Justin Sim and Mina Gao (Lab Group 4)
EE 371
February 23, 2024
Lab 5 Report

Procedure

This lab consists of three tasks. In the first task, we played a music file through the audio input of the DE1-SoC, which was then simultaneously output through the audio output of the device. In the second task, we generated a static tone from an internal memory. The third task involved creating a Finite Impulse Response (FIR) filter using a FIFO buffer and accumulator to filter noise from the audio in both the first and second tasks.

Task #1

To play and record the external music file 'piano_noisy', we modified the provided starter kit circuit to pass audio input to audio output only when the CODEC was ready. To ensure the successful passing of data, we needed to assert that the circuit was both read_ready and write_ready, and also to write data while simultaneously reading data.

```
24
25      assign writedata_left = readdata_left;
26      assign writedata_right = readdata_right;
27      assign read = read_ready && write_ready;
28      assign write = write_ready && read_ready;
29
```

Figure 1: Screenshot of the starter code set up

Task #2

The second task involved producing a static tone from memory. We first used a Python script to generate a MIF file for the note C4, guided by the tutorial document, which contains a sample size of 48,000, 24 bits long each. We then created a ROM initialized with the values stored in the MIF file. To output the values stored in the ROM sequentially to the Audio CODEC, we used an always_ff block to loop back to the start upon reaching the end. We modified these changes in the provided starter code. Then combined this task with task 1 by instantiating them in a top-level module that played the external audio (Task 1) when SW9 was set to 0 and our tone from memory (Task 2) when SW9 was set to 1.

Task #3

The third task involved learning a basic signal processing technique known as filtering. We implemented a noise-filtering technique using an averaging Finite Impulse Response (FIR)

filter. This filter worked by averaging the values of adjacent samples to remove noise from a sound. We implemented this using a FIFO buffer and accumulator. The task involved experimenting with different values to see the effect on the noisy audio sample.
We first started with dividing the input sample by N and storing the result in the FIFO buffer. When the FIFO buffer is full, the new sample is added to the accumulator, and the oldest sample is subtracted from the FIFO, to maintain the average of the last N samples. Used switch SW8 to toggle between filtered (SW8=1) and unfiltered (SW8=0) audio output. This filter was able to interact with both Task 1 and Task 2.
The difficulty of this task is to account for the right delays, as well as finding a way to divide the incoming signal. We were able to work around the delay issues by providing a delay into the variable preval, however, the testbench was insufficient to show us the capabilities of the current algorithm. For the sake of time, it is on pause.

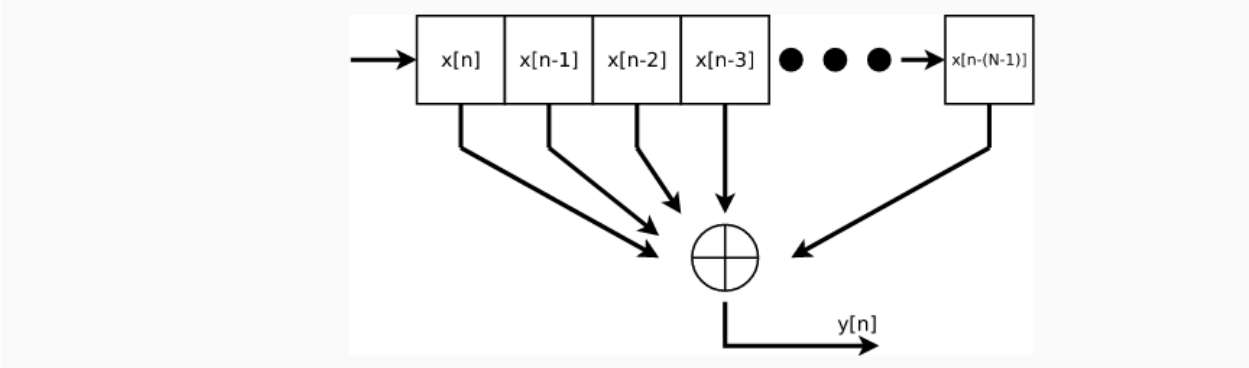

Figure: Task 3 Signal Flow through Boxcar/Moving Average Filter

Results

Task 1: All under top level module

Task 2: All under top level module, except for rom unit

Task 3: testbench under construction

Appendix

Task 1

```
1   /*
2   Mina Gao and Justin Sim
3   2/23/2024
4   EE 371 Hussein
5   Lab 4, Task 1
6
7      This module allows the DE1_SoC audio driver to play an
8      input data file. It monitors read and write control signals
9      to do so.
10
11     Relavant inputs:
12        KEY[0]              Reset
13     outputs:
14        driven by VHDL audio driver files
15
16  */
17  module part1 #(parameter IW=24,N=4,LGMEM=$clog2(N)) (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK
    , FPGA_I2C_SDAT, AUD_XCK,
18                 AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);
19
20     input CLOCK_50, CLOCK2_50;
21     input [0:0] KEY;
22     // I2C Audio/Video config interface
23     output FPGA_I2C_SCLK;
24     inout FPGA_I2C_SDAT;
25     // Audio CODEC
26     output AUD_XCK;
27     input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
28     input AUD_ADCDAT;
29     output AUD_DACDAT;
30
31     // Local wires.
32     wire read_ready, write_ready, read, write;
33     wire [23:0] readdata_left, readdata_right;
34     wire [23:0] writedata_left, writedata_right;
35     wire reset = ~KEY[0];
36
37     /////////////////////////////////
38     // Your code goes here
39     /////////////////////////////////
40
41     assign writedata_left = readdata_left;
42     assign writedata_right = readdata_right;
43     assign read = read_ready && write_ready;
44     assign write = write_ready && read_ready;
45
46  /////////////////////////////////////////////////////////////////////////////////
47  // Audio CODEC interface.
48  //
49  // The interface consists of the following wires:
50  // read_ready, write_ready - CODEC ready for read/write operation
51  // readdata_left, readdata_right - left and right channel data from the CODEC
52  // read - send data from the CODEC (both channels)
53  // writedata_left, writedata_right - left and right channel data to the CODEC
54  // write - send data to the CODEC (both channels)
55  // AUD_* - should connect to top-level entity I/O of the same name.
56  //         These signals go directly to the Audio CODEC
57  // I2C_* - should connect to top-level entity I/O of the same name.
58  //         These signals go directly to the Audio/Video Config module
59  /////////////////////////////////////////////////////////////////////////////////
60     clock_generator my_clock_gen(
61        // inputs
62        CLOCK2_50,
63        reset,
64
65        // outputs
66        AUD_XCK
67     );
68
69     audio_and_video_config cfg(
70        // Inputs
71        CLOCK_50,
72        reset,
```

```
73
74          // Bidirectionals
75          FPGA_I2C_SDAT,
76          FPGA_I2C_SCLK
77      );
78
79      audio_codec codec(
80          // Inputs
81          CLOCK_50,
82          reset,
83
84          read, write,
85          writedata_left, writedata_right,
86
87          AUD_ADCDAT,
88
89          // Bidirectionals
90          AUD_BCLK,
91          AUD_ADCLRCK,
92          AUD_DACLRCK,
93
94          // Outputs
95          read_ready, write_ready,
96          readdata_left, readdata_right,
97          AUD_DACDAT
98      );
99
100 endmodule
101
102
```

Task 2


```
1   /*
2   Mina Gao and Justin Sim
3   2/23/2024
4   EE 371 Hussein
5   Lab 4, Task 2
6
7      This module stores a note C4 into a ROM unit, once Compiled
8      it allows the DE1_SoC to toggle back and forth between the
9      stored note and an uploaded mp3 file.
10
11     Relavant inputs:
12        KEY[0]               Reset
13     outputs:
14        driven by VHDL audio driver files
15
16  */
17  module DE1_SoC (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
18                  AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT, SW);
19
20     input CLOCK_50, CLOCK2_50;
21     input [0:0] KEY;
22     input logic [9:0] SW;
23     // I2C Audio/Video config interface
24     output FPGA_I2C_SCLK;
25     inout FPGA_I2C_SDAT;
26     // Audio CODEC
27     output AUD_XCK;
28     input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
29     input AUD_ADCDAT;
30     output AUD_DACDAT;
31
32     // Local wires.
33     wire read_ready, write_ready, read, write;
34     wire [23:0] readdata_left, readdata_right;
35     wire [23:0] writedata_left, writedata_right;
36     logic reset;
37     logic [15:0] addr;
38     logic [23:0] q;
39
40     /////////////////////////////////
41     // Your code goes here
42     /////////////////////////////////
43     always_ff @(posedge CLOCK_50) begin
44        if (addr == 47999)       addr <= 0;
45        else if (read && write) addr <= addr + 1'b1;
46        else                     addr <= addr;
47     end
48
49     rom_task_2 ROM(
50     .address(addr),
51     .clock(CLOCK_50),
52     .q(q));
53
54     always_comb begin
55        case(SW[9])
56           1'b0: begin
57              writedata_left = readdata_left;
58              writedata_right = readdata_right;
59           end
60           1'b1: begin
61              writedata_left = q;
62              writedata_right = q;
63           end
64        endcase
65     end
66
67     assign reset = ~KEY[0];
68     assign read = read_ready && write_ready;
69     assign write = write_ready && read_ready;
70
71  /////////////////////////////////////////////////////////////////////////////////
72  // Audio CODEC interface.
73  //
```

```
74  // The interface consists of the following wires:
75  // read_ready, write_ready - CODEC ready for read/write operation
76  // readdata_left, readdata_right - left and right channel data from the CODEC
77  // read - send data from the CODEC (both channels)
78  // writedata_left, writedata_right - left and right channel data to the CODEC
79  // write - send data to the CODEC (both channels)
80  // AUD_* - should connect to top-level entity I/O of the same name.
81  //         These signals go directly to the Audio CODEC
82  // I2C_* - should connect to top-level entity I/O of the same name.
83  //         These signals go directly to the Audio/Video Config module
84  ////////////////////////////////////////////////////////////////////////////////
85     clock_generator my_clock_gen(
86        // inputs
87        CLOCK2_50,
88        reset,
89
90        // outputs
91        AUD_XCK
92     );
93
94     audio_and_video_config cfg(
95        // Inputs
96        CLOCK_50,
97        reset,
98
99        // Bidirectionals
100       FPGA_I2C_SDAT,
101       FPGA_I2C_SCLK
102    );
103
104    audio_codec codec(
105       // Inputs
106       CLOCK_50,
107       reset,
108
109       read, write,
110       writedata_left, writedata_right,
111
112       AUD_ADCDAT,
113
114       // Bidirectionals
115       AUD_BCLK,
116       AUD_ADCLRCK,
117       AUD_DACLRCK,
118
119       // Outputs
120       read_ready, write_ready,
121       readdata_left, readdata_right,
122       AUD_DACDAT
123    );
124
125 endmodule
```

```
1   /* Register file module for specified data and address bus widths.
2    * Asynchronous read port (r_addr -> r_data) and synchronous write
3    * port (w_data -> w_addr if w_en).
4    */
5   module reg_file #(parameter DATA_WIDTH=8, ADDR_WIDTH=2)
6                    (clk, w_data, w_en, w_addr, r_addr, r_data);
7
8      input  logic clk, w_en;
9      input  logic [ADDR_WIDTH-1:0] w_addr, r_addr;
10     input  logic [DATA_WIDTH-1:0] w_data;
11     output logic [DATA_WIDTH-1:0] r_data;
12
13     // array declaration (registers)
14     logic [DATA_WIDTH-1:0] array_reg [0:2**ADDR_WIDTH-1];
15
16     // write operation (synchronous)
17     always_ff @(posedge clk)
18        if (w_en)
19           array_reg[w_addr] <= w_data;
20
21     // read operation (asynchronous)
22     assign r_data = array_reg[r_addr];
23
24  endmodule  // reg_file
```

```
1   // megafunction wizard: %ROM: 1-PORT%
2   // GENERATION: STANDARD
3   // VERSION: WM1.0
4   // MODULE: altsyncram
5
6   // ============================================================
7   // File Name: rom_task_2.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_task_2 (
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_task_2.v TRUE
157   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_task_2.inc FALSE
158   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_task_2.cmp FALSE
159   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_task_2.bsf FALSE
160   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_task_2_inst.v FALSE
161   // Retrieval info: GEN_FILE: TYPE_NORMAL rom_task_2_bb.v TRUE
162   // Retrieval info: LIB_FILE: altera_mf
163
```

Task 3

```
1   /*
2   Mina Gao and Justin Sim
3   2/23/2024
4   EE 371 Hussein
5   Lab 4, Task 3
6
7      This module takes the moving average of N samples
8      of an input signal. It does this using a divider
9      to divide the signal, FIFO to delay the samples
10     and an accumulator.
11
12     Relavant inputs:
13        dataIn
14        clk, rst
15        read_data
16     outputs:
17        dataout
18
19  */
20  module moving_avg_filter #(parameter IW=24,N=4,LGMEM=$clog2(N)) (
21     input logic    [IW-1:0]       datain,
22     input logic                   clk,rst,
23     input logic                   read_data,  // T if DataIn exists
24
25     output logic   [IW-1:0]       dataout
26  );
27
28     /*initialize values*/
29     logic [IW-1:0] div_sig_in; // divided signal
30           assign div_sig_in = {{(N-1){datain[24-1]}}, datain[24-1:(N-1)]};
31     logic [IW-1:0] mem[0:LGMEM-1]; // memory register
32     logic [0:LGMEM-1] wraddr, rdaddr;   //write and read address
33     logic [IW-1:0] memout, preval, memval, minus, accum; //preliminary variables described
   below
34     logic full;    // control signal for full FIFO
35
36     //1. Shift the write address to emulate a FIFO Controller
37     initial wraddr = 1'b0;
38     always_ff@(posedge clk) begin
39        if (rst)
40           wraddr <= 1'b0;
41        else if (read_data)
42           wraddr <= wraddr + 1'b1;
43     end   //always_ff
44
45     /*-------------------------------------------------------*/
46     //2. Shift the read address to emulate a FIFO Controller
47     initial rdaddr = 1'b0;
48     always_ff@(posedge clk) begin
49        if (rst)
50           rdaddr <= N-1;
51        else if (read_data)
52           rdaddr <= rdaddr + 1'b1;
53     end   //always_ff
54
55     //3. input values into the memory unit
56     always_ff @(posedge clk)
57        if (read_data)
58           mem[wraddr] <= div_sig_in;
59
60     //4. memout: unit in memory coming out of FIFO (N+1 signal)
61     initial memout = 1'b0;
62     always_ff @(posedge clk)
63        if (read_data)
64           memout <= mem[rdaddr];
65
66     //5. Intermediate step to account for delay of input
67     initial preval = 1'b0;
68     always_ff @(posedge clk)
69        if (rst)
70           preval <= 0;
71        else if (read_data)
72           preval <= div_sig_in;
```

```
73
74      //6. controls for 'full' identification
75      initial full = 1'b0;
76      always_ff @(posedge clk)
77         if (rst)
78            full <= 0;
79         else if (read_data)
80            full <= full || (rdaddr == 0);
81
82      // 7. subtract the sample falling out of register
83      initial minus = 0;
84      always @(posedge clk)
85         if (rst)
86            minus <= 0;
87         else if (read_data) begin
88            if (full)
89               minus <= {preval[IW-1],preval}
90                           - {mem[(IW-1)], memval};
91            else
92               minus <= {preval[(IW-1)], preval};
93         end //else if
94
95      //8. Accumulate!
96      initial accum = 0;
97      always_ff @(posedge clk)
98         if (rst)
99            accum <= 0;
100        else if (read_data)
101           accum <= accum + { {(LGMEM-1){minus[IW-1]}}, minus};
102
103
104     assign dataout = accum;
105  endmodule
106
107   module moving_avg_filter_tb();
108     parameter IW=24, N=4, LGMEM=$clog2(N);
109
110     logic    [IW-1:0]       datain;
111     logic                   clk,rst;
112     logic                   read_data;  // T if DataIn exists
113     logic    [IW-1:0]       dataout;
114
115     moving_avg_filter dut(.*);
116
117     parameter clock_period = 100;
118
119     initial begin clk <= 0;
120        forever #(clock_period /2) clk <= ~clk; end
121
122     initial begin
123     rst <= 1; @(posedge clk);
124     datain <= 100; read_data <= 1; @(posedge clk);
125     @(posedge clk);@(posedge clk);@(posedge clk);
126     @(posedge clk);@(posedge clk);@(posedge clk);
127     @(posedge clk);@(posedge clk);@(posedge clk);
128     @(posedge clk);@(posedge clk);@(posedge clk);
129   //    Datain <= 100; wr <=1; rd<=0; reset <= 0; @(posedge clk);
130   //    Datain <= 120; wr <=1; rd<=0; reset <= 0; @(posedge clk);
131   //    Datain <= 160; wr <=1; rd<=0; reset <= 0; @(posedge clk);
132   //    Datain <= 200; wr <=1; rd<=0; reset <= 0; @(posedge clk);
133   //    Datain <= 40; wr <=1; rd<=0;  reset <= 0; @(posedge clk);
134   //
135   //    wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
136   //    wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
137   //    wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
138   //    wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
139   //    Datain <= 40; wr <=1; rd<=0;  reset <= 0; @(posedge clk);
140   //    Datain <= 80; wr <=1; rd<=0;  reset <= 0; @(posedge clk);
141   //    Datain <= 20; wr <=1; rd<=0;  reset <= 0; @(posedge clk);
142   //    Datain <= 24; wr <=1; rd<=0;  reset <= 0; @(posedge clk);
143   //// wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
144   //    wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
145   //    wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
```

```
146    //   wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
147    //   wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
148    //   wr <= 0; rd<=1;   reset <= 0; @(posedge clk);
149
150    //   rd <=1; wr<=0;                @(posedge clk);@(posedge clk);
151      $stop;
152      end
153    endmodule
```

```
1   /*
2   Mina Gao and Justin Sim
3   2/23/2024
4   EE 371 Hussein
5   Lab 4, Task 3
6
7      This module instantiates the filter such that
8      it filters the input audio signal in the moving
9      average filter
10
11     Relavant inputs:
12        KEY[0]               Reset
13     outputs:
14        driven by VHDL audio driver files
15
16  */
17  module part3 #(parameter IW=24,N=4,LGMEM=$clog2(N)) (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK
    , FPGA_I2C_SDAT, AUD_XCK,
18                 AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);
19
20     input CLOCK_50, CLOCK2_50;
21     input [0:0] KEY;
22     // I2C Audio/Video config interface
23     output FPGA_I2C_SCLK;
24     inout FPGA_I2C_SDAT;
25     // Audio CODEC
26     output AUD_XCK;
27     input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
28     input AUD_ADCDAT;
29     output AUD_DACDAT;
30
31     // Local wires.
32     wire read_ready, write_ready, read, write;
33     wire [23:0] readdata_left, readdata_right;
34     wire [23:0] writedata_left, writedata_right;
35     wire reset = ~KEY[0];
36
37     /////////////////////////////////
38     // Your code goes here
39     /////////////////////////////////
40  //
41  // assign writedata_left = readdata_left;
42  // assign writedata_right = readdata_right;
43     assign read = read_ready && write_ready;
44     assign write = write_ready && read_ready;
45
46     moving_avg_filter filter_L (.datain(read_data_left), .clk(CLOCK_50), .rst(reset),
47                                                 .read_data(read_ready),.dataout(write_data_left
    ));
48
49
50      moving_avg_filter filter_R (.datain(read_data_right), .clk(CLOCK_50), .rst(reset),
51                                                 .read_data(read_ready),.dataout(
    write_data_right));
52
53  ////////////////////////////////////////////////////////////////////////////////
54  // Audio CODEC interface.
55  //
56  // The interface consists of the following wires:
57  // read_ready, write_ready - CODEC ready for read/write operation
58  // readdata_left, readdata_right - left and right channel data from the CODEC
59  // read - send data from the CODEC (both channels)
60  // writedata_left, writedata_right - left and right channel data to the CODEC
61  // write - send data to the CODEC (both channels)
62  // AUD_* - should connect to top-level entity I/O of the same name.
63  //         These signals go directly to the Audio CODEC
64  // I2C_* - should connect to top-level entity I/O of the same name.
65  //         These signals go directly to the Audio/Video Config module
66  ////////////////////////////////////////////////////////////////////////////////
67     clock_generator my_clock_gen(
68        // inputs
69        CLOCK2_50,
70        reset,
```

```
71
72          // outputs
73          AUD_XCK
74      );
75
76      audio_and_video_config cfg(
77          // Inputs
78          CLOCK_50,
79          reset,
80
81          // Bidirectionals
82          FPGA_I2C_SDAT,
83          FPGA_I2C_SCLK
84      );
85
86      audio_codec codec(
87          // Inputs
88          CLOCK_50,
89          reset,
90
91          read, write,
92          writedata_left, writedata_right,
93
94          AUD_ADCDAT,
95
96          // Bidirectionals
97          AUD_BCLK,
98          AUD_ADCLRCK,
99          AUD_DACLRCK,
100
101         // Outputs
102         read_ready, write_ready,
103         readdata_left, readdata_right,
104         AUD_DACDAT
105     );
106
107 endmodule
108
109
```