Khoa Tran and Ravi Sangani
EE 371
May 22, 2021
Lab 5 Report

Procedure

Task #1

In order to approach this task, we looked at the block diagram to understand the connection between the audio CODEC interface as well as the audio/video configuration and the noise generator. Looking at these modules and their connection, we were able to understand when read and write is enabled allowing for writedata_left and writedata_right to receive the read data with the addition of the noise. We were able to understand how KEY0 was implemented to add the noise in the given audio input. However, without clicking any KEY the audio will receive the input from the given piano file.

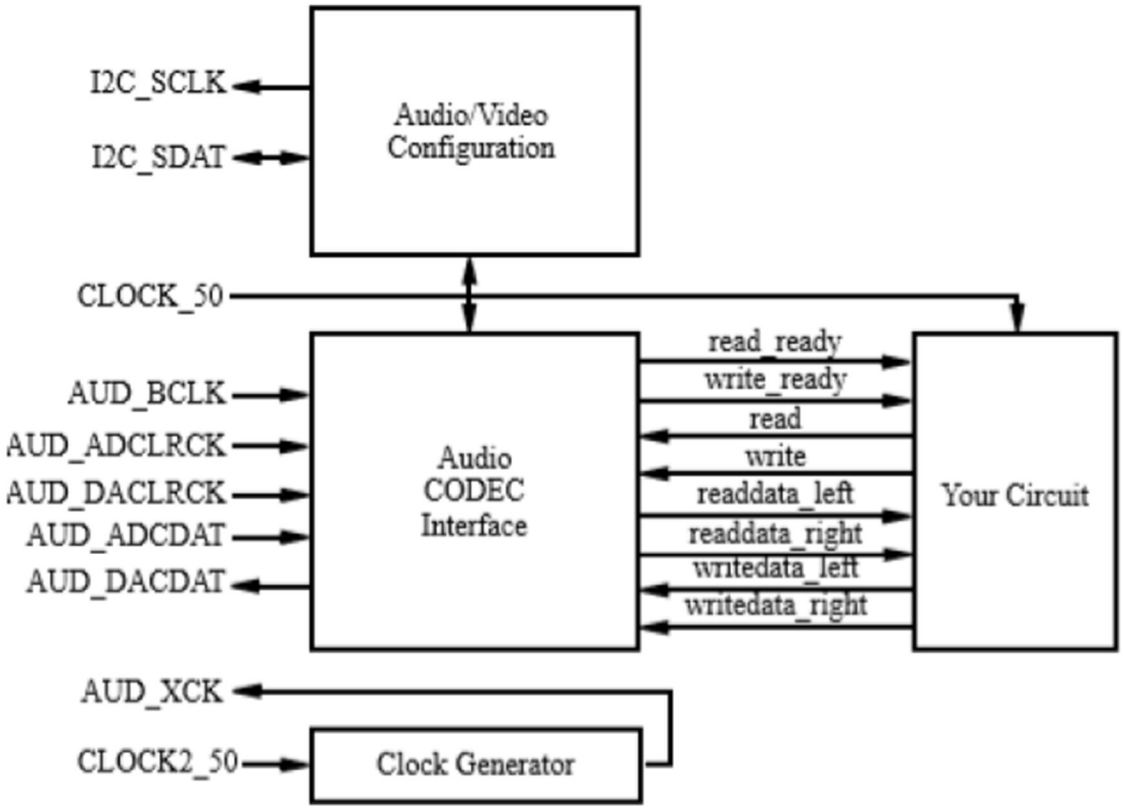

Figure 1: Block diagram for task 1

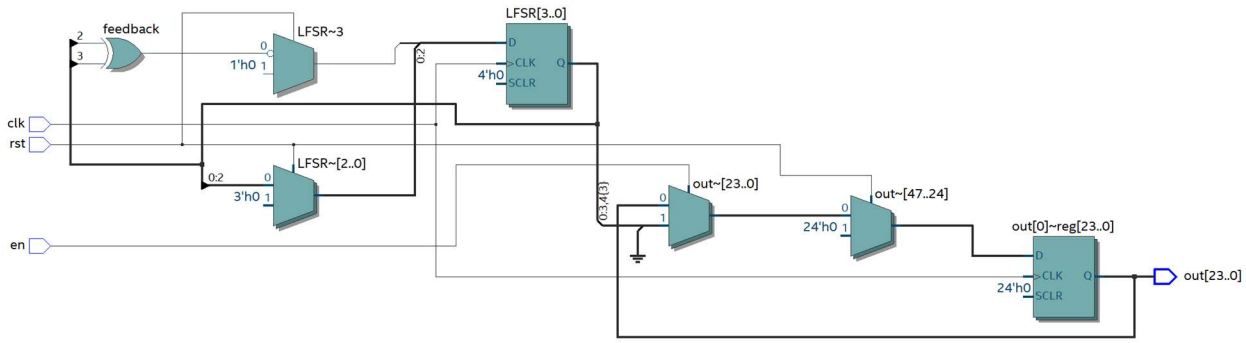

Figure 2: Block Diagram for noise generator

Task #2

Approaching this problem, we developed the block diagram for task 2, which is a fir filter that takes 8 samples and shifts between 7 different registers and from the output of each register, divide the value by 8 and add for the total in order to get the average and eliminate the noise through the average of 8 samples and continues onwards.

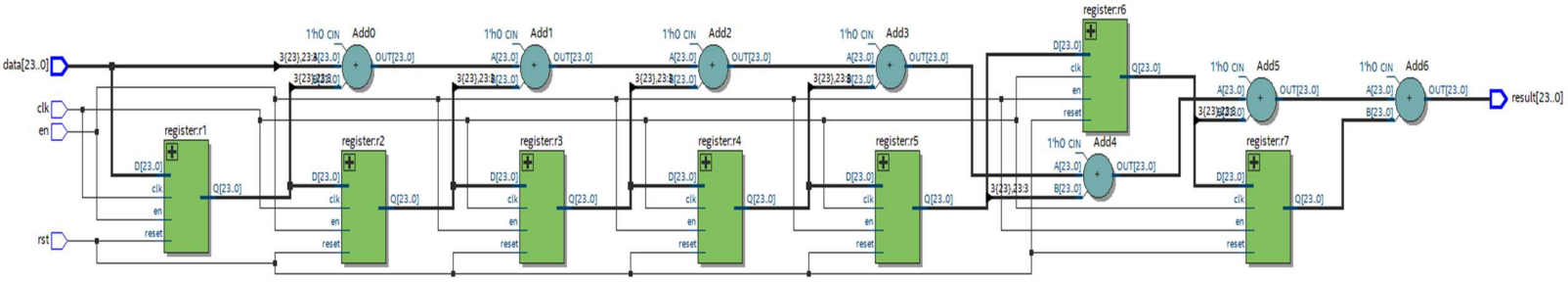

Figure 3: Block diagram for task 2

Task #3

In order to solve this problem, we developed the block diagram, which consist of 16 registers or n shift registers acting like an n size buffer. This was developed to parameterize the FIR filter in task 2, in order to increase the amount of samples to average. This parameterization allows for filtering of the noise to increase and eliminating some of the high pitch sounds that still exist in task 2. In this task, we used a generate statement to have a for loop, calling n number of registers and shifting them, storing the values in a 2D array. This way, we can keep track of the oldest value that was entered and subtract from the final output value. The accumalator also works like a register, adding divided values as well as subtracting the oldest value.

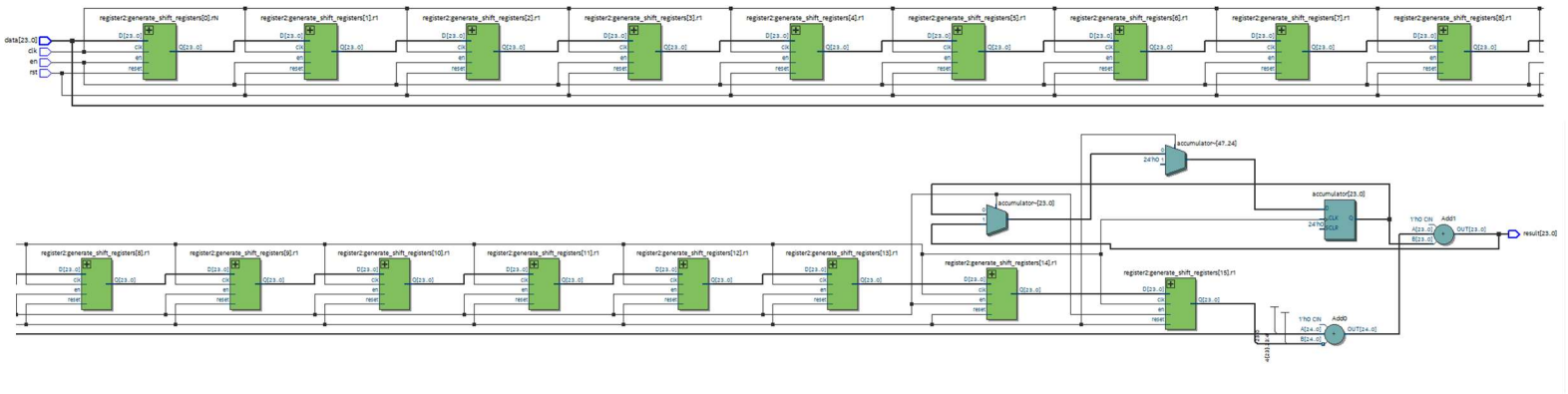

Figure 4: Block diagram for task 3 with n = 16

Results

Task 1:

No additional modules or testbench was needed for the completion of this task.

Task 2:

For the first part of task 2, we developed the registers to shift and store 8 consecutive samples. From these 8 samples, data is divided by 8 for each output of a register and added together for the resulting output. In Figure 6, the waveform simulation shows how Q outputs D on posedge clk. As well as Figure 5, showing the waveform simulation of the filter, averaging samples given, outputting to result.

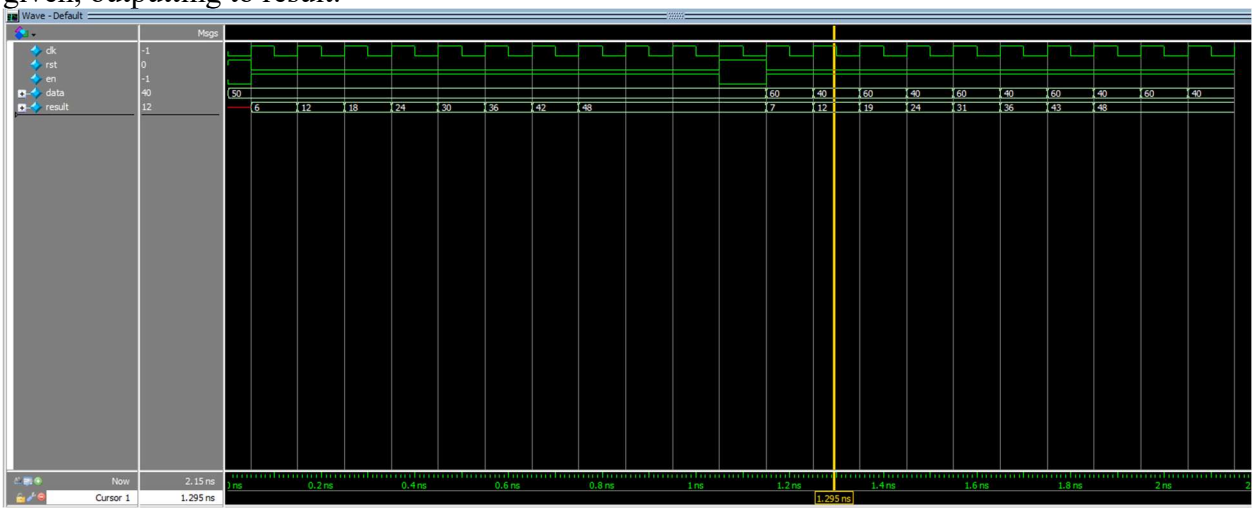

Figure 5: Waveform simulation for fir Filter in task 1

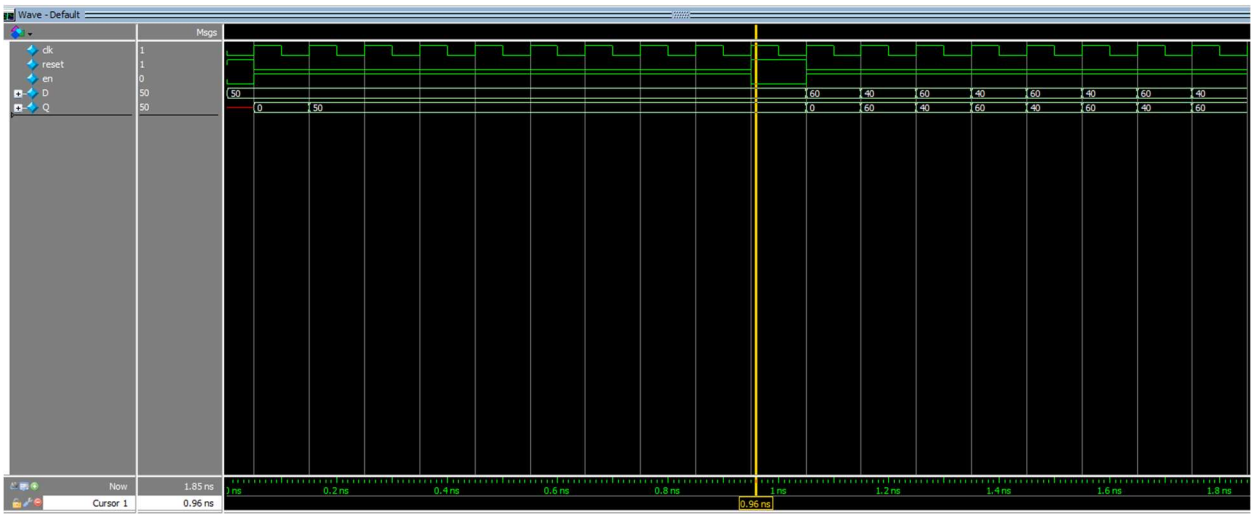

Figure 6: Waveform simulation for register in task 1

Task 3:

In this tasks, we performed simulation test on the register2 module, as well as the parameterized FIR filter that uses a buffer sized n, and an accumulator. This method allows for n samples to be average and filter out the noise. In Figure 8, the simulation for the register module can be seen as well as Figure 7, for the FIR filter with N = 16, the values are taken, shifted across registers, and averaged out for the output of result.

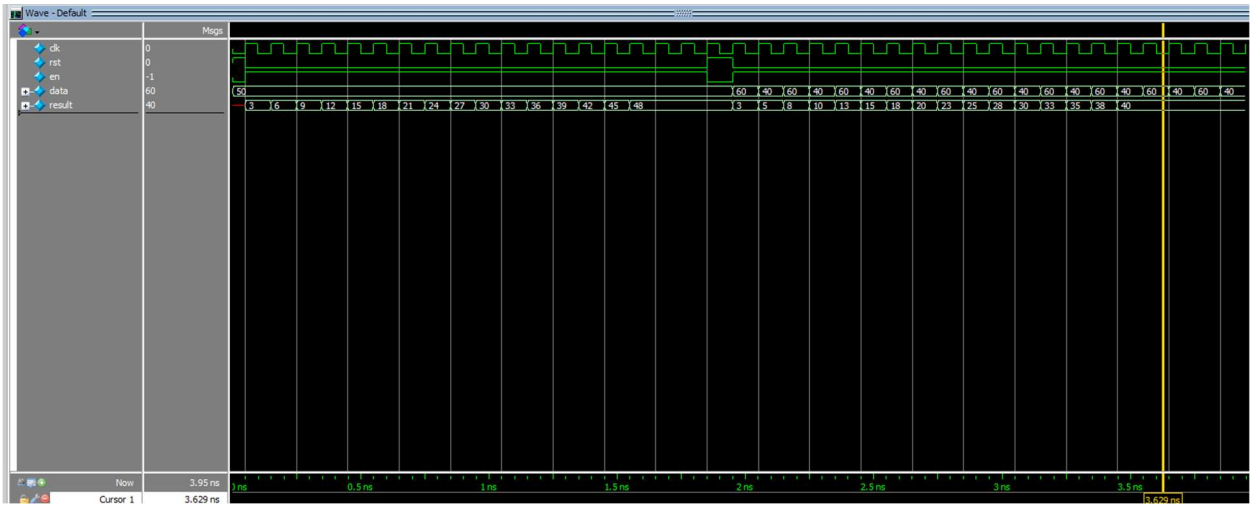

Figure 7: Waveform simulation for parameterized fir filter for task 2

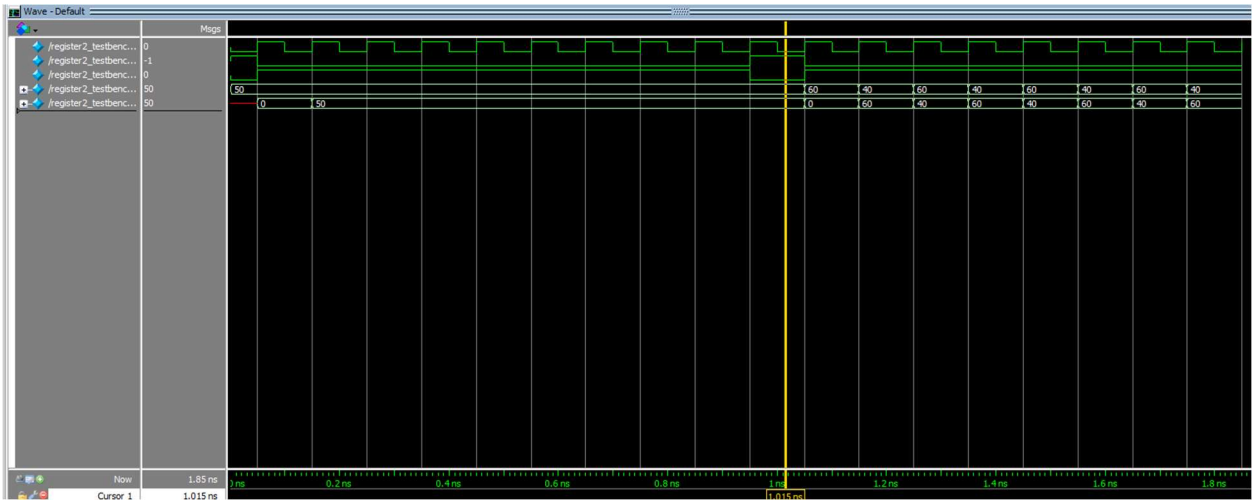

Figure 8: Waveform simulation for register2 module for task 2

Final Product

The overarching goal of this project was to design a FIR filter to filter out the noise created in the task 1 modules as well as parameterizing the FIR filter to be able to average a given number of samples. In task 1, we were able to comprehend the connections between the AUDIO CODEC and our interface, how noise is added, and how to connect our filter to eliminate the noise. From task 2, we understood how the filter operates and was able to create a buffer that is sized n, using shift registers to move the data through n registers. This way, we were able to keep track of the oldest data value and compute the average with the values in all of the registers. We didn't have many issues implementing the FIR filter and we enjoyed the aspect of using frequency and audio in this lab as an introduction to how FPGA systems record, taken in, and output audio.

Appendix: SystemVerilog Code

1) firFilter.sv (task 1)

```
1   //Khoa Tran and Ravi Sangani
2   //05/22/2020
3   //Lab 5, Task 1
4   //Module firFilter represents a fir Filter for 8 samples of inputs
5   //averaging them in order to filter out the noise created.
6   //The fir filter uses a series of shift registers to shift the inputs, divide
7   //and combine to output to the result
8   module firFilter(clk, rst, en, data, result);
9      input logic clk, rst, en;
10     input logic [23:0] data;
11     output logic [23:0] result;
12
13     logic [23:0] temp1, temp2, temp3, temp4, temp5, temp6, temp7;
14     logic [23:0] combine1, combine2, combine3, combine4, combine5, combine6;
15
16     //instatiation of register modules presenting a series of shift registers
17     register r1(.clk, .reset(rst), .en, .D(data), .Q(temp1));
18     register r2(.clk, .reset(rst), .en, .D(temp1), .Q(temp2));
19     register r3(.clk, .reset(rst), .en, .D(temp2), .Q(temp3));
20     register r4(.clk, .reset(rst), .en, .D(temp3), .Q(temp4));
21     register r5(.clk, .reset(rst), .en, .D(temp4), .Q(temp5));
22     register r6(.clk, .reset(rst), .en, .D(temp5), .Q(temp6));
23     register r7(.clk, .reset(rst), .en, .D(temp6), .Q(temp7));
24
25     //divide each registers' output and add them together
26     assign combine1 = ({{{3{data[23]}}, data[23:3]}) + ({{{3{temp1[23]}}, temp1[23:3]}) + ({{{3
       {temp2[23]}}, temp2[23:3]}) + ({{{3{temp3[23]}}, temp3[23:3]}) + ({{{3{temp4[23]}}, temp4[23:3
       ]}) + ({{{3{temp5[23]}}, temp5[23:3]}) + ({{{3{temp6[23]}}, temp6[23:3]}));
27     //output result as combination of all
28     assign result = ((combine1) + ({{{3{temp7[23]}}, temp7[23:3]}));
29
30  endmodule
31
32  //Module firFilter represents a fir Filter for 8 samples of inputs
33  //averaging them in order to filter out the noise created.
34  //The fir filter uses a series of shift registers to shift the inputs, divide
35  //and combine to output to the result
36  module firFilter_testbench();
37     logic clk, rst, en;
38     logic [23:0] data;
39     logic [23:0] result;
40
41     //device under test
42     firFilter dut(.clk, .rst, .en, .data, .result);
43
44     parameter clock_period = 100;
45
46     initial begin
47        clk <= 0;
48        forever #(clock_period /2) clk <= ~clk;
49     end
50
51     //initial simulation
52     initial begin
53        rst <= 1; en <= 0; data <= 23'd50; @(posedge clk);
54        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
55        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
56        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
57        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
58        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
59        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
60        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
61        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
62        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
63        rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
64        rst <= 1; en <= 0; data <= 23'd50; @(posedge clk);
65        rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
66        rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
67        rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
68        rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
69        rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
70        rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
71        rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
72        rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
73        rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
74        rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
```



```
75        $stop;
76     end
77  endmodule
```

2) register.sv (task 1)

```
1   //Khoa Tran and Ravi Sangani
2   //05/22/2020
3   //Lab 5, Task 1
4   //Module register outputs 0 on reset and the input of D on enable (en).
5   //The output is on variable Q and using sequential DFF, Q output is
6   //on each posedge clk
7   module register(clk, reset, en, D, Q);
8      input logic clk, reset, en;
9      input logic [23:0] D;
10     output logic [23:0] Q;
11
12     always_ff @(posedge clk)
13        begin
14           if (reset)
15              Q <= 0;
16           else if (en)
17              Q <= D;
18        end
19  endmodule
20
21  //Module register is a testbench to see if the outputs on
22  //the register module of Q is correct along the posedge
23  //clk with a series of inputs on reset, en, and D
24  module register_testbench();
25     logic clk, reset, en;
26     logic [23:0] D;
27     logic [23:0] Q;
28
29     //device under test
30     register dut(.clk, .reset, .en, .D, .Q);
31
32     parameter clock_period = 100;
33
34     initial begin
35        clk <= 0;
36        forever #(clock_period /2) clk <= ~clk;
37     end
38
39     //initial simulation
40        initial begin
41           reset <= 1; en <= 0; D <= 23'd50; @(posedge clk);
42           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
43           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
44           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
45           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
46           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
47           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
48           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
49           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
50           reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
51           reset <= 1; en <= 0; D <= 23'd50; @(posedge clk);
52           reset <= 0; en <= 1; D <= 23'd60; @(posedge clk);
53           reset <= 0; en <= 1; D <= 23'd40; @(posedge clk);
54           reset <= 0; en <= 1; D <= 23'd60; @(posedge clk);
55           reset <= 0; en <= 1; D <= 23'd40; @(posedge clk);
56           reset <= 0; en <= 1; D <= 23'd60; @(posedge clk);
57           reset <= 0; en <= 1; D <= 23'd40; @(posedge clk);
58           reset <= 0; en <= 1; D <= 23'd60; @(posedge clk);
59           reset <= 0; en <= 1; D <= 23'd40; @(posedge clk);
60           $stop;
61     end
62  endmodule
63
```

3) nFirFilter.sv (task 2)

```
1   //Khoa Tran and Ravi Sangani
2   //05/22/2020
3   //Lab 5, Task 2
4   //Module nFirFilter parameterize the firFilter module by allowing inputs of
5   //n size for the number of samples to average and combine. In this module,
6   //to parameterize the firFilter, generate allows for a for loop, calling
7   //register2 module n-1 times and storing the output on a 2D array, that has
8   //n locations and storing a 24 bit value given from the register
9   module nFirFilter#(parameter n=16)(clk, rst, en, data, result);
10     input logic clk, rst, en;
11     input logic [23:0] data;
12     output logic [23:0] result;
13
14
15     logic [23:0] resultTemp;
16     logic [n-1:0][23:0] temp;
17     logic [23:0] dividedData;
18     logic signed [23:0] last;
19     logic [23:0] accumulator, addTemp;
20
21     //divide input data by n
22     assign dividedData = {{$clog2(n){data[23]}}, data[23:$clog2(n)]};
23
24     //generate statement
25     genvar i;
26     generate
27        //for loop from 0 to n-1
28        for (i = 0; i <= n - 1; i = i+1) begin: generate_shift_registers
29           //at first index, store dividedData into the first index of 2D temp array
30           if (i == 0)
31              register2 rN(.clk, .reset(rst), .en, .D(dividedData), .Q(temp[0][23:0]));
32           //otherwise shift the values of the index
33           else
34              register2 r1(.clk, .reset(rst), .en, .D(temp[i-1][23:0]), .Q(temp[i][23:0]));
35        end
36     endgenerate
37
38     //get the last value in 2d temp array
39     assign last = temp[n-1][23:0];
40     //get dividedData - last to keep the average for n samples
41     assign resultTemp = dividedData - last;
42
43     //sequential DFF for accumulator getting resultTemp + previous accumulator value on
   posedge clk
44     always_ff @(posedge clk) begin
45        if (rst)
46           accumulator <= 0;
47        else if (en)
48           accumulator <= addTemp;
49     end
50
51     assign addTemp = resultTemp + accumulator;
52     //output the result which is the total of accumulator plus resultTemp
53     assign result = addTemp;
54
55  endmodule
56
57  //Testbench for nFirFilter, parameterizing firFilter
58  //giving n = 16, and a series of inputs on data and en,
59  //seeing if the output on result is correct along
60  //posedge clk
61  module nFirFilter_testbench#(parameter n = 16)();
62     logic clk, rst, en;
63     logic [23:0] data;
64     logic [23:0] result;
65
66     //device under test
67     nFirFilter dut(.clk, .rst, .en, .data, .result);
68        defparam dut.n = n;
69
70     parameter clock_period = 100;
71
72     initial begin
73        clk <= 0;
74        forever #(clock_period /2) clk <= ~clk;
75     end
```

```
 76
 77      //initial simulation
 78      initial begin
 79         rst <= 1; en <= 0; data <= 23'd50; @(posedge clk);
 80         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 81         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 82         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 83         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 84         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 85         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 86         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 87         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 88         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 89         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 90         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 91         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 92         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 93         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 94         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 95         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 96         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 97         rst <= 0; en <= 1; data <= 23'd50; @(posedge clk);
 98         rst <= 1; en <= 0; data <= 23'd50; @(posedge clk);
 99         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
100         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
101         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
102         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
103         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
104         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
105         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
106         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
107         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
108         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
109         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
110         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
111         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
112         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
113         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
114         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
115         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
116         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
117         rst <= 0; en <= 1; data <= 23'd60; @(posedge clk);
118         rst <= 0; en <= 1; data <= 23'd40; @(posedge clk);
119         $stop;
120      end
121   endmodule
122
```

4) register2.sv (task 2)

```
1   //Khoa Tran and Ravi Sangani
2   //05/22/2020
3   //Lab 5, Task 2
4   //Module register2 outputs 0 on reset and the input of D on enable (en).
5   //The output is on variable Q and using sequential DFF, Q output is
6   //on each posedge clk
7   module register2(clk, reset, en, D, Q);
8      input logic clk, reset, en;
9      input logic [23:0] D;
10     output logic [23:0] Q;
11
12     always_ff @(posedge clk)
13        begin
14           if (reset)
15              Q <= 0;
16           else if (en)
17              Q <= D;
18        end
19  endmodule
20
21  //Module register is a testbench to see if the outputs on
22  //the register module of Q is correct along the posedge
23  //clk with a series of inputs on reset, en, and D
24  module register2_testbench();
25     logic clk, reset, en;
26     logic [23:0] D;
27     logic [23:0] Q;
28
29     register2 dut(.clk, .reset, .en, .D, .Q);
30
31     parameter clock_period = 100;
32
33     initial begin
34        clk <= 0;
35        forever #(clock_period /2) clk <= ~clk;
36     end
37
38        initial begin
39        reset <= 1; en <= 0; D <= 23'd50; @(posedge clk);
40        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
41        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
42        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
43        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
44        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
45        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
46        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
47        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
48        reset <= 0; en <= 1; D <= 23'd50; @(posedge clk);
49        reset <= 1; en <= 0; D <= 23'd50; @(posedge clk);
50        reset <= 0; en <= 1; D <= 23'd60; @(posedge clk);
51        reset <= 0; en <= 1; D <= 23'd40; @(posedge clk);
52        reset <= 0; en <= 1; D <= 23'd60; @(posedge clk);
53        reset <= 0; en <= 1; D <= 23'd40; @(posedge clk);
54        reset <= 0; en <= 1; D <= 23'd60; @(posedge clk);
55        reset <= 0; en <= 1; D <= 23'd40; @(posedge clk);
56        reset <= 0; en <= 1; D <= 23'd60; @(posedge clk);
57        reset <= 0; en <= 1; D <= 23'd40; @(posedge clk);
58        $stop;
59     end
60  endmodule
61
```

5) DE1_SoC.sv (task 1,2,3)

```
1   //Khoa Tran and Ravi Sangani
2   //05/22/2020
3   //Lab 5, Task 1
4   //Module DE1_SoC uses an audio coder/decoder in order to recieve audio inputs
5   //and record it, outputting the given input sound. This DE1_SoC module has KEY3 as the
6   //reset, KEY0 as the input for task 1, KEY1 as the input for task 2, and KEY2 as the input
7   //for task 3. In task 1, noise is added into the sound of the input audio, and in task 2
8   //a fir filter is implemented in order to reduce the noise by averaging 8 consecutive samples
9   //and for task 3, the parameterized version of fir filter allows for averaging n samples
10  //and reduce the noise even further or less based on the n value.
11  module DE1_SoC #(parameter n = 16)(CLOCK_50, CLOCK2_50, FPGA_I2C_SCLK, FPGA_I2C_SDAT,
12     AUD_XCK, AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT,
13     KEY, SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, LEDR);
14
15     input logic CLOCK_50, CLOCK2_50;
16     input logic [3:0] KEY;
17     input logic [9:0] SW;
18     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
19     output logic [9:0] LEDR;
20
21     // I2C Audio/Video config interface
22     output FPGA_I2C_SCLK;
23     inout FPGA_I2C_SDAT;
24     // Audio CODEC
25     output AUD_XCK;
26     input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
27     input AUD_ADCDAT;
28     output AUD_DACDAT;
29
30     // Local wires
31     logic read_ready, write_ready, read, write;
32     logic signed [23:0] readdata_left, readdata_right;
33     logic signed [23:0] writedata_left, writedata_right;
34     logic signed [23:0] task2_left, task2_right, task3_left, task3_right;
35     logic signed [23:0] noisy_left, noisy_right;
36     logic reset;
37
38     logic [23:0] noise, filtered1Left, filtered1Right, filtered2Left, filtered2Right;
39     noise_gen noise_generator (.clk(CLOCK_50), .en(read), .rst(reset), .out(noise));
40     assign noisy_left = readdata_left + noise;
41     assign noisy_right = readdata_right + noise;
42
43     //instatiation of firFilter for both noisy_left and noisy_right as inputs to data and
   result on task2_left and task2_right
44     firFilter filterL(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
   filtered1Left));
45     firFilter filterR(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
   filtered1Right));
46     assign task2_left = filtered1Left;
47     assign task2_right = filtered1Right;
48
49     //instatiation of nFirFilter with n as 16 for both noisy_left and noisy_right as inputs
   to data and result on task3_left and task3_right
50     nFirFilter filter2L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
   filtered2Left));
51        defparam filter2L.n = n;
52     nFirFilter filter2R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
   filtered2Right));
53        defparam filter2R.n = n;
54     assign task3_left = filtered2Left;
55     assign task3_right = filtered2Right;
56
57     //combinational logic for KEY to see which key calls which task
58     always_comb begin
59        case(KEY[2:0])
60           3'b110: begin // KEY0 outputs noise
61              writedata_left = noisy_left;
62              writedata_right = noisy_right;
63           end
64           3'b101: begin // KEY1 outputs task2 filtered noise
65              writedata_left = task2_left;
66              writedata_right = task2_right;
67           end
68           3'b011: begin // KEY2 outputs task3 filtered noise
69              writedata_left = task3_left;
70              writedata_right = task3_right;
```

```
71              end
72              default: begin // default output raw data
73                 writedata_left = readdata_left;
74                 writedata_right = readdata_right;
75              end
76           endcase
77        end
78
79        assign reset = ~KEY[3];
80        assign {HEX0, HEX1, HEX2, HEX3, HEX4, HEX5} = '1;
81        assign LEDR = SW;
82
83        // only read or write when both are possible
84        assign read = read_ready & write_ready;
85        assign write = read_ready & write_ready;
86
87     ////////////////////////////////////////////////////////////////////////
88     // Audio CODEC interface.
89     //
90     // The interface consists of the following wires:
91     // read_ready, write_ready - CODEC ready for read/write operation
92     // readdata_left, readdata_right - left and right channel data from the CODEC
93     // read - send data from the CODEC (both channels)
94     // writedata_left, writedata_right - left and right channel data to the CODEC
95     // write - send data to the CODEC (both channels)
96     // AUD_* - should connect to top-level entity I/O of the same name.
97     //         These signals go directly to the Audio CODEC
98     // I2C_* - should connect to top-level entity I/O of the same name.
99     //         These signals go directly to the Audio/Video Config module
100    ////////////////////////////////////////////////////////////////////////////////
101       clock_generator my_clock_gen(
102          // inputs
103          CLOCK2_50,
104          1'b0,
105
106          // outputs
107          AUD_XCK
108       );
109
110       audio_and_video_config cfg(
111          // Inputs
112          CLOCK_50,
113          1'b0,
114
115          // Bidirectionals
116          FPGA_I2C_SDAT,
117          FPGA_I2C_SCLK
118       );
119
120       audio_codec codec(
121          // Inputs
122          CLOCK_50,
123          1'b0,
124
125          read, write,
126          writedata_left, writedata_right,
127
128          AUD_ADCDAT,
129
130          // Bidirectionals
131          AUD_BCLK,
132          AUD_ADCLRCK,
133          AUD_DACLRCK,
134
135          // Outputs
136          read_ready, write_ready,
137          readdata_left, readdata_right,
138          AUD_DACDAT
139       );
140
141    endmodule
142
```