1'b0),
```

```
                .aclr1 (1'b0),
                .addressstall_a (1'b0),
                .addressstall_b (1'b0),
                .byteena_a (1'b1),
                .byteena_b (1'b1),
                .clock1 (1'b1),
                .clocken0 (1'b1),
                .clocken1 (1'b1),
                .clocken2 (1'b1),
                .clocken3 (1'b1),
                .data_b ({4{1'b1}}),
                .eccstatus (),
                .q_a (),
                .rden_a (1'b1),
                .rden_b (1'b1),
                .wren_b (1'b0));
    defparam
        altsyncram_component.address_aclr_b = "NONE",
        altsyncram_component.address_reg_b = "CLOCK0",
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_input_b = "BYPASS",
        altsyncram_component.clock_enable_output_b = "BYPASS",
        altsyncram_component.init_file = "ram32x4.mif",
        altsyncram_component.intended_device_family = "Cyclone V",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.numwords_a = 32,
        altsyncram_component.numwords_b = 32,
        altsyncram_component.operation_mode = "DUAL_PORT",
        altsyncram_component.outdata_aclr_b = "NONE",
        altsyncram_component.outdata_reg_b = "UNREGISTERED",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.ram_block_type = "M10K",
        altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
        altsyncram_component.widthad_a = 5,
        altsyncram_component.widthad_b = 5,
        altsyncram_component.width_a = 4,
        altsyncram_component.width_b = 4,
        altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
// Retrieval info: PRIVATE: CLRdata NUMERIC "0"
// Retrieval info: PRIVATE: CLRq NUMERIC "0"
// Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
// Retrieval info: PRIVATE: CLRrren NUMERIC "0"
// Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
// Retrieval info: PRIVATE: CLRwren NUMERIC "0"
// Retrieval info: PRIVATE: Clock NUMERIC "0"
// Retrieval info: PRIVATE: Clock_A NUMERIC "0"
// Retrieval info: PRIVATE: Clock_B NUMERIC "0"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_B"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
```

```
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MEMSIZE NUMERIC "128"
// Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "ram32x4.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 "4"
// Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "4"
// Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "4"
// Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "4"
// 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 "ram32x4.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 "4"
// Retrieval info: CONSTANT: WIDTH_B NUMERIC "4"
// 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 4 0 INPUT NODEFVAL "data[3..0]"
// Retrieval info: USED_PORT: q 0 0 4 0 OUTPUT NODEFVAL "q[3..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 4 0 data 0 0 4 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 4 0 @q_b 0 0 4 0
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
```

TASK2 address7seg.sv:


```
/*=====================================================================*/
// Brian Chen
// 01/28/2025
//  LAB2 -- Memory Blocks
//  Device under Test (dut) --- address7seg
//  File Name: address7seg.sv
/*=====================================================================*/
module address7seg(address, seg1, seg0);

input logic [4:0] address;// 5-bit input for the address
output logic [6:0] seg1, seg0;// 7-bit output for 7-segment display encoding

parameter [6:0] Blank = 7'b111_1111;  // 7'h7f
parameter [6:0] Chr_0 = 7'b100_0000;  // 7'h40
parameter [6:0] Chr_1 = 7'b111_1001;  // 7'h79
parameter [6:0] Chr_2 = 7'b010_0100;  // 7'h24
parameter [6:0] Chr_3 = 7'b011_0000;  // 7'h30
parameter [6:0] Chr_4 = 7'b001_1001;  // 7'h19
parameter [6:0] Chr_5 = 7'b001_0010;  // 7'h12
parameter [6:0] Chr_6 = 7'b000_0010;  // 7'h02
parameter [6:0] Chr_7 = 7'b101_1000;  // 7'h58
parameter [6:0] Chr_8 = 7'b000_0000;  // 7'h00
parameter [6:0] Chr_9 = 7'b001_0000;  // 7'h10
parameter [6:0] Chr_A = 7'b000_1000;  // 7'h08
parameter [6:0] Chr_b = 7'b000_0011;  // 7'h03
parameter [6:0] Chr_C = 7'b100_0110;  // 7'h46
parameter [6:0] Chr_d = 7'b010_0001;  // 7'h21
parameter [6:0] Chr_E = 7'b000_0110;  // 7'h06
parameter [6:0] Chr_F = 7'b000_1110;  // 7'h0e


    // Always block to assign the correct segment pattern based on the input data

always_comb begin
case(address)
     0:  {seg1, seg0} = {Chr_0, Chr_0};
     1:  {seg1, seg0} = {Chr_0, Chr_1};
     2:  {seg1, seg0} = {Chr_0, Chr_2};
     3:  {seg1, seg0} = {Chr_0, Chr_3};

     4:  {seg1, seg0} = {Chr_0, Chr_4};
     5:  {seg1, seg0} = {Chr_0, Chr_5};
     6:  {seg1, seg0} = {Chr_0, Chr_6};
     7:  {seg1, seg0} = {Chr_0, Chr_7};

     8:  {seg1, seg0} = {Chr_0, Chr_8};
     9:  {seg1, seg0} = {Chr_0, Chr_9};
    10:  {seg1, seg0} = {Chr_0, Chr_A};
    11:  {seg1, seg0} = {Chr_0, Chr_b};

    12:  {seg1, seg0} = {Chr_0, Chr_C};
    13:  {seg1, seg0} = {Chr_0, Chr_d};
    14:  {seg1, seg0} = {Chr_0, Chr_E};
    15:  {seg1, seg0} = {Chr_0, Chr_F};

    16:  {seg1, seg0} = {Chr_1, Chr_0};
    17:  {seg1, seg0} = {Chr_1, Chr_1};
    18:  {seg1, seg0} = {Chr_1, Chr_2};
    19:  {seg1, seg0} = {Chr_1, Chr_3};

    20:  {seg1, seg0} = {Chr_1, Chr_4};
    21:  {seg1, seg0} = {Chr_1, Chr_5};
    22:  {seg1, seg0} = {Chr_1, Chr_6};
    23:  {seg1, seg0} = {Chr_1, Chr_7};

    24:  {seg1, seg0} = {Chr_1, Chr_8};
    25:  {seg1, seg0} = {Chr_1, Chr_9};
    26:  {seg1, seg0} = {Chr_1, Chr_A};
    27:  {seg1, seg0} = {Chr_1, Chr_b};

    28:  {seg1, seg0} = {Chr_1, Chr_C};
    29:  {seg1, seg0} = {Chr_1, Chr_d};
    30:  {seg1, seg0} = {Chr_1, Chr_E};
    31:  {seg1, seg0} = {Chr_1, Chr_F};
```

```
75      default: {seg1, seg0} = {Blank, Blank};
76  endcase
77  end
78
79  endmodule
80
81
82  module address7seg_testbench();
83
84      // Testbench signals
85      logic [4:0] address;         // 5-bit input for the address (hexadecimal value)
86      logic [6:0] seg1, seg0;      // 7-bit outputs for the two 7-segment encodings
87
88      // Instantiate the address7seg module (DUT - Device Under Test)
89      address7seg uut (
90          .address(address),  // Connect the testbench 'address' to the DUT 'address'
91          .seg1(seg1),        // Connect the testbench 'seg1' to the DUT 'seg1'
92          .seg0(seg0)         // Connect the testbench 'seg0' to the DUT 'seg0'
93      );
94
95      // Stimulus generation (apply different test cases to 'address' and observe the result)
96      initial begin
97          // Display headers for the testbench output
98          $display("Time\tAddress\tSeg1\t\tSeg0");
99
100         // Test cases for all address values (0 to 31)
101         // Apply each value of address (0 to 31) and monitor the seg1 and seg0 outputs
102         for (int i = 0; i < 32; i++) begin
103             address = i;             // Assign value to 'address'
104             #10;                     // Wait for 10 time units to simulate behavior
105             $display("%0t\t%0d\t\t%b\t%b", $time, address, seg1, seg0);  // Display the
    time, address, seg1, and seg0
106         end
107
108         // End the simulation
109         $finish;
110     end
111
112 endmodule
113
114
```

TASK2 data7seg.sv:


```
1   /*=================================================================*/
2   // Brian Chen
3   // 01/28/2025
4   //  LAB2 -- Memory Blocks
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  module data7seg_testbech();
60  // Testbench signals
61      logic [3:0] data;
62      logic [6:0] seg;
63
64      // Instantiate the data7seg module
65      data7seg uut (
66          .data(data),
67          .seg(seg)
68      );
69
70      // Testbench stimulus generation
71      initial begin
72          // Display headers
73          $display("Time\tdata\tseg");
74
75          // Apply test vectors (0 to 15)
```

```
76            for (int i = 0; i < 16; i++) begin
77                data = i;           // Apply value to data
78                #10;                // Wait 10 time units (simulation time)
79                $display("%0t\t%0d\t%b", $time, data, seg);  // Display time, data, and seg
80            end
81
82            // End the simulation
83            $finish;
84        end
85
86    endmodule
87
88
```

Task2 counter5b.sv:


```
1   /*=================================================================*/
2   // Brian Chen
3   // 01/28/2025
4   //  LAB2 -- Memory Blocks
5   //  Device under Test (dut)---
6   //  File Name: counter5b.sv
7   /*=================================================================*/
8   module counter5b (clock, reset, count);
9
10  input logic clock, reset;  //Input clock , reset signal
11
12  output logic [4:0] count; //5-bit counter output
13
14   // Always block that triggers on the rising edge of the clock or reset
15
16  always_ff @(posedge clock or posedge reset) begin
17      if (reset)
18          count <= 0; // If reset is active (high), set the counter to 0
19      else
20          count <= count + 1; // Otherwise, increment the counter on every clock cycle
21      end
22
23  endmodule
24  /*=================================================================*/
25  //   Testbench
26  /*=================================================================*/
27  module counter5b_testbench();
28
29  logic clk, rst;
30  logic [4:0] count;
31
32   // Instantiate the counter5b module (Device Under Test - DUT)
33
34  counter5b dut (.clock(clk), .reset(rst), .count);
35
36  parameter CLK_Period = 100;
37
38  initial begin
39      clk <= 1'b0;
40      forever #(CLK_Period/2) clk <= ~clk;
41  end
42
43  initial begin
44      rst <= 1;
45
46      repeat(3)
47      @(posedge clk);
48      rst <= 0;
49  end
50
51  initial begin
52      repeat(80)
53      @(posedge clk);
54
55      $stop;
56  end
57
58  endmodule
```

Task2 clock_divider.sv:


```
1   /*=================================================================*/
2   // Brian Chen
3   // 01/28/2025
4   //  LAB2 -- Memory Blocks
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  module clock_divider_testbench();
22
23      // Testbench signals
24      logic reset;           // Reset signal for the DUT (Device Under Test)
25      logic clock;           // Clock signal for the DUT
26      logic [31:0] divided_clocks;  // Output signal from the DUT
27
28      // Instantiate the clock_divider module (DUT)
29      clock_divider uut (
30          .clock(clock),               // Connect the testbench clock to DUT's clock
31          .reset(reset),               // Connect the testbench reset to DUT's reset
32          .divided_clocks(divided_clocks)  // Connect the DUT output to the testbench signal
33      );
34
35      // Clock generation (period = 10 time units for a frequency of 50MHz)
36      initial begin
37          clock = 0;
38          forever #5 clock = ~clock;  // Toggle clock every 5 time units to get a 50MHz clock
39      end
40
41      // Stimulus generation (reset and monitoring the divided clocks)
42      initial begin
43          // Initialize signals
44          reset = 0;
45
46          // Display headers for simulation output
47          $display("Time\tReset\tDivided Clocks");
48
49          // Reset the DUT
50          #5 reset = 1;   // Assert reset
51          #10 reset = 0;  // Deassert reset
52
53          // Monitor the divided clocks for some time after reset is deasserted
54          #5;
55          $display("%0t\t%b\t%h", $time, reset, divided_clocks);  // Display reset and
    divided clocks
56
57          // Let the clock run and observe the incrementing divided_clocks
58          #50;
59          $display("%0t\t%b\t%h", $time, reset, divided_clocks);  // Display the value
    after 50 time units
60
61          #50;
62          $display("%0t\t%b\t%h", $time, reset, divided_clocks);  // Display the value
    after another 50 time units
63
64          // Test longer simulation time to see clock divider behavior
65          #100;
66          $display("%0t\t%b\t%h", $time, reset, divided_clocks);  // Display the value
    after 100 time units
67
68          // End the simulation
69          $finish;
```

```
70        end
71
72    endmodule
73
```

TASK 3 DE1_SoC.sv:


```
1   /*=================================================================*/
2   //Brian Chen
3   // 01/28/2025
4   //  LAB2 -- Memory Blocks (Task 3)
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 [7:0] outputBus;
16  logic r_eff, w_eff;
17
18  assign LEDR[7:0] = 8'b0000_0000;
19  //assign HEX3 = 7'h7f;
20  //assign HEX2 = 7'h7f;
21
22  // Generate clk off of CLOCK_50, whichClock picks rate.
23  logic [31:0] div_clk;
24
25  parameter whichClock = 25; // 0.75 Hz clock  (50 MHz / 2^26 = 0.745 Hz)
26  /*=================================================================*/
27  //     clock_divider (File Name: clock_divider.sv)
28  /*=================================================================*/
29      clock_divider cdiv (.clock(CLOCK_50), .reset(SW[9]), .divided_clocks(div_clk));
30  /*-----------------------------------------------------------------*/
31  //     Clock selection;
32  //     allows for easy switching between simulation and board clocks
33  /*-----------------------------------------------------------------*/
34  logic clkSelect;
35
36      //Uncomment ONE of the following two lines depending on intention
37  //*************************************************************
38      assign clkSelect = CLOCK_50;           // for simulation
39      //assign clkSelect = div_clk[whichClock]; // for DE1_SoC  board
40  //*************************************************************
41  logic [3:0] wrAddr, rdAddr;
42
43
44  // Instantiate RAM module
45  FIFO #(4, 8)      m1 (.clk(clkSelect), .reset(SW[9]), .read(r_eff), .write(w_eff),
46                        .inputBus(SW[7:0]), .empty(LEDR[8]), .full(LEDR[9]), .outputBus,
47                        .wrAddr, .rdAddr // monitor write/read address
48                        );
49
50  data7seg          m2 (.data(SW[7:4]), .seg(HEX5));
51  data7seg          m3 (.data(SW[3:0]), .seg(HEX4));
52
53  data7seg          m4 (.data(outputBus[7:4]), .seg(HEX1));
54  data7seg          m5 (.data(outputBus[3:0]), .seg(HEX0));
55
56  userInput         m6 (.clk(clkSelect), .rst(SW[9]), .Qin(~KEY[3]), .Qeff(w_eff));
57  userInput         m7 (.clk(clkSelect), .rst(SW[9]), .Qin(~KEY[0]), .Qeff(r_eff));
58
59
60  // display wrAddr and rdAddr
61
62  data7seg          m8 (.data(wrAddr), .seg(HEX3));  // monitor write address
63  data7seg          m9 (.data(rdAddr), .seg(HEX2));  // monitor read address
64
65  endmodule
66
67  /*=================================================================*/
68  //  Testbench
69  /*=================================================================*/
70  `timescale 1ps/ 1ps
71  module DE1_SoC_testbench();
72
73  logic CLOCK_50;
74  logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
75  logic [3:0] KEY;
```

```
76   logic [9:0] SW;
77   logic [9:0] LEDR;
78
79   integer i;
80
81   DE1_SoC dut (.CLOCK_50, .SW, .KEY,
82                .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0, .LEDR);
83
84   // Set up a simulated clock: 50 MHz
85   parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
86
87   initial begin
88       CLOCK_50 <= 0;
89       forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
90   end
91
92   // Test the design.
93   initial begin
94       repeat(1)
95      @(posedge CLOCK_50);
96       SW[9] <= 1;
97       repeat(2)
98      @(posedge CLOCK_50);
99       SW[9] <= 0;
100  end
101
102  initial begin
103      for (i = 0; i < 20; i++) begin
104          @(negedge CLOCK_50);
105          {KEY[0], KEY[3]} <= 2'b10;
106
107          repeat(3)
108          @(negedge CLOCK_50);
109
110          {KEY[0], KEY[3]} <= 2'b11;
111
112          repeat(2)
113          @(negedge CLOCK_50);
114      end
115
116      for (i = 23; i < 26; i++) begin
117          @(negedge CLOCK_50);
118          {KEY[0], KEY[3]} <= 2'b01;
119
120          repeat(3)
121          @(negedge CLOCK_50);
122      end
123
124      for (i = 26; i < 29; i++) begin
125          @(negedge CLOCK_50);
126          {KEY[0], KEY[3]} <= 2'b10;
127
128          repeat(3)
129          @(negedge CLOCK_50);
130      end
131
132      for (i = 29; i < 32; i++) begin
133          @(negedge CLOCK_50);
134          {KEY[0], KEY[3]} <= 2'b10;
135
136          repeat(3)
137          @(negedge CLOCK_50);
138
139          {KEY[0], KEY[3]} <= 2'b11;
140
141          repeat(2)
142          @(negedge CLOCK_50);
143      end
144
145      for (i = 32; i < 55; i++) begin
146          @(negedge CLOCK_50);
147          {KEY[0], KEY[3]} <= 2'b01;
148
149          repeat(3)
150          @(negedge CLOCK_50);
```

```
151         end
152
153         for (i = 55; i < 78; i++) begin
154             @(negedge CLOCK_50);
155             {KEY[0], KEY[3]} <= 2'b10;
156
157             repeat(6)
158             @(negedge CLOCK_50);
159         end
160
161         for (i = 78; i < 100; i++) begin
162             @(negedge CLOCK_50);
163             {KEY[0], KEY[3]} <= 2'b01;
164
165             repeat(3)
166             @(negedge CLOCK_50);
167
168             {KEY[0], KEY[3]} <= 2'b11;
169
170             @(negedge CLOCK_50);
171         end
172
173         $stop; // End the simulation.
174     end
175
176     endmodule
```

TASK 3 ram16x8.sv:

```
// megafunction wizard: %RAM: 2-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: ram16x8.v
// Megafunction Name(s):
//          altsyncram
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.0.0 Build 595 04/25/2017 SJ Lite Edition
// ************************************************************


//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.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module ram16x8 (
    clock,
    data,
    rdaddress,
    wraddress,
    wren,
    q);

    input     clock;
    input   [7:0]  data;
    input   [3:0]  rdaddress;
    input   [3:0]  wraddress;
    input     wren;
    output  [7:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri1      clock;
    tri0      wren;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

    wire [7:0] sub_wire0;
    wire [7:0] q = sub_wire0[7:0];

    altsyncram  altsyncram_component (
                .address_a (wraddress),
                .address_b (rdaddress),
                .clock0 (clock),
                .data_a (data),
                .wren_a (wren),
                .q_b (sub_wire0),
                .aclr0 (1'b0),
```

```
                .aclr1 (1'b0),
                .addressstall_a (1'b0),
                .addressstall_b (1'b0),
                .byteena_a (1'b1),
                .byteena_b (1'b1),
                .clock1 (1'b1),
                .clocken0 (1'b1),
                .clocken1 (1'b1),
                .clocken2 (1'b1),
                .clocken3 (1'b1),
                .data_b ({8{1'b1}}),
                .eccstatus (),
                .q_a (),
                .rden_a (1'b1),
                .rden_b (1'b1),
                .wren_b (1'b0));
    defparam
        altsyncram_component.address_aclr_b = "NONE",
        altsyncram_component.address_reg_b = "CLOCK0",
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_input_b = "BYPASS",
        altsyncram_component.clock_enable_output_b = "BYPASS",
        altsyncram_component.intended_device_family = "Cyclone V",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.numwords_a = 16,
        altsyncram_component.numwords_b = 16,
        altsyncram_component.operation_mode = "DUAL_PORT",
        altsyncram_component.outdata_aclr_b = "NONE",
        altsyncram_component.outdata_reg_b = "UNREGISTERED",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.ram_block_type = "M10K",
        altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
        altsyncram_component.widthad_a = 4,
        altsyncram_component.widthad_b = 4,
        altsyncram_component.width_a = 8,
        altsyncram_component.width_b = 8,
        altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
// Retrieval info: PRIVATE: CLRdata NUMERIC "0"
// Retrieval info: PRIVATE: CLRq NUMERIC "0"
// Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
// Retrieval info: PRIVATE: CLRrren NUMERIC "0"
// Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
// Retrieval info: PRIVATE: CLRwren NUMERIC "0"
// Retrieval info: PRIVATE: Clock NUMERIC "0"
// Retrieval info: PRIVATE: Clock_A NUMERIC "0"
// Retrieval info: PRIVATE: Clock_B NUMERIC "0"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_B"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
```

```
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MEMSIZE NUMERIC "128"
// Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING ""
// 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: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "16"
// Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "16"
// 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 "4"
// Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "4"
// 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 4 0 INPUT NODEFVAL "rdaddress[3..0]"
// Retrieval info: USED_PORT: wraddress 0 0 4 0 INPUT NODEFVAL "wraddress[3..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT GND "wren"
// Retrieval info: CONNECT: @address_a 0 0 4 0 wraddress 0 0 4 0
// Retrieval info: CONNECT: @address_b 0 0 4 0 rdaddress 0 0 4 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 ram16x8.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
```

Task3 FIFO_Control.sv:


```
1   /*=====================================================================*/
2   //Brian Chen
3   // 01/28/2025
4   //  LAB2 -- Memory Blocks (Task 3)
5   //  Device under Test (dut) --- FIFO_Control
6   //  File Name: FIFO_Control_SoC.sv
7   /*=====================================================================*/
8   module FIFO_Control #(
9                         parameter depth = 4
10                        )(
11                          input logic clk, reset,
12                          input logic read, write,
13                        output logic wr_en,
14                        output logic empty, full,
15                        output logic [depth-1:0] readAddr, writeAddr
16                        );
17
18  /*    Define_Variables_Here      */
19  // State variables
20  typedef enum logic [3:0] {
21      s0, s1, s2, s3, s4, s5, s6, s7, s10, s40, s50
22      } state_t;
23      state_t ps, ns;
24
25  /*    Combinational_Logic_Here    */
26  // Next State logic
27  always_comb begin
28      ns = s0; // Default to avoid latches
29      case (ps)
30          s0: if ({read, write} == 2'b01) ns = s1;
31                  else ns = s0;
32
33          s1: if      (({read, write} == 2'b01) && ((writeAddr + 4'b0001) != readAddr)) ns
    = s1;
34              else if (({read, write} == 2'b01) && ((writeAddr + 4'b0001) == readAddr)) ns
    = s2;
35              else if (({read, write} == 2'b10) && ((readAddr + 4'b0001) != writeAddr)) ns
    = s4;
36              else if (({read, write} == 2'b10) && ((readAddr + 4'b0001) == writeAddr)) ns
    = s6;
37              else    ns = s10;
38
39          s2: if      ({read, write} == 2'b10) ns = s4;
40              else    ns = s3;
41
42          s3: if      ({read, write} == 2'b10) ns = s4;
43              else    ns = s3;
44
45          s4: if      (({read, write} == 2'b01) && ((writeAddr + 4'b0001) != readAddr)) ns
    = s5;
46              else if (({read, write} == 2'b01) && ((writeAddr + 4'b0001) == readAddr)) ns
    = s2;
47              else if (({read, write} == 2'b10) && ((readAddr + 4'b0001) != writeAddr)) ns
    = s4;
48              else if (({read, write} == 2'b10) && ((readAddr + 4'b0001) == writeAddr)) ns
    = s6;
49              else      ns = s40;
50
51          s5: if      (({read, write} == 2'b01) && ((writeAddr + 4'b0001) != readAddr)) ns
    = s5;
52              else if (({read, write} == 2'b01) && ((writeAddr + 4'b0001) == readAddr)) ns
    = s2;
53              else if (({read, write} == 2'b10) && ((readAddr + 4'b0001) != writeAddr)) ns
    = s4;
54              else if (({read, write} == 2'b10) && ((readAddr + 4'b0001) == writeAddr)) ns
    = s6;
55              else    ns = s50;
56
57          s6: if      ({read, write} == 2'b01) ns = s5;
58              else    ns = s7;
59
60          s7: if      ({read, write} == 2'b01) ns = s5;
61              else    ns = s7;
62
63          s10: if       (({read, write} == 2'b01) && ((writeAddr + 4'b0001) != readAddr))
```

```
      ns = s1;
64                else if  (({read, write} == 2'b01) && ((writeAddr + 4'b0001) == readAddr))
      ns = s2;
65                else if  (({read, write} == 2'b10) && ((readAddr + 4'b0001) != writeAddr))
      ns = s4;
66                else if  (({read, write} == 2'b10) && ((readAddr + 4'b0001) == writeAddr))
      ns = s6;
67                else      ns = s10;
68
69            s40: if       (({read, write} == 2'b01) && ((writeAddr + 4'b0001) != readAddr))
      ns = s5;
70                else if  (({read, write} == 2'b01) && ((writeAddr + 4'b0001) == readAddr))
      ns = s2;
71                else if  (({read, write} == 2'b10) && ((readAddr + 4'b0001) != writeAddr))
      ns = s4;
72                else if  (({read, write} == 2'b10) && ((readAddr + 4'b0001) == writeAddr))
      ns = s6;
73                else      ns = s40;
74
75            s50: if       (({read, write} == 2'b01) && ((writeAddr + 4'b0001) != readAddr))
      ns = s5;
76                else if  (({read, write} == 2'b01) && ((writeAddr + 4'b0001) == readAddr))
      ns = s2;
77                else if  (({read, write} == 2'b10) && ((readAddr + 4'b0001) != writeAddr))
      ns = s4;
78                else if  (({read, write} == 2'b10) && ((readAddr + 4'b0001) == writeAddr))
      ns = s6;
79                else      ns = s50;
80
81            default: ns = s0;
82        endcase
83    end
84
85
86    always_comb begin
87        case(ps)
88            s0: {empty, full} = 2'b10;
89            s1: {empty, full} = 2'b00;
90            s2: {empty, full} = 2'b01;
91            s3: {empty, full} = 2'b01;
92
93            s4: {empty, full} = 2'b00;
94            s5: {empty, full} = 2'b00;
95            s6: {empty, full} = 2'b10;
96            s7: {empty, full} = 2'b10;
97
98            s10: {empty, full} = 2'b00;
99            s40: {empty, full} = 2'b00;
100           s50: {empty, full} = 2'b00;
101       endcase
102   end
103
104   always_ff @(negedge clk) begin // use negedge for readAddr / writeAddr
105       if(reset) begin
106           readAddr <= '0;
107           writeAddr <= '0;
108       end
109       else begin
110           case (ps)
111               s0: begin
112                       readAddr <= 0;
113                      writeAddr <= 0;
114                   end
115
116               s1: begin
117                       readAddr <= 0;
118                      writeAddr <= writeAddr + 4'b0001;
119                   end
120
121               s2: begin
122                       readAddr <= readAddr;
123                      writeAddr <= writeAddr + 4'b0001;
124                   end
125
126               s3: begin
```

```
127                         readAddr <= readAddr;
128                        writeAddr <= writeAddr;
129                     end
130
131                 s4: begin
132                         readAddr <= readAddr + 4'b0001;
133                        writeAddr <= writeAddr;
134                     end
135
136                 s5: begin
137                         readAddr <= readAddr;
138                        writeAddr <= writeAddr + 4'b0001;
139                     end
140
141                 s6: begin
142                         readAddr <= readAddr + 4'b0001;
143                        writeAddr <= writeAddr;
144                     end
145
146                 s7: begin
147                         readAddr <= readAddr;
148                        writeAddr <= writeAddr;
149                     end
150
151                s10: begin
152                         readAddr <= readAddr;
153                        writeAddr <= writeAddr;
154                     end
155
156                s40: begin
157                         readAddr <= readAddr;
158                        writeAddr <= writeAddr;
159                     end
160
161                s50: begin
162                         readAddr <= readAddr;
163                        writeAddr <= writeAddr;
164                     end
165
166                default: begin
167                         readAddr <= readAddr;
168                        writeAddr <= writeAddr;
169                        end
170            endcase
171        end
172    end
173
174
175            // DFFs
176    always_ff @(posedge clk or posedge reset) begin
177        if (reset)
178            ps <= s0;
179        else
180            ps <= ns;
181        end
182
183    assign wr_en = write & (~full);
184
185    endmodule
186
187    /*=====================================================================*/
188    //  Testbench
189    /*=====================================================================*/
190
191    module FIFO_Control_testbench ();
192
193    parameter depth = 4, width = 8;
194    logic clk, reset;
195    logic read, write;
196    logic empty, full;
197    logic wr_en;
198    logic [3:0] readAddr, writeAddr;
199
200    integer i;
201
```

```
202   FIFO_Control #(depth) dut
203                (.clk, .reset, .read, .write, .wr_en, .empty, .full, .readAddr, .writeAddr);
204
205   parameter CLK_Period = 100;
206
207   initial begin
208       clk <= 1'b0;
209       forever #(CLK_Period/2) clk <= ~clk;
210   end
211
212   initial begin
213       reset <= 1;
214
215       repeat(3)
216       @(posedge clk);
217
218       reset <= 0;
219   end
220
221   initial begin
222       for (i = 0; i < 3; i++) begin
223           @(negedge clk);
224           {read, write} <= 2'b01;
225
226           repeat(10)
227           @(negedge clk);
228
229           {read, write} <= 2'b00;
230           repeat(2)
231           @(negedge clk);
232        end
233
234        for (i = 3; i < 6; i++) begin
235            @(negedge clk);
236            {read, write} <= 2'b10;
237
238            repeat(3)
239            @(negedge clk);
240        end
241
242        for (i = 6; i < 9; i++) begin
243            @(negedge clk);
244            {read, write} <= 2'b01;
245
246            repeat(3)
247            @(negedge clk);
248        end
249
250        for (i = 9; i < 12; i++) begin
251            @(negedge clk);
252            {read, write} <= 2'b01;
253
254            repeat(10)
255            @(negedge clk);
256
257            {read, write} <= 2'b00;
258
259            repeat(2)
260            @(negedge clk);
261        end
262
263        for (i = 12; i < 15; i++) begin
264            @(negedge clk);
265            {read, write} <= 2'b10;
266
267            repeat(20)
268            @(negedge clk);
269        end
270
271        for (i = 15; i < 18; i++) begin
272            @(negedge clk);
273            {read, write} <= 2'b01;
274
275            repeat(6)
276            @(negedge clk);
```

```
277         end
278
279         for (i = 12; i < 15; i++) begin
280             @(negedge clk);
281             {read, write} <= 2'b10;
282
283             repeat(16)
284             @(negedge clk);
285
286             {read, write} <= 2'b00;
287             @(negedge clk);
288         end
289
290         $stop;
291     end
292     endmodule
293
294
295
296
297
```

Task3 FIFO.sv:


```
1   /*=================================================================*/
2   //Brian Chen
3   // 01/28/2025
4   //  LAB2 -- Memory Blocks (Task 3)
5   //  Device under Test (dut) --- FIFO
6   //  File Name: FIFO.sv
7   /*=================================================================*/
8   module FIFO #(
9                 parameter depth = 4,
10                parameter width = 8
11                )(
12                  input logic clk, reset,
13                  input logic read, write,
14                  input logic [width-1:0] inputBus,
15                 output logic empty, full,
16                 output logic [width-1:0] outputBus,
17                 output logic [3:0] wrAddr,   // monitor write address
18                 output logic [3:0] rdAddr     // monitor read address
19                 );
20
21     /*    Define_Variables_Here      */
22
23  //=====================================
24        //logic [3:0] rdAddr, wrAddr;  <--comment for monitoring write/read address
25  //=====================================
26
27  logic wren;
28
29     /*       Instantiate_Your_Dual-Port_RAM_Here        */
30
31  ram16x8 m1 (.clock(clk), .data(inputBus), .rdaddress(rdAddr), .wraddress(wrAddr),
    .wren(wren), .q(outputBus));
32
33     /*       FIFO-Control_Module         */
34  FIFO_Control #(depth) FC (.clk, .reset,
35                            .read,
36                            .write,
37                            .wr_en(wren),
38                            .empty,
39                            .full,
40                            .readAddr(rdAddr),
41                            .writeAddr(wrAddr)
42                              );
43  endmodule
44
45  /*=================================================================*/
46  //  Testbench
47  /*=================================================================*/
48  `timescale 1ps/1ps
49  module FIFO_testbench();
50
51  parameter depth = 4, width = 8;
52
53  logic clk, reset;
54  logic read, write;
55  logic [width-1:0] inputBus;
56  logic resetState;
57  logic empty, full;
58  logic [width-1:0] outputBus;
59
60  //monitor write/read address
61  logic [3:0] wrAddr, rdAddr; // monitor write/read address
62
63
64  integer i;
65
66  FIFO #(depth, width) dut (.*);
67
68  parameter CLK_Period = 100;
69
70  initial begin
71      clk <= 1'b0;
72      forever #(CLK_Period/2) clk <= ~clk;
73  end
74
```

```
 75  initial begin
 76      reset <= 1;
 77
 78      repeat(3)
 79      @(posedge clk);
 80
 81      reset <= 0;
 82  end
 83
 84  initial begin
 85      for (i = 0; i < 3; i++) begin
 86          @(negedge clk);
 87          {read, write} <= 2'b01;
 88
 89          repeat(10)
 90          @(negedge clk);
 91
 92          {read, write} <= 2'b00;
 93
 94          repeat(2)
 95          @(negedge clk);
 96      end
 97
 98      for (i = 3; i < 6; i++) begin
 99          @(negedge clk);
100         {read, write} <= 2'b10;
101
102          repeat(3)
103          @(negedge clk);
104      end
105
106      for (i = 6; i < 9; i++) begin
107          @(negedge clk);
108          {read, write} <= 2'b01;
109
110          repeat(3)
111          @(negedge clk);
112      end
113
114      for (i = 9; i < 12; i++) begin
115          @(negedge clk);
116          {read, write} <= 2'b01;
117
118          repeat(10)
119          @(negedge clk);
120
121          {read, write} <= 2'b00;
122
123          repeat(2)
124          @(negedge clk);
125      end
126
127      for (i = 12; i < 15; i++) begin
128          @(negedge clk);
129          {read, write} <= 2'b10;
130
131          repeat(20)
132          @(negedge clk);
133      end
134
135      for (i = 15; i < 18; i++) begin
136          @(negedge clk);
137          {read, write} <= 2'b01;
138
139          repeat(6)
140          @(negedge clk);
141      end
142
143      for (i = 12; i < 15; i++) begin
144          @(negedge clk);
145          {read, write} <= 2'b10;
146
147          repeat(16)
148          @(negedge clk);
149
```

```
150             {read, write} <= 2'b00;
151             @(negedge clk);
152         end
153
154         $stop;
155     end
156
157     endmodule
```

Task3 usuerInput.sv:


```
1   /*=================================================================*/
2   //Brian Chen
3   // 01/28/2025
4   //  LAB2 -- Memory Blocks (Task 3)
5   //  Device under Test (dut) --- userInput
6   //  File Name: userInput.sv
7   /*=================================================================*/
8   module userInput (clk, rst, Qin, Qeff);
9   input logic clk, rst;
10  input logic Qin;
11  output logic Qeff;
12
13  // State variables
14      typedef enum logic [1:0] {
15          start, s1, s2
16      } state_t;
17      state_t ps, ns;
18
19      // Next State logic
20  always_comb begin
21      ns = start; // Default to avoid latches
22      case (ps)
23          start: if (Qin == 1) ns = s1;
24                 else ns = start;
25
26          s1:    if (Qin == 1) ns = s2;
27                 else ns = start;
28
29          s2:    if (Qin == 1) ns = s2;
30                 else ns = start;
31          default: ns = start;
32      endcase
33  end
34
35      // Output logic
36  always_comb begin
37      Qeff = 0;  // Default to avoid latches
38      case (ps)
39          start:   Qeff = 0;
40             s1:   Qeff = 1;
41             s2:   Qeff = 0;
42          default: Qeff = 0;
43      endcase
44  end
45
46      // DFFs
47  always_ff @(posedge clk or posedge rst) begin
48      if (rst)
49          ps <= start;
50      else
51          ps <= ns;
52      end
53
54  endmodule
55
56
57
58  /*=================================================================*/
59  // Testbench for userInput module
60  /*=================================================================*/
61  module userInput_testbench();
62
63      // Declare signals
64      logic clk, rst;              // Clock and reset signals
65      logic Qin;                   // Input signal
66      logic Qeff;                  // Output signal
67
68      // Instantiate the Device Under Test (DUT)
69      userInput dut (
70          .clk(clk),   // Connect clock
71          .rst(rst),   // Connect reset
72          .Qin(Qin),   // Connect Qin input
73          .Qeff(Qeff)  // Connect Qeff output
74      );
75
```

```
76      // Clock generation: Generate a clock signal with period 10 time units
77      parameter CLK_PERIOD = 10;
78
79      initial begin
80          clk = 0;
81          forever #(CLK_PERIOD / 2) clk = ~clk;  // Toggle clock every half period
82      end
83
84      // Reset generation: Assert reset at the beginning, then de-assert it after 2 clock
    cycles
85      initial begin
86          rst = 1;                // Assert reset at the start
87          #(CLK_PERIOD * 2);      // Wait for 2 clock cycles
88          rst = 0;                // De-assert reset
89      end
90
91      // Test sequence: Stimulate the Qin input and observe the Qeff output
92      initial begin
93          // Initial state with reset, Qeff should be 0
94          Qin = 0;                // Set Qin to 0
95          #(CLK_PERIOD * 3);      // Wait for 3 clock cycles
96
97          // Test 1: Transition from start -> s1 -> s2 -> start
98          Qin = 1;                // Set Qin to 1, should transition to s1
99          #(CLK_PERIOD);          // Wait 1 clock cycle
100         Qin = 1;                // Qin still 1, should transition to s2
101         #(CLK_PERIOD);          // Wait 1 clock cycle
102         Qin = 0;                // Set Qin to 0, should transition to start
103         #(CLK_PERIOD * 2);      // Wait for 2 clock cycles
104
105         // Test 2: Transition back to s1 and stay in s2
106         Qin = 1;                // Qin 1, should transition to s1
107         #(CLK_PERIOD);          // Wait 1 clock cycle
108         Qin = 1;                // Qin still 1, should transition to s2
109         #(CLK_PERIOD * 2);      // Wait for 2 clock cycles
110
111         // Test 3: Return to start state when Qin is 0
112         Qin = 0;                // Set Qin to 0, should transition to start
113         #(CLK_PERIOD);          // Wait 1 clock cycle
114
115         // Stop the simulation after a few cycles
116         $stop;                  // Stop simulation after the tests
117     end
118
119     // Monitor: Observe the signals during the simulation
120     initial begin
121         $monitor("Time = %0t | Qin = %b | Qeff = %b" , $time, Qin, Qeff);
122     end
123
124 endmodule
125
```