UW Student: Brian Chen

EE 371

February 28, 2025

Lab5 Report

I. Procedure

Task 1 and Task 2:

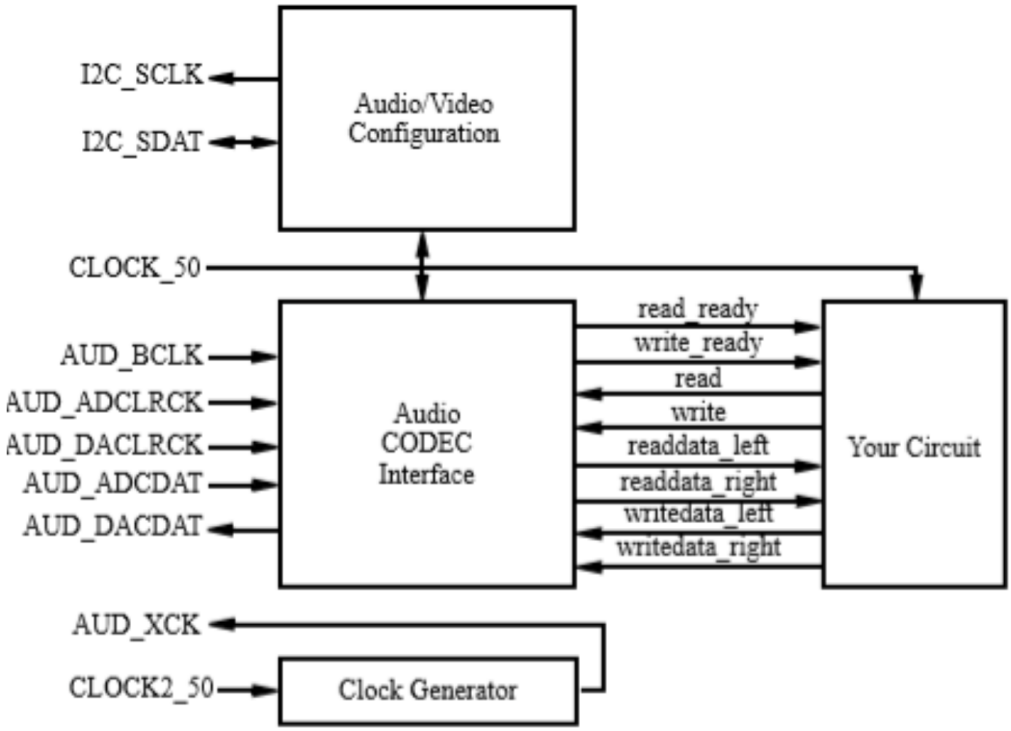

Figure 1: The audio system used in this Lab. (This picture is from handout of Lab 5.)

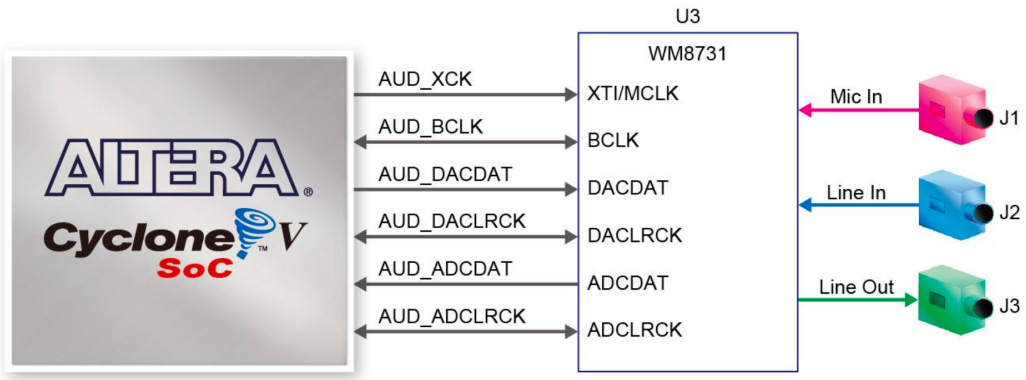

Figure 2: Connections between FPGA and Audio CODEC. (This picture is from DE1-SoC User Manual.)

Figure 1 shows the audio system used in this Lab. Figure 2 shows the onnections between FPGA and Audio CODEC. According to the guidance in Lab handout, I mode some modifications on the SystemVerilog codes in part1.v and part2.v as shown in Figures 3 and 4, respectively. In this design, the inputs of DE1_SoC are arranged as follows. KEY[0] is for the reset signal; SW[9] is for selecting the audio sources either from the audio file "piano_noisy.mp3" or the tone stored in memory. In order to arrange this task, I created the file "note_data.mif" by python script provided by the starterkit. According to the memory contents specified in "note_data.mif", I created a 48000-word x 24-bit ROM called "rom_lab5.v" for storing the tone. I used a counter to generate the running addresses for reading the data from memory.

After writing the SystemVerilog codes, I simulated this design on Modelsim-Altera and verifyied the functions on LabsLand.

```
31      // Send the data from mic to speakers.
32      assign writedata_left = readdata_left;
33      assign writedata_right = readdata_right;
34      assign read = read_ready ? 1 : 0;
35      assign write = write_ready ? 1 : 0;
```

Figure 3: Modifications in part1.sv.

```
31      // Generate clk off of CLOCK_50, whichClock picks rate.
32         wire [31:0] div_clk;
33         parameter whichClock = 9; // 48.8 kHz clock  (50 MHz / 2^10 = 48.8 kHz)
34      /*=================================================================*/
35      //     clock_divider (File Name: clock_divider.sv)
36      /*=================================================================*/
37         clock_divider cdiv (.clock(CLOCK_50), .reset(reset), .divided_clocks(div_clk));
38      /*-----------------------------------------------------------------*/
39      //     Clock selection;
40      //     allows for easy switching between simulation and board clocks
41      /*-----------------------------------------------------------------*/
42         wire clkSelect;
43      //Uncomment ONE of the following two lines depending on intention
44      //*****************************************************************
45        // assign clkSelect = CLOCK_50;           // for simulation
46        assign clkSelect = div_clk[whichClock];  // for DE1_SoC  board
47      //*****************************************************************
48      wire [23:0] readdata;
49      wire [15:0] address;
50
51      // Instantiate rom_lab5.
52      rom_lab5 m1     (.address(address), .clock(clkSelect), .q(readdata));
53
54      // Instantiate counter to generate the addresses for memory read operations.
55      counter  m2     (.clock(clkSelect), .reset(reset), .count(address));
56
57      // Send the data read from rom_lab5 to speakers.
58      assign writedata_left = readdata;
59      assign writedata_right = readdata;
60      assign read = read_ready ? 1 : 0;
61      assign write = write_ready ? 1 : 0;
```

Figure 4 (a): Modifications in part2.sv.

```
33      // Use SW[9] to select the output signals to DE1_SoC.
34      assign   FPGA_I2C_SCLK = SW[9] ? FPGA_I2C_SCLK2 : FPGA_I2C_SCLK1;
35      assign   AUD_XCK       = SW[9] ? AUD_XCK2 : AUD_XCK1;
36      assign   AUD_DACDAT    = SW[9] ? AUD_DACDAT2  : AUD_DACDAT1 ;
```

Figure 4 (b): Modifications in part2.sv.

```
38      // Instantiate part1
39      part1 p1 (.CLOCK_50,
40                .CLOCK2_50,
41                .KEY,
42
43                 // Rename FPGA_I2C_SCLK as FPGA_I2C_SCLK1
44                .FPGA_I2C_SCLK(FPGA_I2C_SCLK1), //<---This line.
45                .FPGA_I2C_SDAT,
46
47                 // Rename AUD_XCK as AUD_XCK1.
48                .AUD_XCK(AUD_XCK1), //<---This line.
49                .AUD_DACLRCK,
50                .AUD_ADCLRCK,
51                .AUD_BCLK,
52                .AUD_ADCDAT,
53
54                 // Rename AUD_DACDAT as AUD_DACDAT1.
55                .AUD_DACDAT(AUD_DACDAT1) //<---This line.
56                );
57
58       // Instantiate part2
59       part2 p2 (.CLOCK_50,
60                 .CLOCK2_50,
61                 .KEY,
62
63                  // Rename FPGA_I2C_SCLK as FPGA_I2C_SCLK2.
64                 .FPGA_I2C_SCLK(FPGA_I2C_SCLK2),  //<---This line.
65                 .FPGA_I2C_SDAT,
66
67                  // Rename AUD_XCK as AUD_XCK2.
68                 .AUD_XCK(AUD_XCK2), //<---This line.
69                 .AUD_DACLRCK,
70                 .AUD_ADCLRCK,
71                 .AUD_BCLK,
72                 .AUD_ADCDAT,
73
74                  // Rename AUD_DACDAT as AUD_DACDAT2.
75                 .AUD_DACDAT(AUD_DACDAT2)  //<---This line.
76                 );
```

Figure 4(c): Modifications in part2.sv.

Task 3: FIR Filter

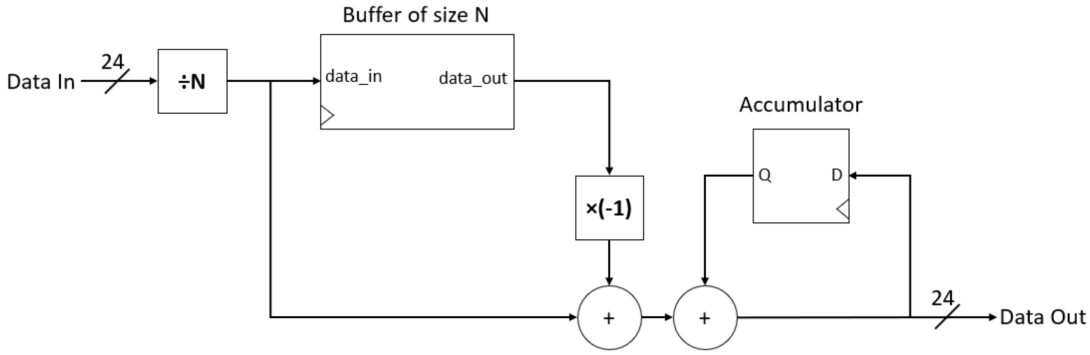

Figure 5: N-sample averaging FIR filter using a FIFO buffer and accumulator that you will implement in Task #3. (This picture is from handout of Lab 5.)

Figure 5 shows t N-sample averaging FIR filter using a FIFO buffer and accumulator. I wrote SystemVerilog codes to implement this FIR filter. According to the guidance in Lab handout, I mode some modifications on the SystemVerilog codes in part1.v and part2.v as shown in Figures 6 and 7, respectively. In this design, the inputs of DE1_SoC are arranged as follows. KEY[0] is for the reset signal; SW[9] is for selecting the audio sources either from the audio file "piano_noisy.mp3" or the tone stored in memory. ; SW[8] is for selecting the filtered signal or the unfiltered signal. After writing the SystemVerilog codes, I simulated this design on Modelsim-Altera and verifyied the functions on LabsLand.

```
35    // Instantiate FIR Filter for part1 (left-hand source).
36    fir_filter #(DATA_WIDTH, ADDR_WIDTH) p3
37              (.clk(CLOCK_50),
38               .reset(~KEY[0]),
39               .wr(1'b1),
40               .data_in(readdata_left),
41               .data_out(readdata_left_FIR)
42               );
43    // Instantiate FIR Filter for part1 (right-hand source).
44    fir_filter #(DATA_WIDTH, ADDR_WIDTH) p4
45              (.clk(CLOCK_50),
46               .reset(~KEY[0]),
47               .wr(1'b1),
48               .data_in(readdata_right),
49               .data_out(readdata_right_FIR)
50               );
51
52    // Send the data from mic to speakers.
53    // Use SW[8] to select filtered or unfiltered.
54    assign writedata_left  = SW[8] ? readdata_left_FIR  : readdata_left;
55    assign writedata_right = SW[8] ? readdata_right_FIR : readdata_right;
56    assign read = read_ready ? 1 : 0;
57    assign write = write_ready ? 1 : 0;
```

Figure 6: Modifications in part1.sv in Task3.

```
54    // Instantiate rom_lab5.
55    rom_lab5 m1     (.address(address), .clock(clkSelect), .q(readdata));
56
57    // Instantiate counter to generate the addresses for memory read operations.
58    counter  m2     (.clock(clkSelect), .reset(reset), .count(address));
59
60    // Instantiate FIR Filter for part2.
61    fir_filter #(DATA_WIDTH, ADDR_WIDTH) p3
62              (.clk(CLOCK_50),
63               .reset(~KEY[0]),
64               .wr(1'b1),
65               .data_in(readdata),
66               .data_out(readdata_FIR)
67               );
68
69    // Send the data read from rom_lab5 to speakers.
70    // Use SW[8] to select filtered or unfiltered.
71    assign writedata_left  = SW[8] ? readdata_FIR  : readdata;
72    assign writedata_right = SW[8] ? readdata_FIR  : readdata;
73    assign read = read_ready ? 1 : 0;
74    assign write = write_ready ? 1 : 0;
```

Figure 7: Modifications in part2.sv in Task 3.

II. Result

Task 1&2:

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

I wrote the testbench to test the top module in Task 1 and Task 2. Figures 9 and 10 are the simulation results of "DE1_SoC.sv" and "part2.v", respectively. 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 $50 \times \frac{10^6}{2^{10}}$ Hz, that is 48.8 kHz. During the verification on LabsLand, this design performs well.

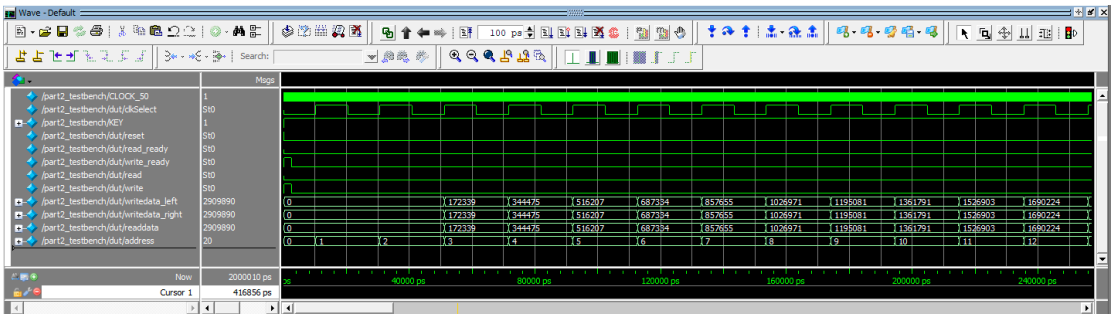

Figure 9: Simulation results of the design "DE1_SoC" in Task 2.

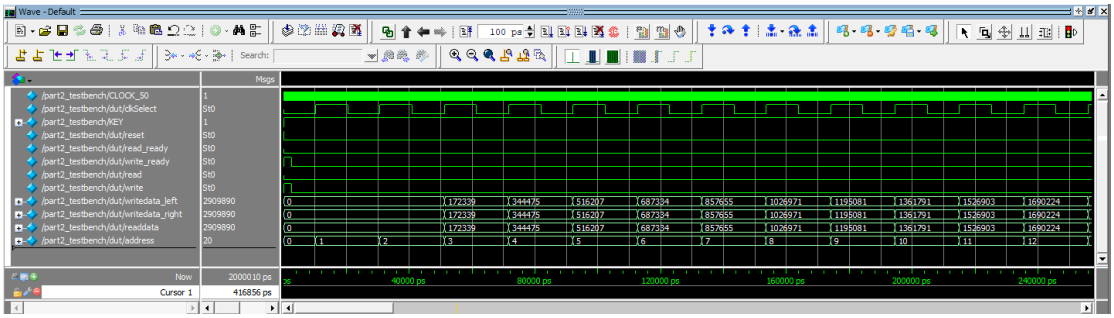

Figure 10: Simulation results of the design "part2" in Task 2.

Task 3: FIR Filter

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

I wrote the testbench to test the module in Task 3. Figures 11 and 12 are the simulation results of "DE1_SoC.sv" and "fir_filter.sv", respectively. The simulation results show that this design performs well. As we can see in Figure 12, the FIR filter does perform the operation of calculating the moving average of last eight data. As indicated in this figure,

$$26 = 2 + 3 + 3 + 3 + 3 + 4 + 4 + 4$$

and

$$68 = 8 + 8 + 8 + 8 + 9 + 9 + 9 + 9.$$

During the verification on LabsLand, this design performs well.

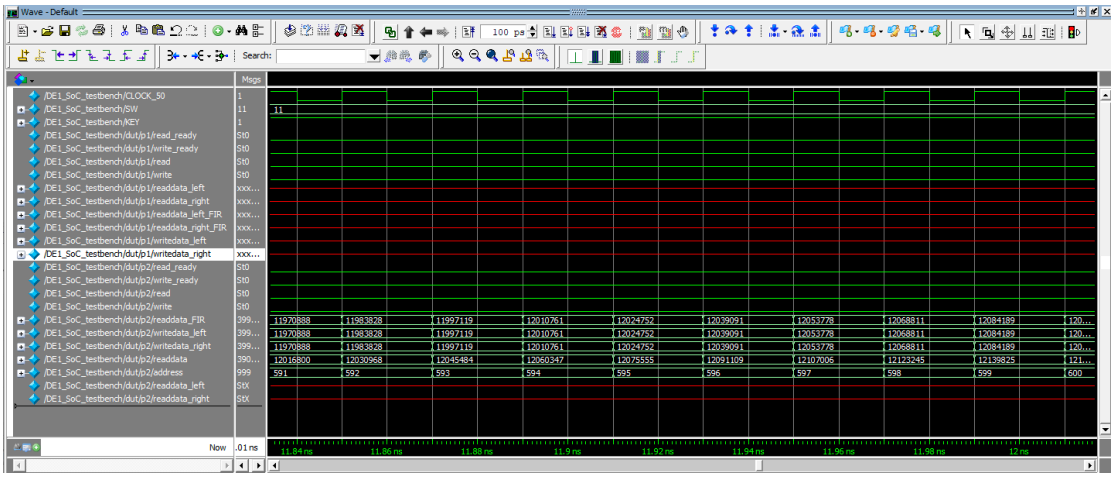

Figure 11: Simulation results of "DE1_SoC" in Task 3.

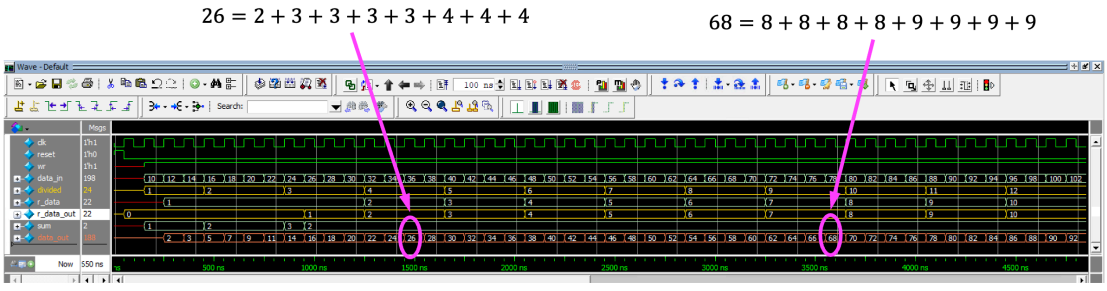

Figure 12: Simulation results of "fir_filter" in Task 3.

III. Final Product

I finished the design specified in Task 1, Task 2, and Task 3. I organized each design as shown in the figures as mentioned before, simulated on the Modelsim with testbenches, and verified the functions with DE1-SoC on the LabsLand. In this Lab, I finished design of the FIR filter circuit which I spent much time.

IV. Appendix

1. Video for Task 2, https://youtu.be/EI7YDYezfHI
2. Video for Task 3, https://youtu.be/fgyuJfonKOU
3. SystemVerilog codes of Task 1 and Task 2, please see the following pages.
4. SystemVerilog codes of Task 3, please see the following pages.

```
1   /*=======================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-28-2025
4   //  EE/CSE371 LAB5--- Digital Signal Processing  (Task 2)
5   //  Device under Test (dut) --- DE1_SoC
6   //  File Name: DE1_SoC.sv
7   /*=======================================================================*/
8   module DE1_SoC (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK,
9                   FPGA_I2C_SDAT, AUD_XCK,
10                  AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK,
11                  AUD_ADCDAT, AUD_DACDAT,
12                  SW);  //<-- Add SW[9] as an input port.
13
14
15      input CLOCK_50, CLOCK2_50;
16      input [0:0] KEY;
17      // I2C Audio/Video config interface
18      output FPGA_I2C_SCLK;
19      inout FPGA_I2C_SDAT;
20      // Audio CODEC
21      output AUD_XCK;
22      input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
23      input AUD_ADCDAT;
24      output AUD_DACDAT;
25      input [9:9] SW;  //<-- Add SW[9] as an input port.
26
27      // Delare the output ports of part1 with new names.
28      logic FPGA_I2C_SCLK1, FPGA_I2C_SDAT1, AUD_XCK1, AUD_DACDAT1;
29
30      // Delare the output ports of part2 with new names.
31      logic FPGA_I2C_SCLK2, FPGA_I2C_SDAT2, AUD_XCK2, AUD_DACDAT2;
32
33      // Use SW[9] to select the output signals to DE1_SoC.
34      assign   FPGA_I2C_SCLK = SW[9] ? FPGA_I2C_SCLK2 : FPGA_I2C_SCLK1;
35      assign   AUD_XCK       = SW[9] ? AUD_XCK2 : AUD_XCK1;
36      assign   AUD_DACDAT    = SW[9] ? AUD_DACDAT2  : AUD_DACDAT1 ;
37
38      // Instantiate part1
39      part1 p1 (.CLOCK_50,
40                .CLOCK2_50,
41                .KEY,
42
43                 // Rename FPGA_I2C_SCLK as FPGA_I2C_SCLK1
44                .FPGA_I2C_SCLK(FPGA_I2C_SCLK1), //<---This line.
45                .FPGA_I2C_SDAT,
46
47                 // Rename AUD_XCK as AUD_XCK1.
48                .AUD_XCK(AUD_XCK1), //<---This line.
49                .AUD_DACLRCK,
50                .AUD_ADCLRCK,
51                .AUD_BCLK,
52                .AUD_ADCDAT,
53
54                 // Rename AUD_DACDAT as AUD_DACDAT1.
55                .AUD_DACDAT(AUD_DACDAT1) //<---This line.
56               );
57
58       // Instantiate part2
59       part2 p2 (.CLOCK_50,
60                 .CLOCK2_50,
61                 .KEY,
62
63                  // Rename FPGA_I2C_SCLK as FPGA_I2C_SCLK2.
64                 .FPGA_I2C_SCLK(FPGA_I2C_SCLK2),  //<---This line.
65                 .FPGA_I2C_SDAT,
66
67                  // Rename AUD_XCK as AUD_XCK2.
68                 .AUD_XCK(AUD_XCK2), //<---This line.
69                 .AUD_DACLRCK,
70                 .AUD_ADCLRCK,
71                 .AUD_BCLK,
72                 .AUD_ADCDAT,
73
74                  // Rename AUD_DACDAT as AUD_DACDAT2.
75                 .AUD_DACDAT(AUD_DACDAT2)  //<---This line.
76                );
77
78  endmodule
```

```
79  /*=============================================================================*/
80  //  Testbench for DE1_SoC
81  /*=============================================================================*/
82  `timescale 1 ps / 1 ps
83  module DE1_SoC_testbench();
84
85      logic CLOCK_50, CLOCK2_50, FPGA_I2C_SCLK,
86            AUD_XCK, AUD_DACLRCK,
87            AUD_ADCLRCK, AUD_BCLK,
88            AUD_ADCDAT, AUD_DACDAT;
89      wire FPGA_I2C_SDAT;
90      logic [9:9] SW;
91      logic [0:0] KEY;
92
93      // Instantiate DE1_SoC module
94      DE1_SoC dut (.*);
95
96      // Set up a simulated clock: 50 MHz
97      parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
98
99      initial begin
00          CLOCK_50 <= 0;
01          forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
02      end
03
04      // Test the design.
05      initial begin
06          KEY[0] <= 0;
07          repeat(1)
08          @(posedge CLOCK_50);
09
10          KEY[0] <= 1;
11          SW[9] = 1'b0;
12          repeat(500)
13          @(posedge CLOCK_50);
14
15          SW[9] = 1'b1;
16          repeat(500)
17          @(posedge CLOCK_50);
18          $stop;
19      end
20
21  endmodule
```

```
/*=======================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing (Task 1)
//  File Name: part1.v
/*=======================================================================*/
module part1 (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
              AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);

   input CLOCK_50, CLOCK2_50;
   input [0:0] KEY;
   // I2C Audio/Video config interface
   output FPGA_I2C_SCLK;
   inout FPGA_I2C_SDAT;
   // Audio CODEC
   output AUD_XCK;
   input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
   input AUD_ADCDAT;
   output AUD_DACDAT;

   // Local wires.
   wire read_ready, write_ready, read, write;
   wire [23:0] readdata_left, readdata_right;
   wire [23:0] writedata_left, writedata_right;
   wire reset = ~KEY[0];

   /////////////////////////////////
   // Your code goes here
   /////////////////////////////////

   // Send the data from mic to speakers.
   assign writedata_left = readdata_left;
   assign writedata_right = readdata_right;
   assign read = read_ready ? 1 : 0;
   assign write = write_ready ? 1 : 0;

////////////////////////////////////////////////////////////////////////////////
// Audio CODEC interface.
//
// The interface consists of the following wires:
// read_ready, write_ready - CODEC ready for read/write operation
// readdata_left, readdata_right - left and right channel data from the CODEC
// read - send data from the CODEC (both channels)
// writedata_left, writedata_right - left and right channel data to the CODEC
// write - send data to the CODEC (both channels)
// AUD_* - should connect to top-level entity I/O of the same name.
//         These signals go directly to the Audio CODEC
// I2C_* - should connect to top-level entity I/O of the same name.
//         These signals go directly to the Audio/Video Config module
////////////////////////////////////////////////////////////////////////////////
   clock_generator my_clock_gen(
      // inputs
      CLOCK2_50,
      reset,

      // outputs
      AUD_XCK
   );

   audio_and_video_config cfg(
      // Inputs
      CLOCK_50,
      reset,

      // Bidirectionals
      FPGA_I2C_SDAT,
      FPGA_I2C_SCLK
   );

   audio_codec codec(
      // Inputs
      CLOCK_50,
      reset,

      read, write,
      writedata_left, writedata_right,

      AUD_ADCDAT,
```

```
79
80          // Bidirectionals
81          AUD_BCLK,
82          AUD_ADCLRCK,
83          AUD_DACLRCK,
84
85          // Outputs
86          read_ready, write_ready,
87          readdata_left, readdata_right,
88          AUD_DACDAT
89      );
90
91  endmodule
92
93
94
```

```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-28-2025
4   //  EE/CSE371 LAB5--- Digital Signal Processing (Task 2)
5   //  File Name: part2.v
6   /*=================================================================*/
7   module part2 (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
8                 AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);
9
10     input CLOCK_50, CLOCK2_50;
11     input [0:0] KEY;
12     // I2C Audio/Video config interface
13     output FPGA_I2C_SCLK;
14     inout FPGA_I2C_SDAT;
15     // Audio CODEC
16     output AUD_XCK;
17     input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
18     input AUD_ADCDAT;
19     output AUD_DACDAT;
20
21     // Local wires.
22     wire read_ready, write_ready, read, write;
23     wire [23:0] readdata_left, readdata_right;
24     wire [23:0] writedata_left, writedata_right;
25     wire reset = ~KEY[0];
26
27     /////////////////////////////////
28     // Your code goes here
29     /////////////////////////////////
30
31     // Generate clk off of CLOCK_50, whichClock picks rate.
32        wire [31:0] div_clk;
33        parameter whichClock = 9; // 48.8 kHz clock  (50 MHz / 2^10 = 48.8 kHz)
34     /*=================================================================*/
35     //     clock_divider (File Name: clock_divider.sv)
36     /*=================================================================*/
37        clock_divider cdiv (.clock(CLOCK_50), .reset(reset), .divided_clocks(div_clk));
38     /*-----------------------------------------------------------------*/
39     //     Clock selection;
40     //     allows for easy switching between simulation and board clocks
41     /*-----------------------------------------------------------------*/
42        wire clkSelect;
43     //Uncomment ONE of the following two lines depending on intention
44     //*****************************************************************
45       // assign clkSelect = CLOCK_50;            // for simulation
46       assign clkSelect = div_clk[whichClock];  // for DE1_SoC  board
47     //*****************************************************************
48     wire [23:0] readdata;
49     wire [15:0] address;
50
51     // Instantiate rom_lab5.
52     rom_lab5 m1     (.address(address), .clock(clkSelect), .q(readdata));
53
54     // Instantiate counter to generate the addresses for memory read operations.
55     counter  m2     (.clock(clkSelect), .reset(reset), .count(address));
56
57     // Send the data read from rom_lab5 to speakers.
58     assign writedata_left = readdata;
59     assign writedata_right = readdata;
60     assign read = read_ready ? 1 : 0;
61     assign write = write_ready ? 1 : 0;
62
63  ////////////////////////////////////////////////////////////////////////////
64  // Audio CODEC interface.
65  //
66  // The interface consists of the following wires:
67  // read_ready, write_ready - CODEC ready for read/write operation
68  // readdata_left, readdata_right - left and right channel data from the CODEC
69  // read - send data from the CODEC (both channels)
70  // writedata_left, writedata_right - left and right channel data to the CODEC
71  // write - send data to the CODEC (both channels)
72  // AUD_* - should connect to top-level entity I/O of the same name.
73  //         These signals go directly to the Audio CODEC
74  // I2C_* - should connect to top-level entity I/O of the same name.
75  //         These signals go directly to the Audio/Video Config module
76  ////////////////////////////////////////////////////////////////////////////
77     clock_generator my_clock_gen(
78        // inputs
```

```
79          CLOCK2_50,
80          reset,
81
82          // outputs
83          AUD_XCK
84      );
85
86      audio_and_video_config cfg(
87          // Inputs
88          CLOCK_50,
89          reset,
90
91          // Bidirectionals
92          FPGA_I2C_SDAT,
93          FPGA_I2C_SCLK
94      );
95
96      audio_codec codec(
97          // Inputs
98          CLOCK_50,
99          reset,
00
01          read, write,
02          writedata_left, writedata_right,
03
04          AUD_ADCDAT,
05
06          // Bidirectionals
07          AUD_BCLK,
08          AUD_ADCLRCK,
09          AUD_DACLRCK,
10
11          // Outputs
12          read_ready, write_ready,
13          readdata_left, readdata_right,
14          AUD_DACDAT
15      );
16
17  endmodule
18
19  /*=============================================================================*/
20  //  Testbench-----------------
    part2_testbench

21  /*=============================================================================*/
22  `timescale 1 ps / 1 ps
23  module part2_testbench();
24
25      reg CLOCK_50, CLOCK2_50;
26      reg [0:0] KEY;
27      // I2C Audio/Video config interface
28      wire FPGA_I2C_SCLK;
29      wire FPGA_I2C_SDAT;
30      // Audio CODEC
31      wire AUD_XCK;
32      wire AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
33      wire AUD_ADCDAT;
34      wire AUD_DACDAT;
35
36      part2 dut (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
37                 AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);
38
39  // Set up a simulated clock: 50 MHz
40  parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
41
42  initial begin
43      CLOCK_50 <= 0;
44      forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
45  end
46
47  // Test the design.
48  initial begin
49      KEY[0] <= 0;
50      @(posedge CLOCK_50);
51      KEY[0] <= 1;
52
53      repeat(100000)
54      @(posedge CLOCK_50);
```

```
155         $stop; // End the simulation.
156     end
157
158     endmodule
```

LAB5-Task1&2


```
1   /*=======================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-28-2025
4   //  EE/CSE371 LAB5--- Digital Signal Processing (Task 2)
5   //  Device Under Test (dut) -- counter
6   //  File Name: counter.sv
7   /*=======================================================================*/
8   module counter (clock, reset, count);
9   parameter N = 17;
10
11  input logic clock, reset;
12  output logic [N-1:0] count;
13
14  always_ff @(posedge clock or posedge reset) begin
15      if (reset)
16          count <= 0;
17      else
18          count <= count + 1;
19      end
20
21  endmodule
22  /*=======================================================================*/
23  //   Testbench
24  /*=======================================================================*/
25  module counter_testbench();
26
27  parameter N = 17;
28  logic clk, rst;
29  logic [N-1:0] count;
30
31  counter dut (.clock(clk), .reset(rst), .count);
32
33  parameter CLK_Period = 100;
34
35  initial begin
36      clk <= 1'b0;
37      forever #(CLK_Period/2) clk <= ~clk;
38  end
39
40  initial begin
41      rst <= 1;
42
43      repeat(3)
44      @(posedge clk);
45      rst <= 0;
46  end
47
48  initial begin
49      repeat(60000)
50      @(posedge clk);
51
52      $stop;
53  end
54
55  endmodule
```

LAB5-Task1&2


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-28-2025
4   //  EE/CSE371 LAB5--- Digital Signal Processing (Task 2)
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
```

```
1   // megafunction wizard: %ROM: 1-PORT%
2   // GENERATION: STANDARD
3   // VERSION: WM1.0
4   // MODULE: altsyncram
5
6   // ============================================================
7   // File Name: rom_lab5.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 rom_lab5 (
41     address,
42     clock,
43     q);
44
45     input [15:0]  address;
46     input   clock;
47     output   [23:0]  q;
48  `ifndef ALTERA_RESERVED_QIS
49  // synopsys translate_off
50  `endif
51     tri1    clock;
52  `ifndef ALTERA_RESERVED_QIS
53  // synopsys translate_on
54  `endif
55
56     wire [23:0] sub_wire0;
57     wire [23:0] q = sub_wire0[23:0];
58
59     altsyncram  altsyncram_component (
60               .address_a (address),
61               .clock0 (clock),
62               .q_a (sub_wire0),
63               .aclr0 (1'b0),
64               .aclr1 (1'b0),
65               .address_b (1'b1),
66               .addressstall_a (1'b0),
67               .addressstall_b (1'b0),
68               .byteena_a (1'b1),
69               .byteena_b (1'b1),
70               .clock1 (1'b1),
71               .clocken0 (1'b1),
72               .clocken1 (1'b1),
73               .clocken2 (1'b1),
74               .clocken3 (1'b1),
75               .data_a ({24{1'b1}}),
76               .data_b (1'b1),
77               .eccstatus (),
78               .q_b (),
```

```
79              .rden_a (1'b1),
80              .rden_b (1'b1),
81              .wren_a (1'b0),
82              .wren_b (1'b0));
83      defparam
84         altsyncram_component.address_aclr_a = "NONE",
85         altsyncram_component.clock_enable_input_a = "BYPASS",
86         altsyncram_component.clock_enable_output_a = "BYPASS",
87         altsyncram_component.init_file = "note_data.mif",
88         altsyncram_component.intended_device_family = "Cyclone V",
89         altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
90         altsyncram_component.lpm_type = "altsyncram",
91         altsyncram_component.numwords_a = 48000,
92         altsyncram_component.operation_mode = "ROM",
93         altsyncram_component.outdata_aclr_a = "NONE",
94         altsyncram_component.outdata_reg_a = "CLOCK0",
95         altsyncram_component.ram_block_type = "M10K",
96         altsyncram_component.widthad_a = 16,
97         altsyncram_component.width_a = 24,
98         altsyncram_component.width_byteena_a = 1;
99
100
101   endmodule
102
103   // ============================================================
104   // CNX file retrieval info
105   // ============================================================
106   // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
107   // Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
108   // Retrieval info: PRIVATE: AclrByte NUMERIC "0"
109   // Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
110   // Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
111   // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
112   // Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
113   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
114   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
115   // Retrieval info: PRIVATE: Clken NUMERIC "0"
116   // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
117   // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
118   // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
119   // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
120   // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
121   // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
122   // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
123   // Retrieval info: PRIVATE: MIFfilename STRING "note_data.mif"
124   // Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "48000"
125   // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
126   // Retrieval info: PRIVATE: RegAddr NUMERIC "1"
127   // Retrieval info: PRIVATE: RegOutput NUMERIC "1"
128   // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
129   // Retrieval info: PRIVATE: SingleClock NUMERIC "1"
130   // Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
131   // Retrieval info: PRIVATE: WidthAddr NUMERIC "16"
132   // Retrieval info: PRIVATE: WidthData NUMERIC "24"
133   // Retrieval info: PRIVATE: rden NUMERIC "0"
134   // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
135   // Retrieval info: CONSTANT: ADDRESS_ACLR_A STRING "NONE"
136   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
137   // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
138   // Retrieval info: CONSTANT: INIT_FILE STRING "note_data.mif"
139   // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
140   // Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
141   // Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
142   // Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "48000"
143   // Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
144   // Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
145   // Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
146   // Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
147   // Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "16"
148   // Retrieval info: CONSTANT: WIDTH_A NUMERIC "24"
149   // Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
150   // Retrieval info: USED_PORT: address 0 0 16 0 INPUT NODEFVAL "address[15..0]"
151   // Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
152   // Retrieval info: USED_PORT: q 0 0 24 0 OUTPUT NODEFVAL "q[23..0]"
153   // Retrieval info: CONNECT: @address_a 0 0 16 0 address 0 0 16 0
154   // Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
155   // Retrieval info: CONNECT: q 0 0 24 0 @q_a 0 0 24 0
156   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5.v TRUE
```

```
157   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5.inc FALSE
158   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5.cmp FALSE
159   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5.bsf FALSE
160   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5_inst.v FALSE
161   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5_bb.v TRUE
162   // Retrieval info: LIB_FILE: altera_mf
163
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing  (Task 3)
//  Device under Test (dut) --- DE1_SoC
//  File Name: DE1_SoC.sv
/*=====================================================================*/
module DE1_SoC (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK,
                FPGA_I2C_SDAT, AUD_XCK,
                AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK,
                AUD_ADCDAT, AUD_DACDAT,
                SW);  //<-- Add SW[9:8] as an input port.

    parameter DATA_WIDTH=24, ADDR_WIDTH=3;

    input CLOCK_50, CLOCK2_50;
    input [0:0] KEY;
    // I2C Audio/Video config interface
    output FPGA_I2C_SCLK;
    inout FPGA_I2C_SDAT;

    // Audio CODEC
    output AUD_XCK;
    input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
    input AUD_ADCDAT;
    output AUD_DACDAT;
    input [9:8] SW;  //<-- Add SW[8:9] as input ports.

    // Delare the output ports of part1 with new names.
    logic FPGA_I2C_SCLK1, AUD_XCK1, AUD_DACDAT1, AUD_DACDAT1_FIR;

    // Delare the output ports of part2 with new names.
    logic FPGA_I2C_SCLK2, AUD_XCK2, AUD_DACDAT2, AUD_DACDAT2_FIR;

    // Use SW[9] to select the output signals to DE1_SoC.
    assign   FPGA_I2C_SCLK = SW[9] ? FPGA_I2C_SCLK2 : FPGA_I2C_SCLK1;
    assign   AUD_XCK       = SW[9] ? AUD_XCK2 : AUD_XCK1;
    assign   AUD_DACDAT    = SW[9] ? AUD_DACDAT2 : AUD_DACDAT1;



    // Instantiate part1
    part1 #(DATA_WIDTH, ADDR_WIDTH) p1 (.CLOCK_50,
              .CLOCK2_50,
              .KEY,

               // Rename FPGA_I2C_SCLK as FPGA_I2C_SCLK1
              .FPGA_I2C_SCLK(FPGA_I2C_SCLK1), //<---This line.
              .FPGA_I2C_SDAT,

               // Rename AUD_XCK as AUD_XCK1.
              .AUD_XCK(AUD_XCK1), //<---This line.
              .AUD_DACLRCK,
              .AUD_ADCLRCK,
              .AUD_BCLK,
              .AUD_ADCDAT,

               // Rename AUD_DACDAT as AUD_DACDAT1.
              .AUD_DACDAT(AUD_DACDAT1), //<---This line.
              .SW(SW[8]) // <--Add SW[8] to select filtered or unfiltered.
              );


     // Instantiate part2
     part2 #(DATA_WIDTH, ADDR_WIDTH) p2 (.CLOCK_50,
              .CLOCK2_50,
              .KEY,

               // Rename FPGA_I2C_SCLK as FPGA_I2C_SCLK2.
              .FPGA_I2C_SCLK(FPGA_I2C_SCLK2),  //<---This line.
              .FPGA_I2C_SDAT,

               // Rename AUD_XCK as AUD_XCK2.
              .AUD_XCK(AUD_XCK2), //<---This line.
              .AUD_DACLRCK,
              .AUD_ADCLRCK,
              .AUD_BCLK,
              .AUD_ADCDAT,
```

```
                 // Rename AUD_DACDAT as AUD_DACDAT2.
                .AUD_DACDAT(AUD_DACDAT2),  //<---This line.
                .SW(SW[8]) // <--Add SW[8] to select filtered or unfiltered.
                );

endmodule
/*=============================================================================*/
//  Testbench for DE1_SoC
/*=============================================================================*/
`timescale 1 ps / 1 ps
module DE1_SoC_testbench();

    logic CLOCK_50, CLOCK2_50, FPGA_I2C_SCLK,
          AUD_XCK, AUD_DACLRCK,
          AUD_ADCLRCK, AUD_BCLK,
          AUD_ADCDAT, AUD_DACDAT;
    wire FPGA_I2C_SDAT;
    logic [9:8] SW;
    logic [0:0] KEY;

    // Instantiate DE1_SoC module
    DE1_SoC dut (.*);

    // Set up a simulated clock: 50 MHz
    parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns

    initial begin
        CLOCK_50 <= 0;
        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
    end

    // Test the design.
    initial begin
        KEY[0] <= 0;
        repeat(1)
        @(posedge CLOCK_50);

        KEY[0] <= 1;
        SW[9:8] = 2'b10;
        repeat(500)
        @(posedge CLOCK_50);

        SW[9:8] = 2'b11;
        repeat(500)
        @(posedge CLOCK_50);
        $stop;
    end

endmodule
```

```
/*=================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing (Task 3)
//  File Name: part1.v
/*=================================================================*/
module part1 #(parameter DATA_WIDTH=24, ADDR_WIDTH=3)
              (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
                AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT,
                SW); //<--Add SW[8] to select filtered or unfiltered.

   input CLOCK_50, CLOCK2_50;
   input [0:0] KEY;
   input [8:8] SW; //<--Add SW[8] to select filtered or unfiltered.
   // I2C Audio/Video config interface
   output FPGA_I2C_SCLK;
   inout FPGA_I2C_SDAT;
   // Audio CODEC
   output AUD_XCK;
   input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
   input AUD_ADCDAT;
   output AUD_DACDAT;

   // Local wires.
   wire read_ready, write_ready, read, write;
   wire [23:0] readdata_left, readdata_right;
   wire [23:0] readdata_left_FIR, readdata_right_FIR;
   wire [23:0] writedata_left, writedata_right;
   wire reset = ~KEY[0];

   ///////////////////////////////////
   // Your code goes here
   ///////////////////////////////////

   // Instantiate FIR Filter for part1 (left-hand source).
   fir_filter #(DATA_WIDTH, ADDR_WIDTH) p3
             (.clk(CLOCK_50),
              .reset(~KEY[0]),
              .wr(1'b1),
              .data_in(readdata_left),
              .data_out(readdata_left_FIR)
              );
   // Instantiate FIR Filter for part1 (right-hand source).
   fir_filter #(DATA_WIDTH, ADDR_WIDTH) p4
             (.clk(CLOCK_50),
              .reset(~KEY[0]),
              .wr(1'b1),
              .data_in(readdata_right),
              .data_out(readdata_right_FIR)
              );

   // Send the data from mic to speakers.
   // Use SW[8] to select filtered or unfiltered.
   assign writedata_left  = SW[8] ? readdata_left_FIR  : readdata_left;
   assign writedata_right = SW[8] ? readdata_right_FIR : readdata_right;
   assign read = read_ready ? 1 : 0;
   assign write = write_ready ? 1 : 0;

////////////////////////////////////////////////////////////////////////////////
// Audio CODEC interface.
//
// The interface consists of the following wires:
// read_ready, write_ready - CODEC ready for read/write operation
// readdata_left, readdata_right - left and right channel data from the CODEC
// read - send data from the CODEC (both channels)
// writedata_left, writedata_right - left and right channel data to the CODEC
// write - send data to the CODEC (both channels)
// AUD_* - should connect to top-level entity I/O of the same name.
//         These signals go directly to the Audio CODEC
// I2C_* - should connect to top-level entity I/O of the same name.
//         These signals go directly to the Audio/Video Config module
////////////////////////////////////////////////////////////////////////////////
   clock_generator my_clock_gen(
      // inputs
      CLOCK2_50,
      reset,

      // outputs
```

```
        AUD_XCK
    );

    audio_and_video_config cfg(
        // Inputs
        CLOCK_50,
        reset,

        // Bidirectionals
        FPGA_I2C_SDAT,
        FPGA_I2C_SCLK
    );

    audio_codec codec(
        // Inputs
        CLOCK_50,
        reset,

        read, write,
        writedata_left, writedata_right,

        AUD_ADCDAT,

        // Bidirectionals
        AUD_BCLK,
        AUD_ADCLRCK,
        AUD_DACLRCK,

        // Outputs
        read_ready, write_ready,
        readdata_left, readdata_right,
        AUD_DACDAT
    );

endmodule
```

```
/*=======================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing (Task 3)
//  File Name: part2.v
/*=======================================================================*/
module part2 #(parameter DATA_WIDTH=24, ADDR_WIDTH=3)
              (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
               AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT,
               SW); //<--Add SW[8] to select filtered or unfiltered.

   input CLOCK_50, CLOCK2_50;
   input [0:0] KEY;
   input [8:8] SW; //<--Add SW[8] to select filtered or unfiltered.
   // I2C Audio/Video config interface
   output FPGA_I2C_SCLK;
   inout FPGA_I2C_SDAT;
   // Audio CODEC
   output AUD_XCK;
   input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
   input AUD_ADCDAT;
   output AUD_DACDAT;

   // Local wires.
   wire read_ready, write_ready, read, write;
   wire [23:0] readdata_FIR;
   wire [23:0] writedata_left, writedata_right;
   wire reset = ~KEY[0];

   ////////////////////////////////
   // Your code goes here
   ////////////////////////////////

   // Generate clk off of CLOCK_50, whichClock picks rate.
      wire [31:0] div_clk;
      parameter whichClock = 9; // 48.8 kHz clock  (50 MHz / 2^10 = 48.8 kHz)
   /*=======================================================================*/
   //     clock_divider (File Name: clock_divider.sv)
   /*=======================================================================*/
      clock_divider cdiv (.clock(CLOCK_50), .reset(reset), .divided_clocks(div_clk));
   /*-----------------------------------------------------------------------*/
   //     Clock selection;
   //     allows for easy switching between simulation and board clocks
   /*-----------------------------------------------------------------------*/
      wire clkSelect;
   //Uncomment ONE of the following two lines depending on intention
   //*****************************************************************
     // assign clkSelect = CLOCK_50;           // for simulation
    assign clkSelect = div_clk[whichClock];  // for DE1_SoC  board
   //*****************************************************************
   wire [23:0] readdata;
   wire [15:0] address;

   // Instantiate rom_lab5.
   rom_lab5 m1     (.address(address), .clock(clkSelect), .q(readdata));

   // Instantiate counter to generate the addresses for memory read operations.
   counter  m2     (.clock(clkSelect), .reset(reset), .count(address));

   // Instantiate FIR Filter for part2.
   fir_filter #(DATA_WIDTH, ADDR_WIDTH) p3
             (.clk(clkSelect),
              .reset(~KEY[0]),
              .wr(1'b1),
              .data_in(readdata),
              .data_out(readdata_FIR)
              );

   // Send the data read from rom_lab5 to speakers.
   // Use SW[8] to select filtered or unfiltered.
   assign writedata_left  = SW[8] ? readdata_FIR  : readdata;
   assign writedata_right = SW[8] ? readdata_FIR  : readdata;
   assign read = read_ready ? 1 : 0;
   assign write = write_ready ? 1 : 0;


////////////////////////////////////////////////////////////////////////////
// Audio CODEC interface.
```

```
//
// The interface consists of the following wires:
// read_ready, write_ready - CODEC ready for read/write operation
// readdata_left, readdata_right - left and right channel data from the CODEC
// read - send data from the CODEC (both channels)
// writedata_left, writedata_right - left and right channel data to the CODEC
// write - send data to the CODEC (both channels)
// AUD_* - should connect to top-level entity I/O of the same name.
//         These signals go directly to the Audio CODEC
// I2C_* - should connect to top-level entity I/O of the same name.
//         These signals go directly to the Audio/Video Config module
////////////////////////////////////////////////////////////////////////////////
   clock_generator my_clock_gen(
      // inputs
      CLOCK2_50,
      reset,

      // outputs
      AUD_XCK
   );

   audio_and_video_config cfg(
      // Inputs
      CLOCK_50,
      reset,

      // Bidirectionals
      FPGA_I2C_SDAT,
      FPGA_I2C_SCLK
   );

   audio_codec codec(
      // Inputs
      CLOCK_50,
      reset,

      read, write,
      writedata_left, writedata_right,

      AUD_ADCDAT,

      // Bidirectionals
      AUD_BCLK,
      AUD_ADCLRCK,
      AUD_DACLRCK,

      // Outputs
      read_ready, write_ready,
      readdata_left, readdata_right,
      AUD_DACDAT
   );

endmodule

/*=============================================================================*/
//  Testbench------------------
part2_testbench

/*=============================================================================*/
`timescale 1 ps / 1 ps
module part2_testbench();

   reg CLOCK_50, CLOCK2_50;
   reg [0:0] KEY;
   // I2C Audio/Video config interface
   wire FPGA_I2C_SCLK;
   wire FPGA_I2C_SDAT;
   // Audio CODEC
   wire AUD_XCK;
   wire AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
   wire AUD_ADCDAT;
   wire AUD_DACDAT;

   part2 dut (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
              AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);
```

```
55
56    // Set up a simulated clock: 50 MHz
57    parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
58
59    initial begin
60        CLOCK_50 <= 0;
61        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
62    end
63
64    // Test the design.
65    initial begin
66       KEY[0] <= 0;
67       @(posedge CLOCK_50);
68       KEY[0] <= 1;
69
70        repeat(100000)
71       @(posedge CLOCK_50);
72        $stop; // End the simulation.
73    end
74
75    endmodule
76
```

```
// megafunction wizard: %ROM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: rom_lab5.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 rom_lab5 (
   address,
   clock,
   q);

   input [15:0]  address;
   input   clock;
   output   [23:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
   tri1    clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

   wire [23:0] sub_wire0;
   wire [23:0] q = sub_wire0[23:0];

   altsyncram  altsyncram_component (
            .address_a (address),
            .clock0 (clock),
            .q_a (sub_wire0),
            .aclr0 (1'b0),
            .aclr1 (1'b0),
            .address_b (1'b1),
            .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_a ({24{1'b1}}),
            .data_b (1'b1),
            .eccstatus (),
            .q_b (),
```

```
9               .rden_a (1'b1),
0               .rden_b (1'b1),
1               .wren_a (1'b0),
2               .wren_b (1'b0));
3       defparam
4          altsyncram_component.address_aclr_a = "NONE",
5          altsyncram_component.clock_enable_input_a = "BYPASS",
6          altsyncram_component.clock_enable_output_a = "BYPASS",
7          altsyncram_component.init_file = "note_data.mif",
8          altsyncram_component.intended_device_family = "Cyclone V",
9          altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
0          altsyncram_component.lpm_type = "altsyncram",
1          altsyncram_component.numwords_a = 48000,
2          altsyncram_component.operation_mode = "ROM",
3          altsyncram_component.outdata_aclr_a = "NONE",
4          altsyncram_component.outdata_reg_a = "CLOCK0",
5          altsyncram_component.ram_block_type = "M10K",
6          altsyncram_component.widthad_a = 16,
7          altsyncram_component.width_a = 24,
8          altsyncram_component.width_byteena_a = 1;
9
0
1   endmodule
2
3   // ============================================================
4   // CNX file retrieval info
5   // ============================================================
6   // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
7   // Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
8   // Retrieval info: PRIVATE: AclrByte NUMERIC "0"
9   // Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
0   // Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
1   // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
2   // Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
3   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
4   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
5   // Retrieval info: PRIVATE: Clken NUMERIC "0"
6   // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
7   // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
8   // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
9   // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
0   // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
1   // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
2   // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
3   // Retrieval info: PRIVATE: MIFfilename STRING "note_data.mif"
4   // Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "48000"
5   // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
6   // Retrieval info: PRIVATE: RegAddr NUMERIC "1"
7   // Retrieval info: PRIVATE: RegOutput NUMERIC "1"
8   // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
9   // Retrieval info: PRIVATE: SingleClock NUMERIC "1"
0   // Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
1   // Retrieval info: PRIVATE: WidthAddr NUMERIC "16"
2   // Retrieval info: PRIVATE: WidthData NUMERIC "24"
3   // Retrieval info: PRIVATE: rden NUMERIC "0"
4   // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
5   // Retrieval info: CONSTANT: ADDRESS_ACLR_A STRING "NONE"
6   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
7   // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
8   // Retrieval info: CONSTANT: INIT_FILE STRING "note_data.mif"
9   // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
0   // Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
1   // Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
2   // Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "48000"
3   // Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
4   // Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
5   // Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
6   // Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
7   // Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "16"
8   // Retrieval info: CONSTANT: WIDTH_A NUMERIC "24"
9   // Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
0   // Retrieval info: USED_PORT: address 0 0 16 0 INPUT NODEFVAL "address[15..0]"
1   // Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
2   // Retrieval info: USED_PORT: q 0 0 24 0 OUTPUT NODEFVAL "q[23..0]"
3   // Retrieval info: CONNECT: @address_a 0 0 16 0 address 0 0 16 0
4   // Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
5   // Retrieval info: CONNECT: q 0 0 24 0 @q_a 0 0 24 0
6   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5.v TRUE
```

```
7   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5.inc FALSE
8   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5.cmp FALSE
9   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5.bsf FALSE
0   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5_inst.v FALSE
1   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_lab5_bb.v TRUE
2   // Retrieval info: LIB_FILE: altera_mf
3
```

LAB5-Task3


```
/*=============================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing (Task 2)
//  Device Under Test (dut) -- counter
//  File Name: counter.sv
/*=============================================================================*/
module counter (clock, reset, count);
parameter N = 17;

input logic clock, reset;
output logic [N-1:0] count;

always_ff @(posedge clock or posedge reset) begin
    if (reset)
        count <= 0;
    else
        count <= count + 1;
    end

endmodule
/*=============================================================================*/
//   Testbench
/*=============================================================================*/
module counter_testbench();

parameter N = 17;
logic clk, rst;
logic [N-1:0] count;

counter dut (.clock(clk), .reset(rst), .count);

parameter CLK_Period = 100;

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

initial begin
    rst <= 1;

    repeat(3)
    @(posedge clk);
    rst <= 0;
end

initial begin
    repeat(60000)
    @(posedge clk);

    $stop;
end

endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing (Task 2)
//  Device Under Test (dut) -- clock_divider
//  File Name: clock_divider.sv
/*=====================================================================*/
// divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz,
...
module clock_divider (clock, reset, divided_clocks);

input logic reset, clock;
output logic [31:0] divided_clocks = 0;

always_ff @(posedge clock) begin
    divided_clocks <= divided_clocks + 1;
end

endmodule

```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing  (Task 3)
//  Device under Test (dut) --- fir_filter
//  File Name: fir_filter.sv
/*=====================================================================*/
module fir_filter #(parameter DATA_WIDTH=24, ADDR_WIDTH=3)
            (clk, reset, wr, data_in, data_out);

   input  logic clk, reset, wr;
   input  logic  [DATA_WIDTH-1:0] data_in;
   output  logic [DATA_WIDTH-1:0] data_out;

   // signal declarations
   logic rd, empty, full, reset_acc;
   logic [DATA_WIDTH-1:0] r_data, divided, r_data_out, sum, acc_q;

   assign rd = full;  // Do not read FIFO until it becomes full.
   assign r_data_out = rd ? r_data : 24'b0;  // This is for getting correct moving average
   assign divided = {{ADDR_WIDTH{data_in[DATA_WIDTH-1]}}, data_in[DATA_WIDTH-1:ADDR_WIDTH
]};
   //assign divided = {{3{data_in[23]}}, data_in[23:3]};

   // instantiate FIFO
   fifo #(DATA_WIDTH, ADDR_WIDTH) m1
            (.clk, .reset, .rd, .wr, .empty, .full, .w_data(divided), .r_data(r_data));

   assign sum = divided - r_data_out;

   // Accumulator
   assign reset_acc = (empty == 0) ? 0 : 1;  // reset signal for accumulator.
   assign data_out = acc_q + sum;

   always_ff @ (posedge clk or posedge reset_acc)
       if (reset_acc)
           acc_q <= sum;
       else
           acc_q <= acc_q + sum;

endmodule  // fir_filter

/*=====================================================================*/
//  Testbench
/*=====================================================================*/
module fir_filter_testbench();

    parameter DATA_WIDTH = 24, ADDR_WIDTH = 3;

    logic clk, reset, wr;
    logic [DATA_WIDTH-1:0] data_in;
    logic [DATA_WIDTH-1:0] data_out;
    integer i;

    // Instantiate fir_filter.
    fir_filter #(DATA_WIDTH, ADDR_WIDTH) dut (.*);

    // clock generator
    parameter CLK_Period = 100;

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

    // Reset stimulus
    initial begin
        reset = 1;  // Assert reset
        repeat(1)
        @(posedge clk);
        reset = 0;   // De-assert reset
    end

    // Use for loop for test generation.
    initial begin
        repeat(2)
        @(posedge clk);
        wr <= 1'b1;
```

```
8           for (i = 10; i < 200; i = i + 2) begin
9               data_in <= i[23:0];
0               @(posedge clk);
1           end
2           $stop;
3       end
4
5   endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing  (Task 3)
//  Device under Test (dut) --- fifo
//  File Name: fifo.sv
//  This is form Homework 3.
/*=====================================================================*/
/* FIFO buffer FWFT implementation for specified data and address
 * bus widths based on internal register file and FIFO controller.
 * Inputs: 1-bit rd removes head of buffer and 1-bit wr writes
 * w_data to the tail of the buffer.
 * Outputs: 1-bit empty and full indicate the status of the buffer
 * and r_data holds the value of the head of the buffer (unless empty).
 */
module fifo #(parameter DATA_WIDTH=24, ADDR_WIDTH=3)
             (clk, reset, rd, wr, empty, full, w_data, r_data);

   input  logic clk, reset, rd, wr;
   output logic empty, full;
   input  logic [DATA_WIDTH-1:0] w_data;
   output logic [DATA_WIDTH-1:0] r_data;

   // signal declarations
   logic [ADDR_WIDTH-1:0] w_addr, r_addr;
   logic w_en;

   // enable write only when FIFO is not full
   // or if reading and writing simultaneously
   assign w_en = wr & (~full | rd);

   // instantiate FIFO controller and register file
   fifo_ctrl #(ADDR_WIDTH) c_unit (.*);
   reg_file #(DATA_WIDTH, ADDR_WIDTH) r_unit (.*);

endmodule  // fifo

/*=====================================================================*/
//  Testbench
/*=====================================================================*/
module fifo_testbench();

parameter DATA_WIDTH = 24, ADDR_WIDTH = 3;

logic clk, reset, rd, wr, empty, full;
logic [DATA_WIDTH-1:0] w_data;
logic [DATA_WIDTH-1:0]r_data;

integer i;

fifo #(DATA_WIDTH, ADDR_WIDTH) dut (.*);

parameter CLK_Period = 100;

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

initial begin
    reset <= 1;

    repeat(1)
    @(posedge clk);

    reset <= 0;
end

initial begin
        @(negedge clk);

        {rd, wr} <= 2'b01;
        w_data <= 24'h9090ab;
        repeat(2**(ADDR_WIDTH-1))
        @(negedge clk);

        w_data <= 24'h9012cd;
        repeat(2**(ADDR_WIDTH-1))
```

```
            @(negedge clk);
            
              {rd, wr} <= 2'b00;
            repeat(2)
            @(negedge clk);
            
              {rd, wr} <= 2'b10;
            repeat(2**(ADDR_WIDTH))
            @(negedge clk);
            
              {rd, wr} <= 2'b01;
            w_data <= 24'h9034ef;
            repeat(2**(ADDR_WIDTH-1))
            @(negedge clk);
            
              {rd, wr} <= 2'b00;
            repeat(2)
            @(negedge clk);
            
              {rd, wr} <= 2'b10;
            repeat(2**(ADDR_WIDTH-1))
            @(negedge clk);
            
              {rd, wr} <= 2'b01;
            w_data <= 24'h9056ca;
            repeat(2**(ADDR_WIDTH-1))
            @(negedge clk);
            
              {rd, wr} <= 2'b10;
            repeat(2**(ADDR_WIDTH-1))
            @(negedge clk);
            
              {rd, wr} <= 2'b11;
            w_data <= 24'h9078db;
            repeat(2**(ADDR_WIDTH))
            @(negedge clk);
            
              {rd, wr} <= 2'b10;
            repeat(2**(ADDR_WIDTH)+3)
            @(negedge clk);
        $stop;
    end

    endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing  (Task 3)
//  Device under Test (dut) --- fifo_ctrl
//  File Name: fifo_ctrl.sv
//  This is form Homework 3.
/*=====================================================================*/
/* FIFO controller to manage a register file as a circular queue.
 * Manipulates output read and write addresses based on 1-bit
 * read (rd) and write (wr) requests and current buffer status
 * signals empty and full.
 */
module fifo_ctrl #(parameter ADDR_WIDTH=3)
                  (clk, reset, rd, wr, empty, full, w_addr, r_addr);

   input  logic clk, reset, rd, wr;
   output logic empty, full;
   output logic [ADDR_WIDTH-1:0] w_addr, r_addr;

   // signal declarations
   logic [ADDR_WIDTH-1:0] rd_ptr, rd_ptr_next;
   logic [ADDR_WIDTH-1:0] wr_ptr, wr_ptr_next;
   logic empty_next, full_next;

   // output assignments
   assign w_addr = wr_ptr;
   assign r_addr = rd_ptr;

   // fifo controller logic
   always_ff @(posedge clk) begin
      if (reset)
         begin
            wr_ptr <= 0;
            rd_ptr <= 0;
            full   <= 0;
            empty  <= 1;
         end
      else
         begin
            wr_ptr <= wr_ptr_next;
            rd_ptr <= rd_ptr_next;
            full   <= full_next;
            empty  <= empty_next;
         end
   end  // always_ff

   // next state logic
   always_comb begin
      // default to keeping the current values
      rd_ptr_next = rd_ptr;
      wr_ptr_next = wr_ptr;
      empty_next = empty;
      full_next = full;
      case ({rd, wr})
         2'b11:  // read and write
            begin
               rd_ptr_next = rd_ptr + 1'b1;
               wr_ptr_next = wr_ptr + 1'b1;
            end
         2'b10:  // read
            if (~empty)
               begin
                  rd_ptr_next = rd_ptr + 1'b1;
                  if (rd_ptr_next == wr_ptr)
                     empty_next = 1;
                  full_next = 0;
               end
         2'b01:  // write
            if (~full)
               begin
                  wr_ptr_next = wr_ptr + 1'b1;
                  empty_next = 0;
                  if (wr_ptr_next == rd_ptr)
                     full_next = 1;
               end
         2'b00: ; // no change
      endcase
```

```
9       end  // always_comb
0
1    endmodule  // fifo_ctrl
2
```

LAB5-Task3


```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 02-28-2025
//  EE/CSE371 LAB5--- Digital Signal Processing  (Task 3)
//  Device under Test (dut) --- reg_file
//  File Name: reg_file.sv
//  This is form Homework 3.
/*=====================================================================*/
/* Register file module for specified data and address bus widths.
 * Asynchronous read port (r_addr -> r_data) and synchronous write
 * port (w_data -> w_addr if w_en).
 */
module reg_file #(parameter DATA_WIDTH=24, ADDR_WIDTH=3)
                 (clk, w_data, w_en, w_addr, r_addr, r_data);

   input  logic clk, w_en;
   input  logic [ADDR_WIDTH-1:0] w_addr, r_addr;
   input  logic [DATA_WIDTH-1:0] w_data;
   output logic [DATA_WIDTH-1:0] r_data;

   // array declaration (registers)
   logic [DATA_WIDTH-1:0] array_reg [0:2**ADDR_WIDTH-1];

   // write operation (synchronous)
   always_ff @(posedge clk)
      if (w_en)
         array_reg[w_addr] <= w_data;

   // read operation (asynchronous)
   assign r_data = array_reg[r_addr];

endmodule  // reg_file
```