Khoa Tran and Ravi Sangani
EE 371
June 7, 2021
Lab 6 Report

Procedure

Task #1

In order to approach this task, we were not sure if we should implement a game or an audio visualizer. However, after discussing the aspects of an audio visualizer, we were certain that we wanted to implement an audio visualizer as it was fresh in our minds from the previous labs and we wanted to visualize and observe the differences between audio with noise, filtered, and without noise. As a result, we drew a block diagram, in figure 1, displaying the connections between the different drivers of the VGA and audio input.

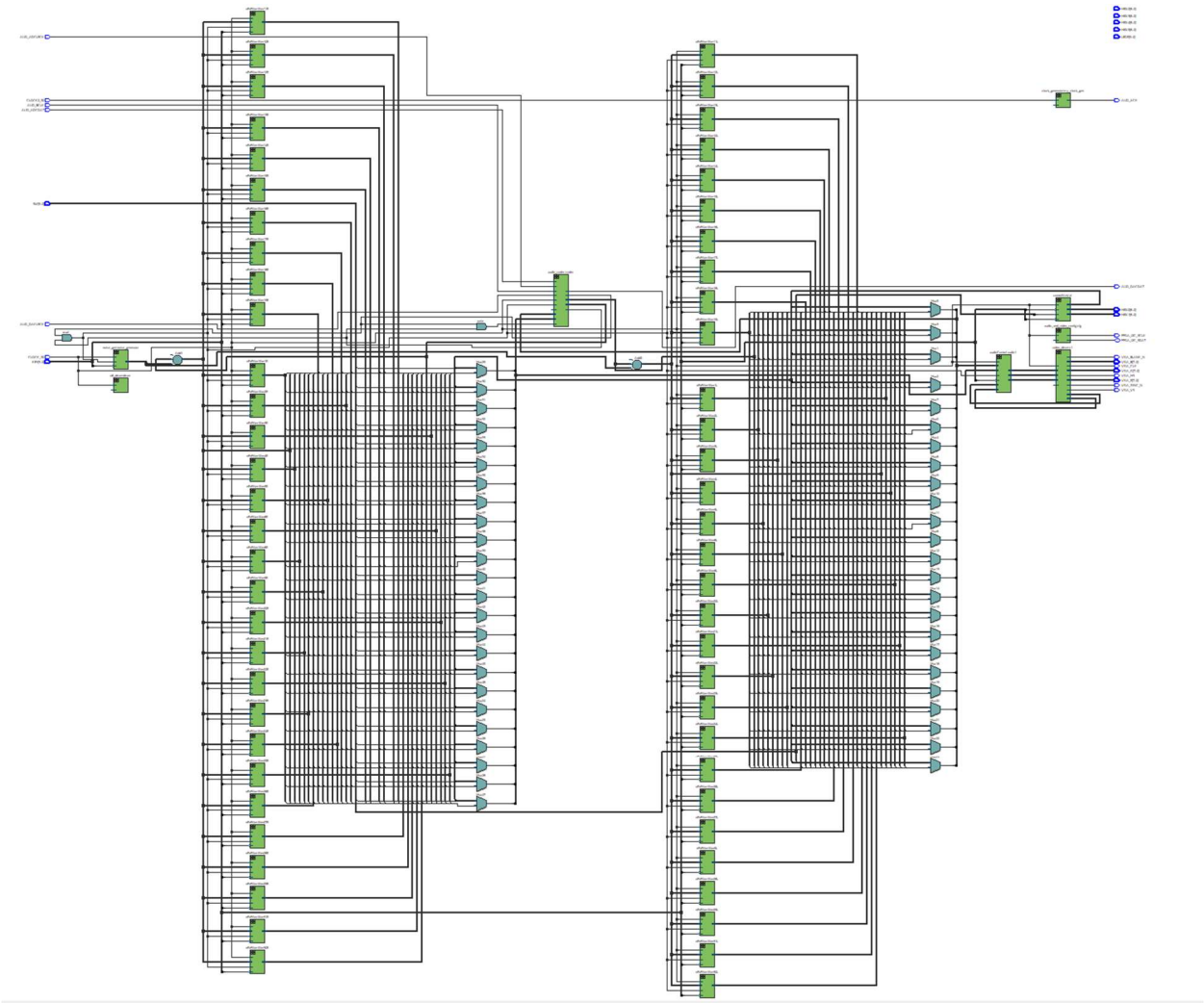

Figure 1: Block diagram for task 1

We first implemented a display counter that allows for the user to decide the number of nFirFilter that should be visualized. The number of filters is displayed on HEX1 and HEX0. The control of the VGA output is based on the input of the selected sound to be displayed and outputs 16 rectangles that is 16 consecutive samples, each has differences in color, width, and height based on audio input.

Results

Task 1:

For the simulation of the audio visualizer, we first simulated the nFirFilter module from lab 5, taking in data and outputting the average of n consecutive samples on result. This value of n is given by the user in the controlN module. However, the nFirFilter uses the module of register2, seen simulated in Figure 3.

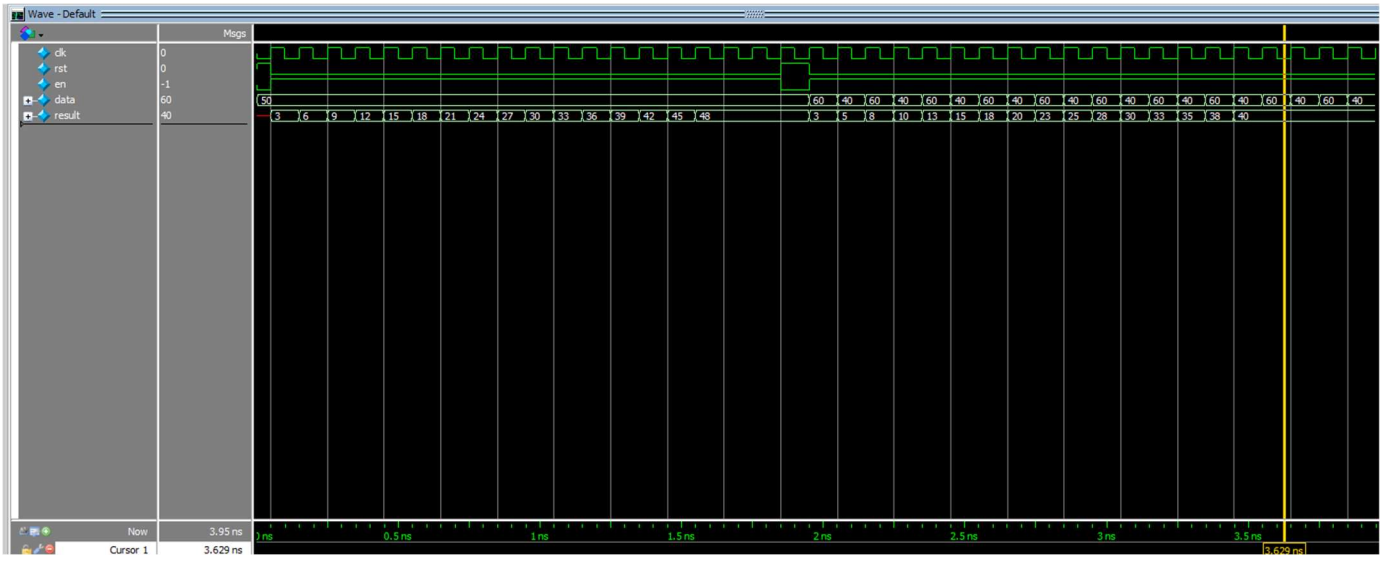

Figure 2: Waveform simulation on nFirFilter

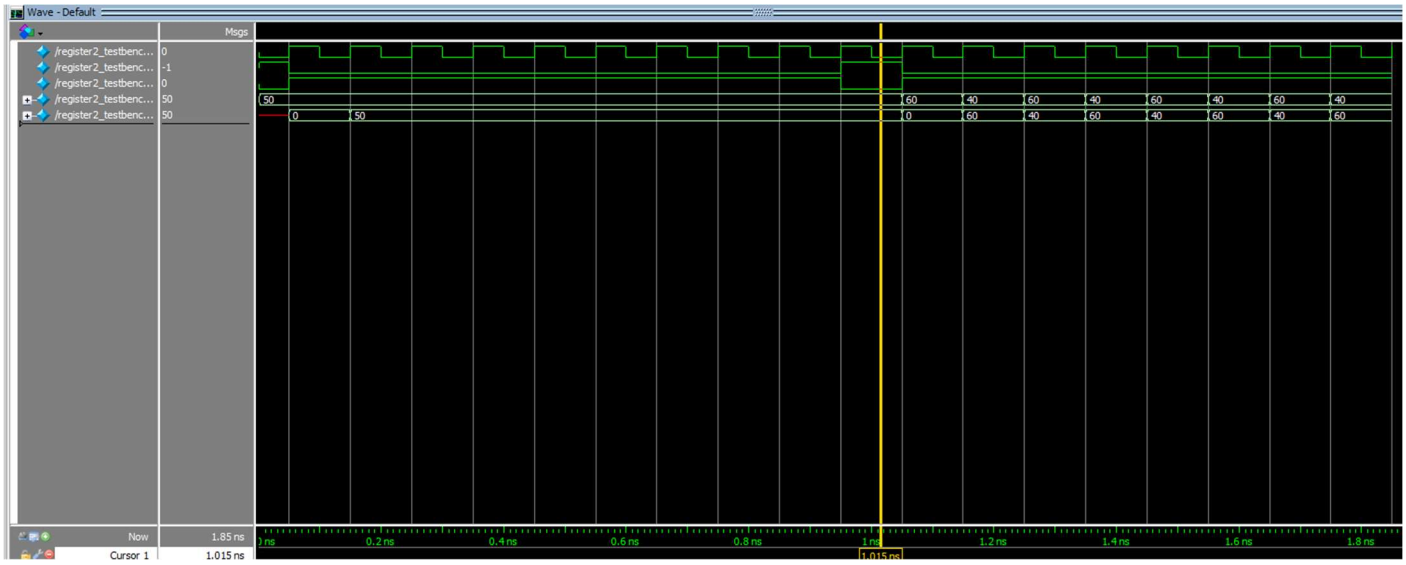

Figure 3: Waveform simulation on register2

As seen in figure 4, we then created controlN module and simulated it as it takes in inputs of start, up, and down to reset, increment, and decrement a counter that its value is displayed on HEX1 and HEX0 in decimal. The count is an output for the DE1_SoC for a state logic on deciding the number of filters to display, or noise data, or the original sound with 0 filters. This way, the user can decide on viewing the noise data, filtered, or original sound in order to distinguish the differences between the audio sound of left and right channels.

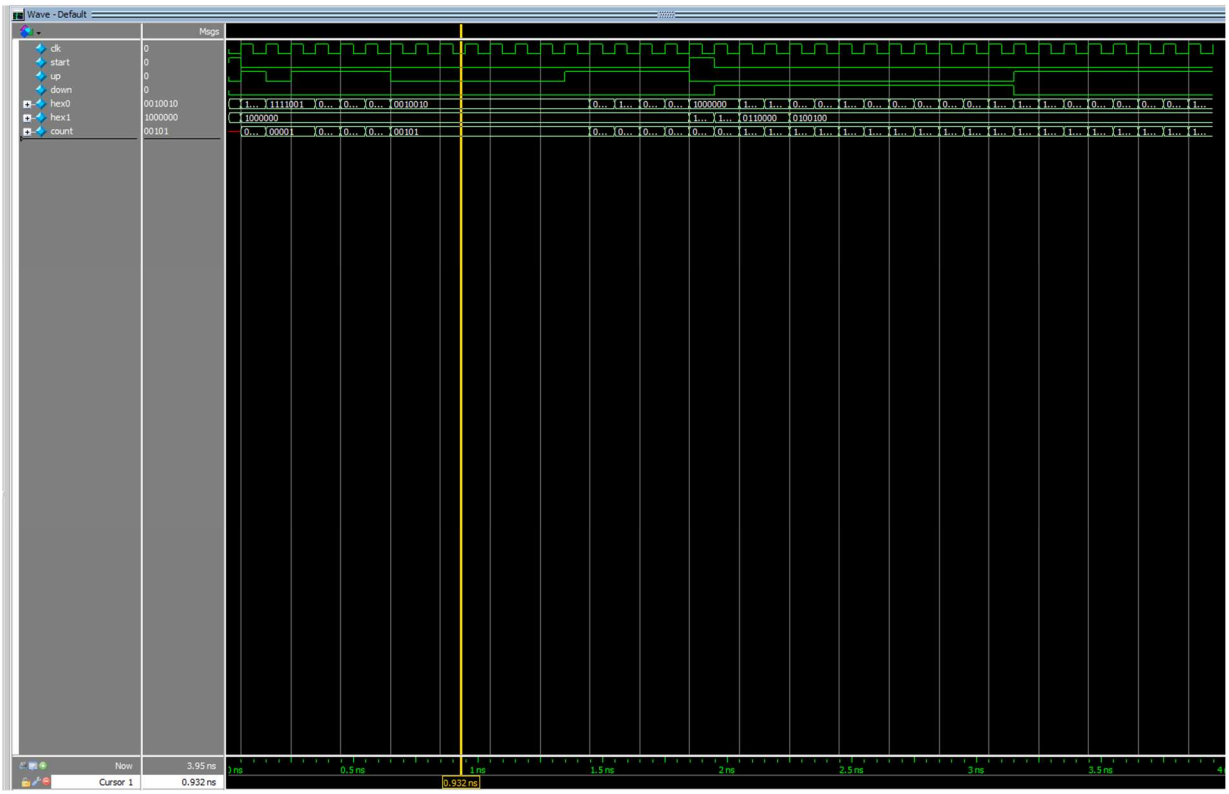

Figure 4: Waveform simulation for controlN module

Lastly, we implemented an audioControl module that takes in inputs of x and y from the video driver of the VGA, and decide on the rgb color at the (x,y) coordinate given in order to display a audio visualizer. The audio visualizer is based on the input of the audio chosen in DE1_SoC and composes of 16 consecutive rectangles or bars that vary in width, height, color. The determination of each bar's width, height, and color is based on the value of the 24 bit binary input of the left and right channels of the type of audio chosen by the user. With noise or without noise, the simulation and visualization of each bar is distinguishingly different. As seen in Figure 5, we simulated various x and y points as well as various audio data to see if the rgb output is correct based on the current coordinate point and the audio data. Using a specialized algorithm seen in the appendix, visualization of each audio sample is unique and varies based on different existence of notes in the sound. Audio Control module also uses the register module, simulated in figure 6 below.

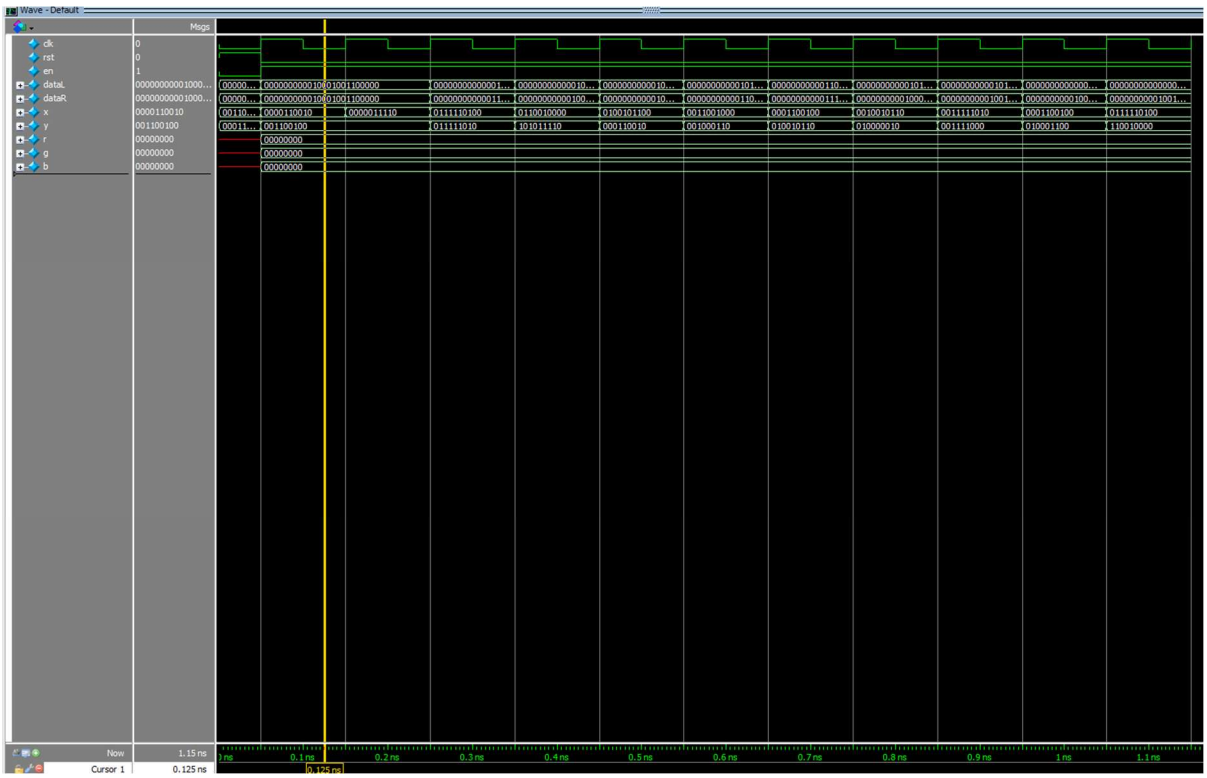

Figure 5: Waveform simulation of audioControl

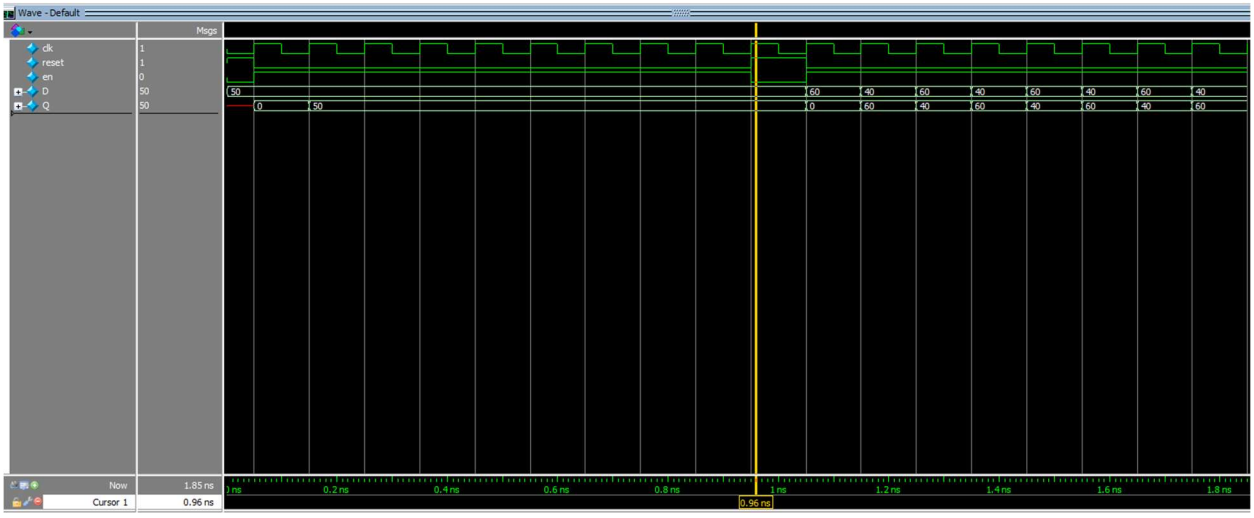

Figure 6: Waveform simulation of register module

Final Product

The overarching goal of this project was to create a project with at least one input device and at least one output device. We designed and coded a project with the N8 Controller and audio as input, and the VGA as output. Our project has the functionality of using the input audio (turned into sampled left and right values using the driver) to display 16 rectangles with widths, heights, and colors based the current 16 audio samples in the series of registers. There is also the additional functionality of using user-determined number of filters or noise to see how that affects the displaying of the 16 rectangles. This means that you can either display the normal audio data, audio data with noise, or audio data with filters. The differences between these were clear in the rectangles, with varying widths, heights, and colors. From this project, we understood how use shift registers to move and save audio data, and how to meaningfully represent audio data in a visual form. We did not have many issues in the designing and coding of this project.

Appendix: SystemVerilog Code

1) audioControl.sv

Date: June 07, 2021 audioControl.sv Project: DE1_SoC

```
1   //Khoa Tran and Ravi Sangani
2   //06/07/2020
3   //Lab 6, Task 1
4
5   //This module takes in clock, reset, and enable, which are 1 bit in size.
6   //It also has inputs dataL and dataR, which are 24 bits in size, and are the two-channel
7   //audio sample values. This module also takes in x and y, which are 10 bits and 9 bits
8   //in size respectively, representing the current coordinate. The outputs are r, g, and b,
9   //which are 8 bits in size, representing the color to be outputted based on current x and y.
10  //Overall, this module enables the creation of 16 adjacent rectangles whose heights, widths,
11  //and colors are based on the input audio sample. For one incoming two-channel audio sample,
12  //we manipulate two rectangles side by side, the left rectangle for left audio and right
    rectangle
13  //for right audio. We use 16 registers to shift the incoming audio samples through by storing
14  //the samples and then passing into the next register on each clock cycle. This means that
    all
15  //the even numbered rectangles are storing 16 consecutive right audio samples,
16  //and all the odd numbered rectangles are storing 16 consecutive left audio samples.
17
18  module audioControl(clk, rst, en, dataL, dataR, x, y, r, g, b);
19     input logic clk, rst, en;
20     input logic [23:0] dataL, dataR;
21     input logic [9:0] x;
22     input logic [8:0] y;
23     output logic [7:0] r, g, b;
24
25     logic [23:0] tempL1, tempL2, tempL3, tempL4, tempL5, tempL6, tempL7;
26     logic [23:0] tempR1, tempR2, tempR3, tempR4, tempR5, tempR6, tempR7;
27
28     //instatiation of register modules presenting a series of shift registers
29     register rL1(.clk, .reset(rst), .en, .D(dataL), .Q(tempL1));
30     register rL2(.clk, .reset(rst), .en, .D(tempL1), .Q(tempL2));
31     register rL3(.clk, .reset(rst), .en, .D(tempL2), .Q(tempL3));
32     register rL4(.clk, .reset(rst), .en, .D(tempL3), .Q(tempL4));
33     register rL5(.clk, .reset(rst), .en, .D(tempL4), .Q(tempL5));
34     register rL6(.clk, .reset(rst), .en, .D(tempL5), .Q(tempL6));
35     register rL7(.clk, .reset(rst), .en, .D(tempL6), .Q(tempL7));
36
37     register rR1(.clk, .reset(rst), .en, .D(dataR), .Q(tempR1));
38     register rR2(.clk, .reset(rst), .en, .D(tempR1), .Q(tempR2));
39     register rR3(.clk, .reset(rst), .en, .D(tempR2), .Q(tempR3));
40     register rR4(.clk, .reset(rst), .en, .D(tempR3), .Q(tempR4));
41     register rR5(.clk, .reset(rst), .en, .D(tempR4), .Q(tempR5));
42     register rR6(.clk, .reset(rst), .en, .D(tempR5), .Q(tempR6));
43     register rR7(.clk, .reset(rst), .en, .D(tempR6), .Q(tempR7));
44
45     logic [9:0] checker1, checker2, checker3, checker4, checker5, checker6, checker7,
    checker8, checker9, checker10, checker11, checker12, checker13, checker14, checker15,
    checker16;
46
47     //Each checker value is the summation of the previous checker value and the value
    represented by the first five digits
48     //of the audio sample in the current corresponding register.
49     assign checker1 = dataL[4:0];
```

```
49      assign checker1 = dataL[4:0];
50      assign checker2 = checker1 + dataR[4:0];
51      assign checker3 = checker2 + tempL1[4:0];
52      assign checker4 = checker3 + tempR1[4:0];
53      assign checker5 = checker4 + tempL2[4:0];
54      assign checker6 = checker5 + tempR2[4:0];
55      assign checker7 = checker6 + tempL3[4:0];
56      assign checker8 = checker7 + tempR3[4:0];
57      assign checker9 = checker8 + tempL4[4:0];
58      assign checker10 = checker9 + tempR4[4:0];
59      assign checker11 = checker10 + tempL5[4:0];
60      assign checker12 = checker11 + tempR5[4:0];
61      assign checker13 = checker12 + tempL6[4:0];
62      assign checker14 = checker13 + tempR6[4:0];
63      assign checker15 = checker14 + tempL7[4:0];
64      assign checker16 = checker15 + tempR7[4:0];
65
66      //This always_ff block handles the logic of figuring out the respective r, g, b values
    based on
67      //the dataL and dataR. The above "checkers" are used in manipulating the widths of each
    rectangle
68      //by only displaying the calculated RGB color if the input x is equal to or past the
    checker threshold.
```



```
69      //The height is manipulated by only displaying the calculated RGB value if the input y
    is greater than
70      //or equal to the value represented by taking the middle 6 digits of audio sample in
    current corresponding register.
71      //For the 16 audio samples, left audio values are red dominant and right audio values
    are blue dominant. The
72      //proportion of green is determined by looking at the value represented by the last
    three digits of
73      //corresponding audio sample. This is also what would change if any noise is added.
74      always_ff @(posedge clk) begin
75          if (rst) begin
76              r <= 8'd0;
77              g <= 8'd0;
78              b <= 8'd0;
79              end
80          else begin
81              if ((x < checker1) && (y >= dataL[12:5])) begin
82                  r <= dataL[20:13];
83                  g <= {dataL[23:21], 5'b00000};
84                  b <= 8'd0;
85                  end
86              else if ((x >= checker1) && (x < checker2) && (y >= dataR[13:8]))
87                  begin
88                      r <= 8'd0;
89                      g <= {dataR[23:21], 5'b00000};
90                      b <= dataR[20:13];
```

```
                g <= {dataR[23:21], 5'b00000};
                b <= dataR[20:13];
            end
        else if ((x >= checker2) && (x < checker3) && (y >= tempL1[13:8]))
            begin
                r <= tempL1[20:13];
                g <= {tempL1[23:21], 5'b00000};
                b <= 8'd0;
            end
        else if ((x >= checker3) && (x < checker4) && (y >= tempR1[13:8]))
            begin
                r <= 8'd0;
                g <= {tempR1[23:21], 5'b00000};
                b <= tempR1[20:13];
            end
        else if ((x >= checker4) && (x < checker5) && (y >= tempL2[13:8]))
            begin
                r <= tempL2[20:13];
                g <= {tempL2[23:21], 5'b00000};
                b <= 8'd0;
            end
        else if ((x >= checker5) && (x < checker6) && (y >= tempR2[13:8]))
            begin
                r <= 8'd0;
                g <= {tempR2[23:21], 5'b00000};
                b <= tempR2[20:13];
            end
        else if ((x >= checker6) && (x < checker7) && (y >= tempL3[13:8]))
            begin
                r <= tempL3[20:13];
                g <= {tempL3[23:21], 5'b00000};
                b <= 8'd0;
            end
        else if ((x >= checker7) && (x < checker8) && (y >= tempR3[13:8]))
            begin
                r <= 8'd0;
                g <= {tempR3[23:21], 5'b00000};
                b <= tempR3[20:13];
            end
        else if ((x >= checker8 ) && (x < checker9) && (y >= tempL4[13:8]))
            begin
                r <= tempL4[20:13];
                g <= {tempL4[23:21], 5'b00000};
                b <= 8'd0;
            end
        else if ((x >= checker9) && (x < checker10) && (y >= tempR4[13:8]))
            begin
                r <= 8'd0;
                g <= {tempR4[23:21], 5'b00000};
                b <= tempR4[20:13];
            end
        else if ((x >= checker10) && (x < checker11) && (y >= tempL5[13:8]))
```

```
141                 begin
142                     r <= tempL5[20:13];
143                     g <= {tempL5[23:21], 5'b00000};
144                     b <= 8'd0;
145                 end
146             else if ((x >= checker11) && (x < checker12) && (y >= tempR5[13:8]))
147                 begin
148                     r <= 8'd0;
149                     g <= {tempR5[23:21], 5'b00000};
150                     b <= tempR5[20:13];
151                 end
152             else if ((x >= checker12) && (x < checker13) && (y >= tempL6[13:8]))
153                 begin
154                     r <= tempL6[20:13];
155                     g <= {tempL6[23:21], 5'b00000};
156                     b <= 8'd0;
157                 end
158             else if ((x >= checker13) && (x < checker14) && (y >= tempR6[13:8]))
159                 begin
160                     r <= 8'd0;
161                     g <= {tempR6[23:21], 5'b00000};
162                     b <= tempR6[20:13];
163                 end
164             else if ((x >= checker14) && (x < checker15) && (y >= tempL7[13:8]))
165                 begin
166                     r <= tempL7[20:13];
167                     g <= {tempL7[23:21], 5'b00000};
168                     b <= 8'd0;
169                 end
170             else if ((x >= checker15) && (x < checker16) && (y >= tempR7[13:8]))
171                 begin
172                     r <= 8'd0;
173                     g <= {tempR7[23:21], 5'b00000};
174                     b <= tempR7[20:13];
175                 end
176             else
177                 begin
178                     r <= 8'd0;
179                     g <= 8'd0;
180                     b <= 8'd0;
181                 end
182             end
183         end
184
185 endmodule
186
187 //This testbench ensures full functionality of the above module. In specific, we test
188 //arbitrary two-channel audio values and input x and y, to make sure that the registers
    work as expected,
189 //the rectangles are manipulated properly in height, width, and color, and that reset works.
190 module audioControl_testbench ();
```

```
189   //the rectangles are manipulated properly in height, width, and color, and that reset works.
190   module audioControl_testbench ();
191      logic clk, rst, en;
192      logic [23:0] dataL, dataR;
193      logic [9:0] x;
194      logic [8:0] y;
195      logic [7:0] r, g, b;
196
197      audioControl dut(.clk, .rst, .en, .dataL, .dataR, .x, .y, .r, .g, .b);
198
199      parameter clock_period = 100;
200
201      initial begin
202         clk <= 0;
203         forever #(clock_period /2) clk <= ~clk;
204      end
205
206      initial begin
207         rst<=1; en <=0; dataL<=24'd500; dataR<=24'd600; x<=10'd200; y<=9'd50; @(posedge clk);
208         rst<=0; en <=1; dataL<=24'd8800; dataR<=24'd8800; x<=10'd50; y<=9'd100; @(posedge clk);
209         rst<=0; en <=1; dataL<=24'd8800; dataR<=24'd8800; x<=10'd30; y<=9'd100; @(posedge clk);
210         rst<=0; en <=1; dataL<=24'd2000; dataR<=24'd3800; x<=10'd500; y<=9'd250; @(posedge clk
      );
211         rst<=0; en <=1; dataL<=24'd2500; dataR<=24'd4800; x<=10'd400; y<=9'd350; @(posedge clk
      );
212         rst<=0; en <=1; dataL<=24'd5000; dataR<=24'd5800; x<=10'd300; y<=9'd50; @(posedge clk);
213         rst<=0; en <=1; dataL<=24'd6000; dataR<=24'd6800; x<=10'd200; y<=9'd70; @(posedge clk);
```



```
214         rst<=0; en <=1; dataL<=24'd7000; dataR<=24'd7800; x<=10'd100; y<=9'd150; @(posedge clk
      );
215         rst<=0; en <=1; dataL<=24'd5500; dataR<=24'd8800; x<=10'd150; y<=9'd130; @(posedge clk
      );
216         rst<=0; en <=1; dataL<=24'd6000; dataR<=24'd9800; x<=10'd250; y<=9'd120; @(posedge clk
      );
217         rst<=0; en <=1; dataL<=24'd700; dataR<=24'd8900; x<=10'd100; y<=9'd140; @(posedge clk);
218         rst<=0; en <=1; dataL<=24'd300; dataR<=24'd9800; x<=10'd500; y<=9'd400; @(posedge clk);
219         $stop;
220      end
221   endmodule
222
223
224
225
```

2) register.sv

```
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) register2.sv

```
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
```

4) ControlN.sv


```
1   //Khoa Tran and Ravi Sangani
2   //06/07/2020
3   //Lab 6, Task 1
4   //This module has inputs of clock, start, up, and down, which are 1 bit in size.
5   //The outputs are count, hex1, and hex0, which are 5 bits, 7 bits, and 7 bits in size
6   //respectively. This module outputs a decimal value on the hex display and this value
7   //can be incremented or decremented.
8   module controlN(clk, start, up, down, count, hex1, hex0);
9      input logic clk, start, up, down;
10     output logic [6:0] hex0, hex1;
11     output logic [4:0] count;
12
13     //This always_ff block handles the logic of incrementing and decrementing the counter
14     //based on the boolean inputs up and down.
15     always_ff @(posedge clk) begin
16        if (start)
17           count <= 5'b00000;
18        else if (up)
19           count <= count + 1;
20        else if (down)
21           count <= count - 1;
22        else
23           count <= count;
24     end
25
26     //storing the hex display decimal values into variables for code readibility
27     logic [6:0] zero, one, two, three, four, five, six, sevup, eight, nine;
28     assign zero = 7'b1000000;
29     assign one = 7'b1111001;
30     assign two = 7'b0100100;
31     assign three = 7'b0110000;
32     assign four = 7'b0011001;
33     assign five = 7'b0010010;
34     assign six = 7'b0000010;
35     assign sevup = 7'b1111000;
36     assign eight = 7'b0000000;
37     assign nine = 7'b0010000;
38
39
40     //combinational for cases of out values 0-31
41     //This always_comb block handles the logic of
42     //assigning the correct decimal display value to hex1 and hex0,
43     //based on the current value of count.
44     always_comb begin
45              case (count)
46                 5'b00000: begin
47                           hex1 = zero;
48                           hex0 = zero;
49                           end
50                 5'b00001: begin
51                           hex1 = zero;
52                           hex0 = one;
53                           end
54                 5'b00010: begin
55                           hex1 = zero;
56                           hex0 = two;
57                           end
58                 5'b00011: begin
59                           hex1 = zero;
60                           hex0 = three;
61                           end
62                 5'b00100: begin
63                           hex1 = zero;
64                           hex0 = four;
65                           end
66                 5'b00101: begin
67                           hex1 = zero;
68                           hex0 = five;
69                           end
70                 5'b00110: begin
71                           hex1 = zero;
72                           hex0 = six;
73                           end
74                 5'b00111: begin
75                           hex1 = zero;
76                           hex0 = sevup;
```

```
77                      end
78          5'b01000: begin
79                      hex1 = zero;
80                      hex0 = eight;
81                      end
82          5'b01001: begin
83                      hex1 = zero;
84                      hex0 = nine;
85                      end
86          5'b01010: begin
87                      hex1 = one;
88                      hex0 = zero;
89                      end
90          5'b01011: begin
91                      hex1 = one;
92                      hex0 = one;
93                      end
94          5'b01100: begin
95                      hex1 = one;
96                      hex0 = two;
97                      end
98          5'b01101: begin
99                      hex1 = one;
100                     hex0 = three;
101                     end
102         5'b01110: begin
103                     hex1 = one;
104                     hex0 = four;
105                     end
106         5'b01111: begin
107                     hex1 = one;
108                     hex0 = five;
109                     end
110         5'b10000: begin
111                     hex1 = one;
112                     hex0 = six;
113                     end
114         5'b10001: begin
115                     hex1 = one;
116                     hex0 = sevup;
117                     end
118         5'b10010: begin
119                     hex1 = one;
120                     hex0 = eight;
121                     end
122         5'b10011: begin
123                     hex1 = one;
124                     hex0 = nine;
125                     end
126         5'b10100: begin
127                     hex1 = two;
128                     hex0 = zero;
129                     end
130         5'b10101: begin
131                     hex1 = two;
132                     hex0 = one;
133                     end
134         5'b10110: begin
135                     hex1 = two;
136                     hex0 = two;
137                     end
138         5'b10111: begin
139                     hex1 = two;
140                     hex0 = three;
141                     end
142         5'b11000: begin
143                     hex1 = two;
144                     hex0 = four;
145                     end
146         5'b11001: begin
147                     hex1 = two;
148                     hex0 = five;
149                     end
150         5'b11010: begin
151                     hex1 = two;
152                     hex0 = six;
```

```
153                         end
154                 5'b11011: begin
155                         hex1 = two;
156                         hex0 = sevup;
157                         end
158                 5'b11100: begin
159                         hex1 = two;
160                         hex0 = eight;
161                         end
162                 5'b11101: begin
163                         hex1 = two;
164                         hex0 = nine;
165                         end
166                 5'b11110: begin
167                         hex1 = three;
168                         hex0 = zero;
169                         end
170                 5'b11111: begin
171                         hex1 = three;
172                         hex0 = one;
173                         end
174                 default: begin
175                         hex1 = 7'b1111111;
176                         hex0 = 7'b1111111;
177                         end
178                 endcase
179         end
180 endmodule
181
182 //This testbench ensures that the above module works with complete functionality.
183 //In specific, we test that the count value and hex output display responds as expected to
    the input booleans of
184 //start, up, and down.
185 module controlN_testbupch();
186    logic clk, start, up, down;
187    logic [6:0] hex0, hex1;
188    logic [4:0] count;
189
190    //device under test
191    controlN dut(.clk, .start, .up, .down, .count, .hex1, .hex0);
192
193    parameter clock_period = 100;
194
195    initial begin
196       clk <= 0;
197       forever #(clock_period /2) clk <= ~clk;
198    end
199
200    initial begin
201       start <= 1; up <= 0; down <= 0; @(posedge clk);
202       start <= 0; up <= 1; down <= 0; @(posedge clk);
203       start <= 0; up <= 0; down <= 0; @(posedge clk);
204       start <= 0; up <= 1; down <= 0; @(posedge clk);
205       start <= 0; up <= 1; down <= 0; @(posedge clk);
206       start <= 0; up <= 1; down <= 0; @(posedge clk);
207       start <= 0; up <= 1; down <= 0; @(posedge clk);
208       start <= 0; up <= 0; down <= 0; @(posedge clk);
209       start <= 0; up <= 0; down <= 0; @(posedge clk);
210       start <= 0; up <= 0; down <= 0; @(posedge clk);
211       start <= 0; up <= 0; down <= 0; @(posedge clk);
212       start <= 0; up <= 0; down <= 0; @(posedge clk);
213       start <= 0; up <= 0; down <= 0; @(posedge clk);
214       start <= 0; up <= 0; down <= 0; @(posedge clk);
215       start <= 0; up <= 1; down <= 0; @(posedge clk);
216       start <= 0; up <= 1; down <= 0; @(posedge clk);
217       start <= 0; up <= 1; down <= 0; @(posedge clk);
218       start <= 0; up <= 1; down <= 0; @(posedge clk);
219       start <= 0; up <= 1; down <= 0; @(posedge clk);
220       start <= 1; up <= 0; down <= 0; @(posedge clk);
221       start <= 0; up <= 0; down <= 1; @(posedge clk);
222       start <= 0; up <= 0; down <= 1; @(posedge clk);
223       start <= 0; up <= 0; down <= 1; @(posedge clk);
224       start <= 0; up <= 0; down <= 1; @(posedge clk);
225       start <= 0; up <= 0; down <= 1; @(posedge clk);
226       start <= 0; up <= 0; down <= 1; @(posedge clk);
227       start <= 0; up <= 0; down <= 1; @(posedge clk);
```

```
228          start <= 0; up <= 0; down <= 1; @(posedge clk);
229          start <= 0; up <= 0; down <= 1; @(posedge clk);
230          start <= 0; up <= 0; down <= 1; @(posedge clk);
231          start <= 0; up <= 0; down <= 1; @(posedge clk);
232          start <= 0; up <= 0; down <= 1; @(posedge clk);
233          start <= 0; up <= 1; down <= 0; @(posedge clk);
234          start <= 0; up <= 1; down <= 0; @(posedge clk);
235          start <= 0; up <= 1; down <= 0; @(posedge clk);
236          start <= 0; up <= 1; down <= 0; @(posedge clk);
237          start <= 0; up <= 1; down <= 0; @(posedge clk);
238          start <= 0; up <= 1; down <= 0; @(posedge clk);
239          start <= 0; up <= 1; down <= 0; @(posedge clk);
240          start <= 0; up <= 1; down <= 0; @(posedge clk);
241          $stop;
242       end
243    endmodule
244
```

5) NFirFilter.sv

Date: May 22, 2021 nFirFilter.sv Project: DE1_SoC

```
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
```

6) DE1_SoC.sv


```
1   //Khoa Tran and Ravi Sangani
2   //06/07/2020
3   //Lab 6, Task 1
4   //Module DE1_SoC uses drivers from audio_codec, n8 controller, and video driver for an
    audio visualizer
5   //on the VGA display. This audio visualizer can visualize audio with noise, or noise with a
    desired
6   //number of firFilter to minimize the noise, or the original audio itself. Using inputs of
    SW[1],
7   //the audio will be the original sound plus the noise. SW[0] will reset back to visualize
    the original
8   //audio and finally, KEY[1] and KEY[0] are used for up and down inputs for the user to
    decide on the
9   //number of firFilter to use to filter out the noise. The amount of firFilter goes from
    1-31, in order to
10  //see the differences between a large number of filters and a small number of filters.
    Overall,
11  //this module can visualize a series of different audios to compare sound, filtering, and
    originality.
12  module DE1_SoC #(parameter n = 16)(CLOCK_50, CLOCK2_50, FPGA_I2C_SCLK, FPGA_I2C_SDAT, KEY,
    SW,
13     AUD_XCK, AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT,
14     HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, LEDR,
15     VGA_R, VGA_G, VGA_B, VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS);
16
17     //input and output wires
18     input logic CLOCK_50, CLOCK2_50;
19     input logic [3:0] KEY;
20     input logic [9:0] SW;
21     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
22     output logic [9:0] LEDR;
23
24     // Video output
25     output [7:0] VGA_R;
26     output [7:0] VGA_G;
27     output [7:0] VGA_B;
28     output VGA_BLANK_N;
29     output VGA_CLK;
30     output VGA_HS;
31     output VGA_SYNC_N;
32     output VGA_VS;
33
34
35     logic startTemp, a, b;
36
37     // I2C Audio/Video config interface
38     output FPGA_I2C_SCLK;
39     inout FPGA_I2C_SDAT;
40     // Audio CODEC
41     output AUD_XCK;
42     input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
43     input AUD_ADCDAT;
44     output AUD_DACDAT;
45
46     //count wire
47     logic [4:0] countTemp;
48
49     // Local wires
50     logic read_ready, write_ready, read, write;
51     logic signed [23:0] readdata_left, readdata_right;
52     logic signed [23:0] writedata_left, writedata_right;
53     logic signed [23:0] task2_left, task2_right, task3_left, task3_right;
54     logic signed [23:0] left1, left2, left3, left4, left5, left6, left7, left8, left9, left10
    , left11, left12, left13, left14, left15, left16, left17, left18, left19, left20, left21,
    left22, left23, left24, left25, left26, left27, left28;
55     logic signed [23:0] right1, right2, right3, right4, right5, right6, right7, right8,
    right9, right10, right11, right12, right13, right14, right15, right16, right17, right18,
    right19, right20, right21, right22, right23, right24, right25, right26, right27, right28;
56     logic signed [23:0] noisy_left, noisy_right;
57     logic reset;
58     logic [23:0] noise, filtered1Left, filtered1Right, filtered2Left, filtered2Right;
59     logic [23:0] filtered3Left, filtered3Right, filtered4Left, filtered4Right, filtered5Left,
     filtered5Right, filtered6Left, filtered6Right, filtered8Left, filtered8Right, filtered9Left
    , filtered9Right, filtered10Left, filtered10Right;
60     logic [23:0] filtered11Left, filtered11Right, filtered12Left, filtered12Right,
    filtered13Left, filtered13Right, filtered14Left, filtered14Right, filtered15Left,
```

```
    filtered15Right, filtered16Left, filtered16Right;
61     logic [23:0] filtered17Left, filtered17Right, filtered18Left, filtered18Right,
    filtered19Left, filtered19Right, filtered20Left, filtered20Right, filtered21Left,
    filtered21Right, filtered22Left, filtered22Right;
62     logic [23:0] filtered23Left, filtered23Right, filtered24Left, filtered24Right,
    filtered25Left, filtered25Right, filtered26Left, filtered26Right, filtered27Left,
    filtered27Right, filtered28Left, filtered28Right,  filtered29Left, filtered29Right;
63     logic [23:0]  filtered30Left, filtered30Right,  filtered31Left, filtered31Right;
64
65     //instatiation of noise_gen to generate noise into readdata_left and readdata_right
66     noise_gen noise_generator (.clk(CLOCK_50), .en(read), .rst(reset), .out(noise));
67     assign noisy_left = readdata_left + noise;
68     assign noisy_right = readdata_right + noise;
69
70     //instatiation of nFirFilter for left and right channels with parameter n = 2
71     nFirFilter filter1L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
    filtered1Left));
72         defparam filter1L.n = 2;
73     nFirFilter filter1R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
    filtered1Right));
74         defparam filter1R.n = 2;
75     assign left1 = filtered1Left;
76     assign right1 = filtered1Right;
77
78     //instatiation of nFirFilter for left and right channels with parameter n = 3
79     nFirFilter filter2L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
    filtered2Left));
80         defparam filter2L.n = 3;
81     nFirFilter filter2R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
    filtered2Right));
82         defparam filter2R.n = 3;
83     assign left2 = filtered2Left;
84     assign right2 = filtered2Right;
85
86     //instatiation of nFirFilter for left and right channels with parameter n = 4
87     nFirFilter filter3L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
    filtered3Left));
88         defparam filter3L.n = 4;
89     nFirFilter filter3R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
    filtered3Right));
90         defparam filter3R.n = 4;
91     assign left3 = filtered3Left;
92     assign right3 = filtered3Right;
93
94     //instatiation of nFirFilter for left and right channels with parameter n = 5
95     nFirFilter filter4L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
    filtered4Left));
96         defparam filter4L.n = 5;
97     nFirFilter filter4R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
    filtered4Right));
98         defparam filter4R.n = 5;
99     assign left4 = filtered4Left;
100    assign right4 = filtered4Right;
101
102    //instatiation of nFirFilter for left and right channels with parameter n = 6
103    nFirFilter filter5L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
    filtered5Left));
104        defparam filter5L.n = 6;
105    nFirFilter filter5R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
    filtered5Right));
106        defparam filter5R.n = 6;
107    assign left5 = filtered5Left;
108    assign right5 = filtered5Right;
109
110    //instatiation of nFirFilter for left and right channels with parameter n = 7
111    nFirFilter filter6L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
    filtered6Left));
112        defparam filter6L.n = 7;
113    nFirFilter filter6R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
    filtered6Right));
114        defparam filter6R.n = 7;
115    assign left6 = filtered6Left;
116    assign right6 = filtered6Right;
117
118    //instatiation of nFirFilter for left and right channels with parameter n = 8
119    nFirFilter filter8L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
```

```
      filtered8Left));
120         defparam filter8L.n = 8;
121      nFirFilter filter8R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered8Right));
122         defparam filter8R.n = 8;
123      assign task2_left = filtered8Left;
124      assign task2_right = filtered8Right;
125
126      //instatiation of nFirFilter for left and right channels with parameter n = 9
127      nFirFilter filter9L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered9Left));
128         defparam filter9L.n = 9;
129      nFirFilter filter9R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered9Right));
130         defparam filter9R.n = 9;
131      assign left7 = filtered9Left;
132      assign right7 = filtered9Right;
133
134      //instatiation of nFirFilter for left and right channels with parameter n = 10
135      nFirFilter filter10L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered10Left));
136         defparam filter10L.n = 10;
137      nFirFilter filter10R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered10Right));
138         defparam filter10R.n = 10;
139      assign left8 = filtered10Left;
140      assign right8 = filtered10Right;
141
142      //instatiation of nFirFilter for left and right channels with parameter n = 11
143      nFirFilter filter11L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered11Left));
144         defparam filter11L.n = 11;
145      nFirFilter filter11R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered11Right));
146         defparam filter11R.n = 11;
147      assign left9 = filtered11Left;
148      assign right9 = filtered11Right;
149
150      //instatiation of nFirFilter for left and right channels with parameter n = 12
151      nFirFilter filter12L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered12Left));
152         defparam filter12L.n = 12;
153      nFirFilter filter12R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered12Right));
154         defparam filter12R.n = 12;
155      assign left10 = filtered12Left;
156      assign right10 = filtered12Right;
157
158      //instatiation of nFirFilter for left and right channels with parameter n = 13
159      nFirFilter filter13L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered13Left));
160         defparam filter13L.n = 13;
161      nFirFilter filter13R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered13Right));
162         defparam filter13R.n = 13;
163      assign left11 = filtered13Left;
164      assign right11 = filtered13Right;
165
166      //instatiation of nFirFilter for left and right channels with parameter n = 14
167      nFirFilter filter14L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered14Left));
168         defparam filter14L.n = 14;
169      nFirFilter filter14R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered14Right));
170         defparam filter14R.n = 14;
171      assign left12 = filtered14Left;
172      assign right12 = filtered14Right;
173
174      //instatiation of nFirFilter for left and right channels with parameter n = 15
175      nFirFilter filter15L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered15Left));
176         defparam filter15L.n = 15;
177      nFirFilter filter15R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered15Right));
178         defparam filter15R.n = 15;
179      assign left13 = filtered15Left;
```

```
180       assign right13 = filtered15Right;
181
182       //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
183       nFirFilter filter16L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered16Left));
184           defparam filter16L.n = 16;
185       nFirFilter filter16R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered16Right));
186           defparam filter16R.n = 16;
187       assign task3_left = filtered16Left;
188       assign task3_right = filtered16Right;
189
190       //instatiation of nFirFilter for left and right channels with parameter n = 17
191       nFirFilter filter17L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered17Left));
192           defparam filter17L.n = 17;
193       nFirFilter filter17R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered17Right));
194           defparam filter17R.n = 17;
195       assign left14 = filtered17Left;
196       assign right14 = filtered17Right;
197
198       //instatiation of nFirFilter for left and right channels with parameter n = 18
199       nFirFilter filter18L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered18Left));
200           defparam filter18L.n = 18;
201       nFirFilter filter18R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered18Right));
202           defparam filter18R.n = 18;
203       assign left15 = filtered18Left;
204       assign right15 = filtered18Right;
205
206       //instatiation of nFirFilter for left and right channels with parameter n = 19
207       nFirFilter filter19L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered19Left));
208           defparam filter19L.n = 19;
209       nFirFilter filter19R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered19Right));
210           defparam filter19R.n = 19;
211       assign left16 = filtered19Left;
212       assign right16 = filtered19Right;
213
214       //instatiation of nFirFilter for left and right channels with parameter n = 20
215       nFirFilter filter20L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered20Left));
216           defparam filter20L.n = 20;
217       nFirFilter filter20R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered20Right));
218           defparam filter20R.n = 20;
219       assign left17 = filtered20Left;
220       assign right17 = filtered20Right;
221
222       //instatiation of nFirFilter for left and right channels with parameter n = 21
223       nFirFilter filter21L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered21Left));
224           defparam filter21L.n = 21;
225       nFirFilter filter21R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered21Right));
226           defparam filter21R.n = 21;
227       assign left18 = filtered21Left;
228       assign right18 = filtered21Right;
229
230       //instatiation of nFirFilter for left and right channels with parameter n = 22
231       nFirFilter filter22L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered22Left));
232           defparam filter22L.n = 22;
233       nFirFilter filter22R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered22Right));
234           defparam filter22R.n = 22;
235       assign left19 = filtered22Left;
236       assign right19 = filtered22Right;
237
238       //instatiation of nFirFilter for left and right channels with parameter n = 23
239       nFirFilter filter23L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered23Left));
```

```
240         defparam filter23L.n = 23;
241      nFirFilter filter23R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
     filtered23Right));
242         defparam filter23R.n = 23;
243      assign left20 = filtered23Left;
244      assign right20 = filtered23Right;
245
246      //instatiation of nFirFilter for left and right channels with parameter n = 24
247      nFirFilter filter24L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
     filtered24Left));
248         defparam filter24L.n = 24;
249      nFirFilter filter24R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
     filtered24Right));
250         defparam filter24R.n = 24;
251      assign left21 = filtered24Left;
252      assign right21 = filtered24Right;
253
254      //instatiation of nFirFilter for left and right channels with parameter n = 25
255      nFirFilter filter25L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
     filtered25Left));
256         defparam filter25L.n = 25;
257      nFirFilter filter25R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
     filtered25Right));
258         defparam filter25R.n = 25;
259      assign left22 = filtered25Left;
260      assign right22 = filtered25Right;
261
262      //instatiation of nFirFilter for left and right channels with parameter n = 26
263      nFirFilter filter26L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
     filtered26Left));
264         defparam filter26L.n = 26;
265      nFirFilter filter26R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
     filtered26Right));
266         defparam filter26R.n = 26;
267      assign left23 = filtered26Left;
268      assign right23 = filtered26Right;
269
270      //instatiation of nFirFilter for left and right channels with parameter n = 27
271      nFirFilter filter27L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
     filtered27Left));
272         defparam filter27L.n = 27;
273      nFirFilter filter27R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
     filtered27Right));
274         defparam filter27R.n = 27;
275      assign left24 = filtered27Left;
276      assign right24 = filtered27Right;
277
278      //instatiation of nFirFilter for left and right channels with parameter n = 28
279      nFirFilter filter28L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
     filtered28Left));
280         defparam filter28L.n = 28;
281      nFirFilter filter28R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
     filtered28Right));
282         defparam filter28R.n = 28;
283      assign left25 = filtered28Left;
284      assign right25 = filtered28Right;
285
286      //instatiation of nFirFilter for left and right channels with parameter n = 29
287      nFirFilter filter29L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
     filtered29Left));
288         defparam filter29L.n = 29;
289      nFirFilter filter29R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
     filtered29Right));
290         defparam filter29R.n = 29;
291      assign left26 = filtered29Left;
292      assign right26 = filtered29Right;
293
294      //instatiation of nFirFilter for left and right channels with parameter n = 30
295      nFirFilter filter30L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
     filtered30Left));
296         defparam filter30L.n = 30;
297      nFirFilter filter30R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
     filtered30Right));
298         defparam filter30R.n = 30;
299      assign left27 = filtered30Left;
300      assign right27 = filtered30Right;
```

```
301
302       //instatiation of nFirFilter for left and right channels with parameter n = 31
303       nFirFilter filter31L(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_left), .result(
      filtered31Left));
304           defparam filter31L.n = 31;
305       nFirFilter filter31R(.clk(CLOCK_50), .rst(reset), .en(read), .data(noisy_right), .result(
      filtered31Right));
306           defparam filter31R.n = 31;
307       assign left28 = filtered31Left;
308       assign right28 = filtered31Right;
309
310       //combinational logic for SW[1] and countTemp from control to decide either
311       //writing of original audio data, original audio and noise, or original audio
312       //and noise filtered countTemp times.
313       always_comb begin
314          case({SW[1], countTemp})
315             6'b00000: begin
316                writedata_left = readdata_left;
317                writedata_right = readdata_right;
318             end
319             6'b00001: begin
320                writedata_left = readdata_left;
321                writedata_right = readdata_right;
322             end
323             6'd2: begin
324                writedata_left = left1;
325                writedata_right = right1;
326             end
327             6'd3: begin
328                writedata_left = left2;
329                writedata_right = right2;
330             end
331             6'd4: begin
332                writedata_left = left3;
333                writedata_right = right3;
334             end
335             6'd5: begin
336                writedata_left = left4;
337                writedata_right = right4;
338             end
339             6'd6: begin
340                writedata_left = left5;
341                writedata_right = right5;
342             end
343             6'd7: begin
344                writedata_left = left6;
345                writedata_right = right6;
346             end
347             6'd8: begin
348                writedata_left = task2_left;
349                writedata_right = task2_right;
350             end
351             6'd9: begin
352                writedata_left = left7;
353                writedata_right = right7;
354             end
355             6'd10: begin
356                writedata_left = left8;
357                writedata_right = right8;
358             end
359             6'd11: begin
360                writedata_left = left9;
361                writedata_right = right9;
362             end
363             6'd12: begin
364                writedata_left = left10;
365                writedata_right = right10;
366             end
367             6'd13: begin
368                writedata_left = left11;
369                writedata_right = right11;
370             end
371             6'd14: begin
372                writedata_left = left12;
373                writedata_right = right12;
374             end
```

```
375             6'd15: begin
376                writedata_left = left13;
377                writedata_right = right13;
378             end
379             6'd16: begin
380                writedata_left = task3_left;
381                writedata_right = task3_right;
382             end
383             6'd17: begin
384                writedata_left = left14;
385                writedata_right = right14;
386             end
387             6'd18: begin
388                writedata_left = left15;
389                writedata_right = right15;
390             end
391             6'd19: begin
392                writedata_left = left16;
393                writedata_right = right16;
394             end
395             6'd20: begin
396                writedata_left = left17;
397                writedata_right = right17;
398             end
399             6'd21: begin
400                writedata_left = left18;
401                writedata_right = right18;
402             end
403             6'd22: begin
404                writedata_left = left19;
405                writedata_right = right19;
406             end
407             6'd23: begin
408                writedata_left = left20;
409                writedata_right = right20;
410             end
411             6'd24: begin
412                writedata_left = left21;
413                writedata_right = right21;
414             end
415             6'd25: begin
416                writedata_left = left22;
417                writedata_right = right22;
418             end
419             6'd26: begin
420                writedata_left = left23;
421                writedata_right = right23;
422             end
423             6'd27: begin
424                writedata_left = left24;
425                writedata_right = right24;
426             end
427             6'd28: begin
428                writedata_left = left25;
429                writedata_right = right25;
430             end
431             6'd29: begin
432                writedata_left = left26;
433                writedata_right = right26;
434             end
435             6'd30: begin
436                writedata_left = left27;
437                writedata_right = right27;
438             end
439             6'd31: begin
440                writedata_left = left28;
441                writedata_right = right28;
442             end
443             default: begin // default output noisy data
444                writedata_left = noisy_left;
445                writedata_right = noisy_right;
446             end
447          endcase
448       end
449
450       //reset to KEY[3]
```

```
451       assign reset = ~KEY[3];
452
453       // only read or write when both are possible
454       assign read = read_ready & write_ready;
455       assign write = read_ready & write_ready;
456
457
458     ////////////////////////////////////////////////////////////////////////////
459     // Audio CODEC interface.
460     //
461     // The interface consists of the following wires:
462     // read_ready, write_ready - CODEC ready for read/write operation
463     // readdata_left, readdata_right - left and right channel data from the CODEC
464     // read - send data from the CODEC (both channels)
465     // writedata_left, writedata_right - left and right channel data to the CODEC
466     // write - send data to the CODEC (both channels)
467     // AUD_* - should connect to top-level entity I/O of the same name.
468     //         These signals go directly to the Audio CODEC
469     // I2C_* - should connect to top-level entity I/O of the same name.
470     //         These signals go directly to the Audio/Video Config module
471     ////////////////////////////////////////////////////////////////////////////
472       clock_generator my_clock_gen(
473          // inputs
474          CLOCK2_50,
475          1'b0,
476
477          // outputs
478          AUD_XCK
479       );
480
481       audio_and_video_config  cfg(
482          // Inputs
483          CLOCK_50,
484          1'b0,
485
486          // Bidirectionals
487          FPGA_I2C_SDAT,
488          FPGA_I2C_SCLK
489       );
490
491       audio_codec codec(
492          // Inputs
493          CLOCK_50,
494          1'b0,
495
496          read, write,
497          writedata_left, writedata_right,
498
499          AUD_ADCDAT,
500
501          // Bidirectionals
502          AUD_BCLK,
503          AUD_ADCLRCK,
504          AUD_DACLRCK,
505
506          // Outputs
507          read_ready, write_ready,
508          readdata_left, readdata_right,
509          AUD_DACDAT
510       );
511
512       //N8 driver
513        n8_driver driver(
514            .clk(CLOCK_50),
515            .data_in(N8_DATA_IN),
516            .latch(N8_LATCH),
517            .pulse(N8_PULSE),
518            .up(),
519            .down(),
520            .left(),
521            .right(),
522            .select(),
523            .start(startTemp),
524            .a(a),
525            .b(b)
526         );
```

```
527
528       //local wires
529       logic [9:0] x;
530       logic [8:0] y;
531       logic [7:0] r, g, bColor;
532       logic [6:0] hex1Temp, hex0Temp;
533
534       //video_driver of VGA 640x480, outputing x and y and inputs rgb for color at the
      coordinate outputted
535       video_driver #(.WIDTH(640), .HEIGHT(480))
536          v1 (.CLOCK_50, .reset(1'b0), .x, .y, .r, .g, .b(bColor),
537              .VGA_R, .VGA_G, .VGA_B, .VGA_BLANK_N,
538              .VGA_CLK, .VGA_HS, .VGA_SYNC_N, .VGA_VS);
539
540       //instatiation of audioControl, using audio data chosen, and x and y coordinate from
      video driver, output rgb based on the values of
541       //left and right channels in 16 rectanges with varying width and height based on audio
      input
542       audioControl audio1(.clk(CLOCK_50), .rst(reset), .en(read), .dataL(writedata_left), .
      dataR(writedata_right), .x, .y, .r, .g, .b(bColor));
543
544       //instatiation of controlN allowing SW[0] to reset the count back to 0 and KEY1 and KEY0
      to increment count that is displayed on HEX1 and HEX0
545       controlN nVal(.clk(CLOCK_50), .start(SW[0]), .up(~KEY[1]), .down(~KEY[0]), .count(
      countTemp), .hex1(HEX1), .hex0(HEX0));
546
547       //make HEX2-5 blank
548       assign HEX2 = 7'b1111111;
549       assign HEX3 = 7'b1111111;
550       assign HEX4 = 7'b1111111;
551       assign HEX5 = 7'b1111111;
552
553
554    endmodule
555
556
```