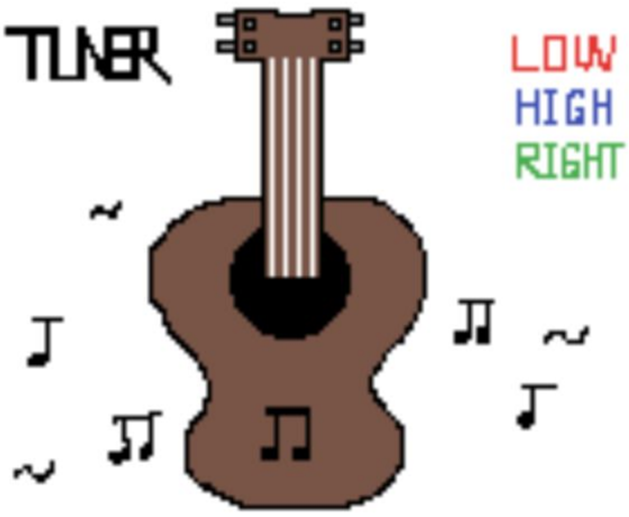
ECE241 Final Project

Ukulele Tuner and Player



Yi Jia Zhang, Zetong Zhao

December 2, 2019

1 Introduction (Overview)

This project's objective is to implement a *pitch tuner* as well as a *tone player* for Ukulele using FPGA board (DE1-SoC). This topic is brought up due to the team members' common interest in signal processing. The final product allows the user to select between the two major features by using a switch on the FPGA board.

In the *tuner* mode, the users need to first input a note number that they want to tune using the switches on the board. After the above action has been made, the system takes in a note played by the ukulele through the mic-input port on De1-SoC and feed the signal into a module called Audio Controller, which would then converts the analog input signal into digital. Once the audio sample is ready, it is passed in to the module call "Fast Fourier Transform" (FFT) in order to transform the time domain sample into data in frequency domain. Then, the next module is responsible for detecting peaks in the signal. The selected peaks are then being sent to the tone check module to determine whether the note played by the user is higher or lower than the desired tone. After the comparison is completed, the output would be shown on the screen display.

The *tone player* mode is controlled by a mouse that is connected to the PS2 port on the FPGA, the user can use the cursor to click on a string that is drawn on the VGA display and hear the corresponding note played through the speaker, which is connected to the line-out port on the board. The system first detects the user's input by determining the location of clicks that are sent by the mouse. The positions are then being converted into the associated note number, this information is sent to the next module for generating the wanted frequency as digital signal. The signal is being fed to the Audio Controller which converts the digital into analog signal, then passed the converted data to the line-out port on the DE1-SoC.

The detailed function of each individual module is discussed in the following sections of this report.

2 The Design (Module Specification)

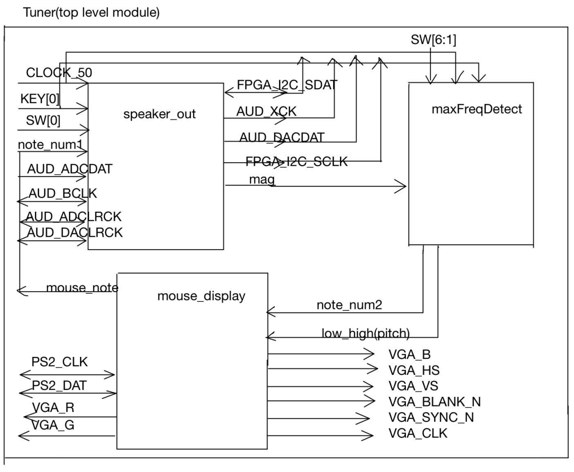


Figure 1 Top Level View

As shown in Figure 1, the Ukulele tuner system contains three major components: speaker_out, maxFreqDetect, and mouse_display. The speaker_out module is in charge of getting the user's input and converts the signal into an analysiable form and output selected frequency, it comprised three sub-parts: the Audio Controller, filter, and playtone. The maxFreqDetect module is responsible for analysing the transformed data received from the previous module (speaker_out), maxFreqDetetc consists of three instances: max_finder, maxFreq, and tonecheck. Lastly, the mouse_display module is the VGA display in this project, it has four sub-parts: transfer, datapath, control, VGA_controller, which shows the mouse's cursor as well as the project display on the screen.

2.1 speaker_out (Block diagram in Appendix A Figure 2)

2.1.1 audio controller

The audio controller module used in this project is provided online by the University of Toronto[Reference 3]. This module uses the Audio CoDec in the FPGA, it receives the analog signals input from the mic-in port, and converts it into digital signals that can be processed by computers. For this project, both the analog to digital, and digital to analog functions are needed. Therefore, an 2to1 selector is added to the module. This selector is controlled by *switch0* on the FPGA board for swapping between the two modes of the system: 1 for *tuner* and 0 for *tone player.*

2.1.2 Filter

The filter module is used when the mode selector is put to *tuner* mode, it consists the FFT IP core provided by Altera [B.1.2]. This IP core is used to transform the time domain signal passed in by audio controller into data in frequency domain. The output of FFT have a real stream as well as an imaginary stream. The filter module then process the two sets of data to find the squared magnitude ($magnitude^2 = real^2 + imag^2$) and send the calculated magnitude to the next module named maxFreqDetect.

2.1.3 playtone

The playtone module is used when the user selects *tone player* mode. This module takes in the user input through mouse_display (section 2.3). The transfer instance (section 2.3.1) in mouse_display would send back the note number selected by the user. Playtone takes this note number and generates the corresponding frequency. The data are then being streamed back to the audio controller to convert into analog signal. The converted signal is sent out from the line-out port on the FPGA. The desired musical note would be heard from the speaker that is connected to the line-out port.

2.2 maxFreqDetect (Block diagram in Appendix A Figure 3)

2.2.2 max_finder and maxFreq

As the name suggests, these module is used to detect the peak frequency according to the output from fourier transformation. Since the FFT will output N (N is chosen to be 1024 in this project) points that are evenly distributed between the sampling frequency (Fs = 48kHz) range, the interval between each sampling point is Fs/N. The maximum magnitude is found linearly, and using the index of the point that has the largest magnitude (I), multiply it by the sampling interval ($maxFreq = I_{max} * (Fs/N)$), the frequency is then being acquired.

2.2.3 Tonecheck

After getting the frequency of the sample point that has the maximum magnitude, the associated frequency is then being fed to the tonecheck module. This module first takes in the note number that the user wants to compare to, and then compares the detected frequency with the correct frequency that are pre-entered in the system according to each note's corresponding frequency. The note considered in this project are the four notes on the ukulele, which are A2 (#22), C3 (#25), E3 (#29), and G3 (#32). The Yamaha's convention of C3 as Middle C and the Rodgers convention for note number #34 as the 440Hz standard A are being used in this project [C.1].

Since the tonecheck module only finds the note for a single frequency input, the module is instantiated several times according to the number of peaks found.

The associated musical note's frequency values are in Appendix C table 1.

2.3 mouse_display (Block diagram in Appendix A Figure 4)

In order to adapt the mouse cursor display, and decrease the flickering caused by the waiting time between erasing the cursor and drawing the background. RAM is used to store the background for easy pixel access to the mif file.

2.3.1 transfer

The transfer module detects the position output of the mouse that is connected to the PS2 port on the FPGA using a sub module called ps2_mouse, which are provided by the University of Toronto Computer Engineering department [Reference 1]. Since the mouse output would only gives its relative position and a signal for clicks, the transfer module is responsible for turning the relative position into the mouse's current position and send out a binary signal for left-clicked or not. Aside from determining the location and clicking status, the transfer module also contains a look-up table that would map the location of the mouse to the corresponding string on the screen as well as the note number associated with that string. The note number, clicking status, and current position are then being outputted by this module.

2.3.1 datapath and control

The datapath and control are the module that serves as the finite state machine for the VGA display. It would display the string as green, once the transfer module send back a mouse clicked signal to the datapath. A state diagram used for this project is being provided in Appendix C table 2.

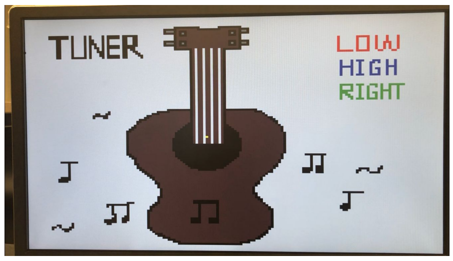
2.3.3 VGA_controller

The VGA_controller module is being provided by the University of Toronto Computer Engineering department. It uses the VGA ports on DE1-SoC.

3 Report on Success

3.1 Success

The *tone player* mode in this project is successful, the following figures (5 and 6) shows that the mouse click, and VGA display are all correct and working nicely. The use of RAM in the mouse_display solved the flickering problem and generates a stable performance for a moving mouse cursor. After a string is being clicked the speaker outputs the note correctly until the user release the left-click, and when the mouse is not clicking the speaker would remain silent.



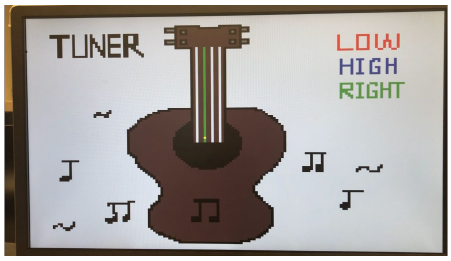
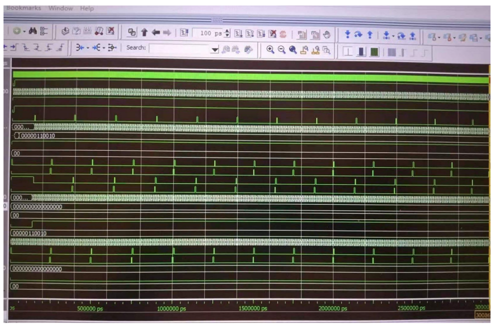


Figure 5 VGA display

Figure 6 mouse clicked display

The *tuner* mode is also partially successful. The simulation of the filter and the maxFreqDetect are shown in Figure 7 and 8 respectively. The simulations shows that both modules are doing what they are intended to achieve.



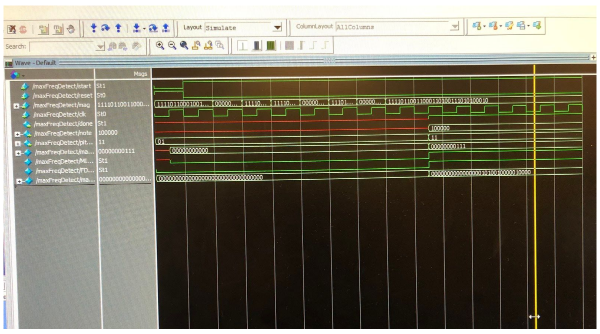


Figure 7 Filter simulation

Figure 8 maxFreqDetect simulation

3.2 Failure

Even though the simulation works fine, in real life situation, there are two major problems encountered that caused a failure in detecting the correct notes. First, the output from FFT are over saturated which cause the maxFreqDetect module unable to detect the maximum frequency. During the testing process of this module 10 LED lights are being assigned to the FFT output data, it has been discovered that even the most significant bits of the FFT output are lighting up dimly. This suggested the possibility that the FFT is generating highly saturated data. Secondly, the input noise are louder than expected. Similarly to the first speculation, the loud input noise would cause the sound played by the instrument hard to detect. According to inspections, the presence of a high frequency noise is the reason why the VGA display always notify the user that the note they played is too high.

Next Steps (what can be done differently)

If more time is allowed for this project, finding a decent way to filter out the noise is definitely the first step to be done. If a chance to start all over the project is given, one of the big changes is to raise awareness of the connection between each module. Since this project requires a real life sound wave input, the completion of the audio input modules is the key for testing the following module. For this project, since lots of time is being wasted on audio controller, even though the maxFreqDetect module has long been finished, it is being tested at the end, which caused time shortage for figure out ways for noise filtering. Another improvement would be to make the mouse cursor more artistic and more identifiable, instead of using a single square pixel.

Appendix A: Schematic

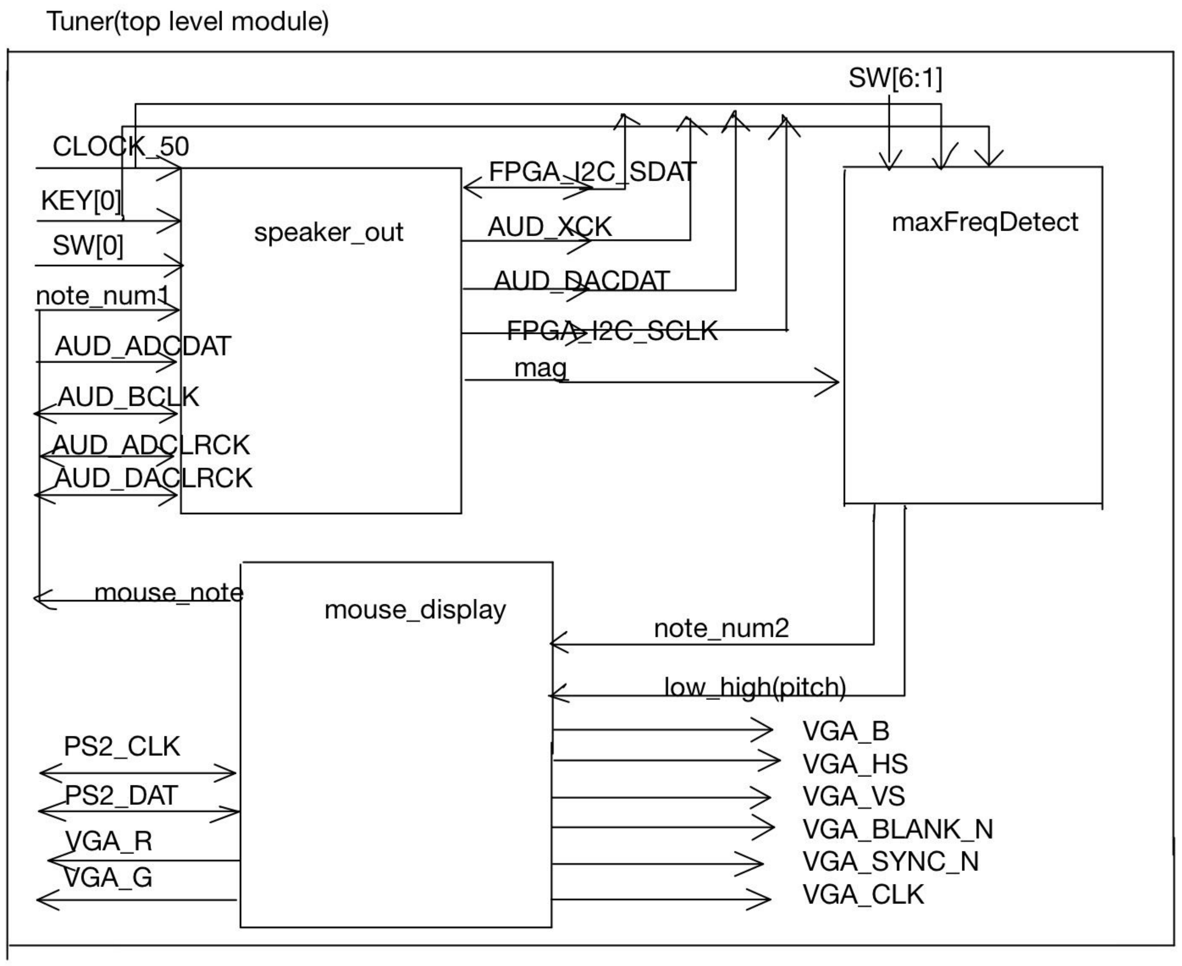


Figure 1 Top Level module

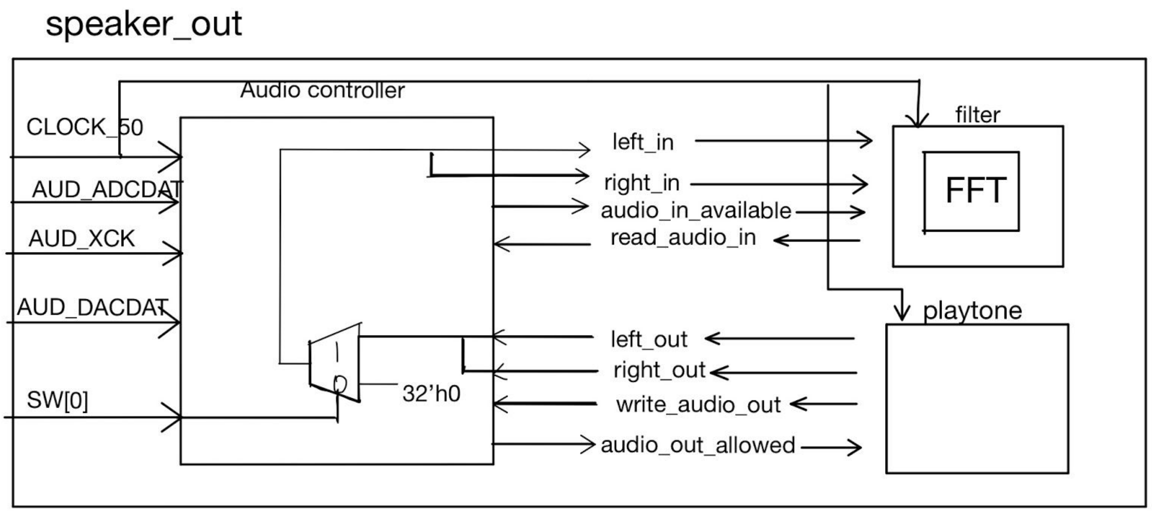


Figure 2 speaker_out module

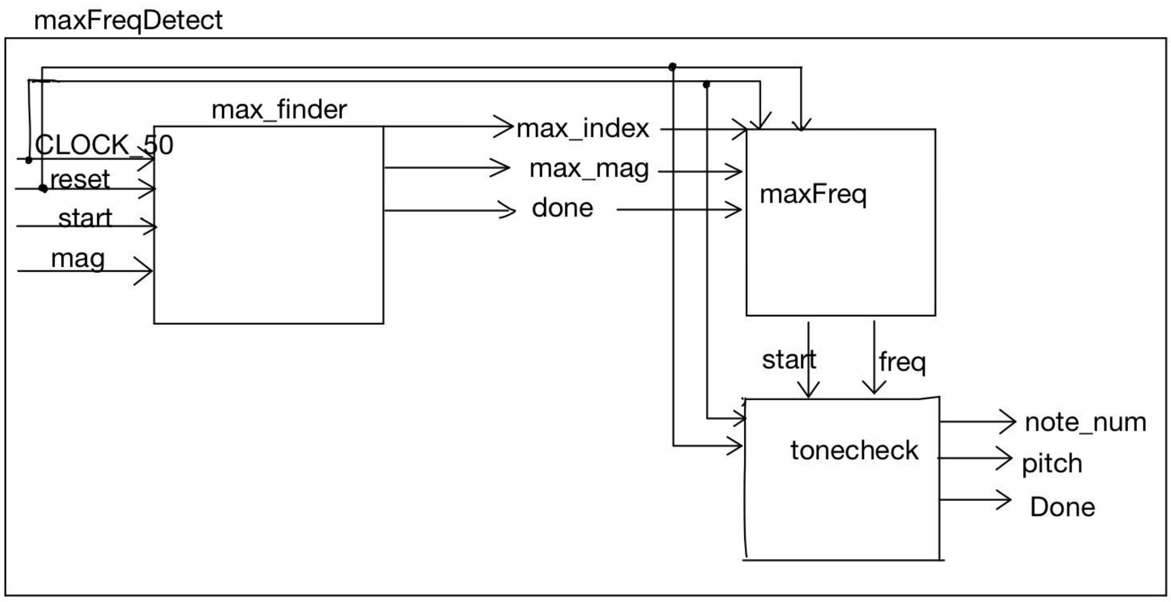


Figure 3 maxFreqDetect module

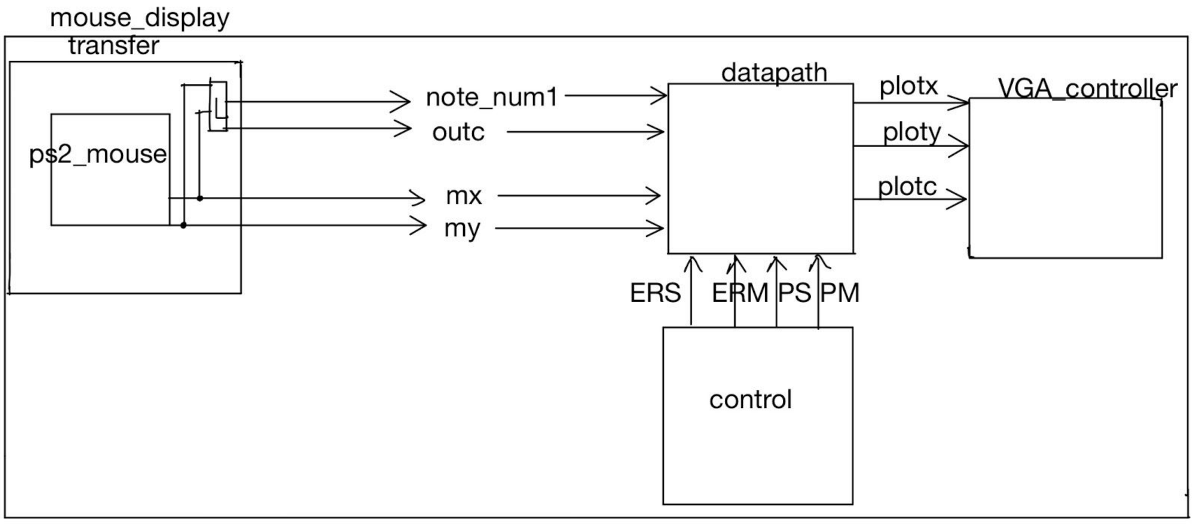


Figure 4 mouse_display module

Appendix B: Verilog Code

B.0 Top Level(Tuner)

```
module Tuner(CLOCK_50, VGA_R,
                              VGA_G,
                              VGA_B,
                              VGA_HS,
                              VGA_VS,
                              VGA_BLANK_N,
                              VGA_SYNC_N,
                              VGA_CLK,
                              KEY, SW,// control playout/mic_in
//input/output from audio controller
        // Inputs

        AUD_ADCDAT,

        // Bidirectionals
        AUD_BCLK,
        AUD_ADCLRCK,
        AUD_DACLRCK,

        FPGA_I2C_SDAT,

        // Outputs
        AUD_XCK,
        AUD_DACDAT,

        FPGA_I2C_SCLK,
PS2_CLK,
PS2_DAT,
HEX0, HEX1, HEX2, HEX3, LEDR, HEX4, HEX5);

input CLOCK_50;
output [7:0] VGA_R;
output [7:0] VGA_G;
output [7:0] VGA_B;
output VGA_HS;
output VGA_VS;
output VGA_BLANK_N;
output VGA_SYNC_N;
output VGA_CLK;

input [2:0]KEY;
input [9:0]SW;

input    AUD_ADCDAT;
inout    AUD_BCLK;
inout    AUD_ADCLRCK;
inout    AUD_DACLRCK;
inout    FPGA_I2C_SDAT;
inout    PS2_CLK;
inout    PS2_DAT;
output AUD_XCK;
output AUD_DACDAT;
output FPGA_I2C_SCLK;

output [6:0]HEX0;
output [6:0]HEX1;
output [6:0]HEX2;
output [6:0]HEX3;
output [6:0]HEX4;
output [6:0]HEX5;
output [9:0]LEDR;


//wire to connect mic_in & maxFreqDetector
wire start;//from maxf to mic_in
wire [63:0]mag;//from mic_in to maxf
//wire [10:0]index;//from mic_in to maxf
wire done;

//wire to connect maxFreq & VGA display
//wire [5:0]note_num;//final diaplay note_num
```

```
	wire [5:0]note_num1;//from ps2 input
	wire [5:0]note_num2;//from freq detect
	wire [1:0]lowhigh;

	//wire to connect mouse and display
	/*wire click;
	wire [7:0]mx;
	wire [6:0]my;
	wire [7:0]mx1;
	wire [6:0]my1;
	wire outc;*/
	//assign LEDR[1:0] = lowhigh[1:0];
	//hex_decoder H0(.hex_digit(note_num2[3:0]), .segments(HEX0[6:0]));
	//hex_decoder H1(.hex_digit(note_num2[5:4]), .segments(HEX1[6:0]));

	reg [27:0]counter =28'b0;
	reg [1:0]pitch;
	always @(posedge CLOCK_50) begin
		if (counter == 0) begin
			if (done) begin
				pitch <= lowhigh;
				counter <= counter+1;
				end
			else begin
				counter <= 0;
				//pitch <= pitch;
				end
			end
		else if (counter == 8333334) begin
			counter <= 0;
			//pitch <= pitch;
			end
		else begin
			counter <= counter+ 1;
			//pitch <= pitch;
			end
	end


//	assign LEDR[0] = done;
	//assign LEDR[7:6] = lowhigh[1:0];
	//assign LEDR[9:8] = pitch[1:0];
	speaker_out S0(.CLOCK_50(CLOCK_50), .KEY(KEY[0]), .SW(SW[0]),
.note_num1(note_num1),
	.AUD_ADCDAT(AUD_ADCDAT),
	.AUD_BCLK(AUD_BCLK),	.AUD_ADCLRCK(AUD_ADCLRCK),
.AUD_DACLRCK(AUD_DACLRCK),
	.FPGA_I2C_SDAT(FPGA_I2C_SDAT), .AUD_XCK(AUD_XCK),
.AUD_DACDAT(AUD_DACDAT),
	.FPGA_I2C_SCLK(FPGA_I2C_SCLK), .start(start), .done(1'b1), .mag(mag),
.LEDR(LEDR[7:0]));//, .LEDR(LEDR[9:2]));

	maxFreqDetect m1(.start(start), .reset(~KEY[0]), .clk(CLOCK_50),
	.mag(mag), .note(note_num2), .done(done), .pitch_indicator(lowhigh),
	.expected(SW[6:1]));//.HEX4(HEX4[6:0]), .HEX5(HEX5[6:0]));//, .LEDR(LEDR[8:0]));//,
.HEX0(HEX0[6:0]), .HEX1(HEX1[6:0]), .HEX2(HEX2[6:0]), .HEX3(HEX3[6:0]),
	//.HEX4(HEX4[6:0]), .HEX5(HEX5[6:0]));


	mouse_display m2(.KEY(KEY[0]), .PS2_CLK(PS2_CLK), .PS2_DAT(PS2_DAT),
				.HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3),
.note_num2(note_num2),
				.lowhigh(pitch), .CLOCK_50(CLOCK_50), .VGA_R(VGA_R),
.VGA_G(VGA_G),
				.VGA_B(VGA_B),	.VGA_HS(VGA_HS), .VGA_VS(VGA_VS),
.VGA_BLANK_N(VGA_BLANK_N),
				.VGA_SYNC_N(VGA_SYNC_N),
				.VGA_CLK(VGA_CLK),
				.mouse_note(note_num1), .SW(SW[0]));


endmodule
```

B.1 Audio (speaker_out/ Filter/FFT/playtone)

B.1.0 speaker_out

```
module speaker_out (
	// Inputs
	CLOCK_50,
	KEY,
	SW, note_num1,

	AUD_ADCDAT,

	// Bidirectionals
	AUD_BCLK,
	AUD_ADCLRCK,
	AUD_DACLRCK,
        FPGA_I2C_SDAT,

        // Outputs
        AUD_XCK,
        AUD_DACDAT,

        FPGA_I2C_SCLK,
        LEDR,
        start,mag,done

);

/*****************************************************************************
 *                    Parameter Declarations                    *
 *****************************************************************************/


/*****************************************************************************
 *                    Port Declarations                    *
 *****************************************************************************/
// Inputs
input                       CLOCK_50;
input           [0:0]   KEY;
input           [0:0]   SW;
input           [5:0]note_num1;

input                       AUD_ADCDAT;

// Bidirectionals
inout                       AUD_BCLK;
inout                       AUD_ADCLRCK;
inout                       AUD_DACLRCK;

inout                       FPGA_I2C_SDAT;

// Outputs
output                      AUD_XCK;
output                      AUD_DACDAT;

output                      FPGA_I2C_SCLK;


output [7:0]LEDR;
output start;
output [63:0]mag;
wire [10:0]index;
```

```
input done;



/*****************************************************************************
 *                Internal Wires and Registers Declarations                 *
 *****************************************************************************/
// Internal Wires
wire                    audio_in_available;
wire        [31:0]  left_channel_audio_in;
wire        [31:0]  right_channel_audio_in;
wire                    read_audio_in;

wire                    audio_out_allowed;
wire        [31:0]  left_channel_audio_out;
wire        [31:0]  right_channel_audio_out;
wire                    write_audio_out;



// Internal Registers

reg [18:0] delay_cnt;
wire [18:0] delay;

reg snd;

// State Machine Registers

/*****************************************************************************
 *                    Finite State Machine(s)                    *
 *****************************************************************************/


/*****************************************************************************
 *                        Sequential Logic                    *

 *****************************************************************************/

always @(posedge CLOCK_50)
	if(delay_cnt == delay) begin
		delay_cnt <= 0;
		snd <= !snd;
	end else delay_cnt <= delay_cnt + 1;

/*****************************************************************************
```

```
 *                       Combinational Logic                        *
 *****************************************************************************/

assign delay = {4'b0000, 15'd3000};

wire [31:0] sound = SW[0] ? 0 : mag_out;


/*****************************************/
wire [31:0]mag_out;
//assign LEDR[9:0] = left_channel_audio_in[31:22];
//assign LEDR[9:0] = mag_out[31:22];
//assign LEDR[0] = audio_in_available;
playtone  P0(.CLOCK_50(CLOCK_50), .reset(~KEY[0]), .note_num(note_num1),
.ready_read(1'b1), .audio_out(mag_out[31:0]), .ready_out());
/*****************************************/
assign read_audio_in                    = audio_in_available & audio_out_allowed;

assign left_channel_audio_out   = left_channel_audio_in+sound;
assign right_channel_audio_out = right_channel_audio_in+sound;
assign write_audio_out                  = audio_in_available & audio_out_allowed;

/*****************************************************************************
 *                        Internal Modules                        *
 *****************************************************************************/

Audio_Controller Audio_Controller (
        // Inputs
        .CLOCK_50                                       (CLOCK_50),
        .reset                                  (~KEY[0]),

        .clear_audio_in_memory          (),
        .read_audio_in                  (read_audio_in),

        .clear_audio_out_memory         (),
        .left_channel_audio_out         (left_channel_audio_out),
        .right_channel_audio_out        (right_channel_audio_out),
        .write_audio_out                (write_audio_out),

        .AUD_ADCDAT                                     (AUD_ADCDAT),

        // Bidirectionals
        .AUD_BCLK                               (AUD_BCLK),
        .AUD_ADCLRCK                            (AUD_ADCLRCK),
        .AUD_DACLRCK                            (AUD_DACLRCK),
	// Outputs
	.audio_in_available			(audio_in_available),
	.left_channel_audio_in		(left_channel_audio_in),
	.right_channel_audio_in		(right_channel_audio_in),

	.audio_out_allowed			(audio_out_allowed),

	.AUD_XCK					(AUD_XCK),
	.AUD_DACDAT						(AUD_DACDAT)

);

avconf #(.USE_MIC_INPUT(1)) avc (
	.FPGA_I2C_SCLK					(FPGA_I2C_SCLK),
	.FPGA_I2C_SDAT					(FPGA_I2C_SDAT),
	.CLOCK_50					(CLOCK_50),
	.reset						(~KEY[0])
);

/*output start;
output [63:0]mag;
wire [10:0]index;
output done;*/
/*wire [31:0]LEFT;
assign LEFT[31:0] = left_channel_audio_in[31:0] >>> 3;*/


//assign mag[31:0]= mag_out[31:0];

	Filter F0(.CLOCK_50(CLOCK_50),
	.reset_n(~KEY[0]),
				.write_in(left_channel_audio_in),
				.ready_write(1'b1),
				.write_out(),
				.start(start),
				.mag(mag),
				.index(index),
				.done(1'b1), .LEDR(LEDR[7:0]));



endmodule
```

B.1.1 Filter

```
module Filter(CLOCK_50,
        reset_n,
                                        write_in,
                                        ready_write,
                                        write_out,
                                        start,
                                        mag,
                                        index,
                                        done, LEDR);

        output [7:0]LEDR;
        assign LEDR[7:0] = mag[63:56];

        input CLOCK_50;
  input reset_n;
        input [31:0]write_in;
        input ready_write;
        output write_out;
        //from upstream module

        output reg start;
        output [63:0]mag;
        reg [63:0]mag1;
        reg [63:0]mag2;
        output [10:0]index;
        input done;
        //to downstream module

        //wires connected to fft
        reg [1:0] sink_error = 2'b0;
        reg sink_sop = 1'b0;
        reg sink_eop = 1'b0;
        wire source_sop;
        wire source_eop;
        wire signed [31:0]source_real;
        wire signed [31:0]source_imag;
        wire [1:0] source_error;
        wire source_valid;
        //wire sink_ready = 1'b1;

        wire [10:0]fftpts_in = 11'b01000000000;
        reg [10:0]counter = 11'b0;
```

```
wire sop;
wire eop;
wire [31:0]sink_imag = 32'b0;
//wire inverse = 1'b0;
wire [1:0]error;
//counter to set sop and eop
always @(posedge CLOCK_50) begin
        if (reset_n) begin
                sink_sop <= 1'b0;
                sink_eop <= 1'b0;
                counter <= 11'b0;
                end
        else begin
                if (counter == 0) begin
                        sink_sop <=1'b1;
                        sink_eop <=1'b0;
                        counter <= counter +1;
                        end
                else if (counter == 1) begin
                        sink_sop <=1'b0;
                        counter <= counter +1;
                        end
                else if (counter == 1023) begin
                        sink_eop <=1'b1;
                        counter <= 0;
                        end
                else begin
                        sink_sop <=1'b0;
                        sink_eop <=1'b0;
                        counter <= counter +1;
                        end
                end
end



//set values for sink_error
always @(posedge CLOCK_50) begin
        if (reset_n) begin
                sink_error <= 2'b00;
                end
        else begin
                if ((!sink_sop) && (counter == 1)) begin
                        sink_error <= 2'b01;
                        end
        /*else if ((counter == 1023) && (!sink_eop)) begin
```

```
                    sink_error <= 2'b10;
                    end*/
          /*else if (write_in <= 10) begin //too small or large magnitude of frequency
                    sink_error <= 2'b11;
                    end*/
                    else begin
                              sink_error <= 2'b00;
                              end
                    end
end


assign sop = sink_sop;
assign eop = sink_eop;
assign error[1:0] = sink_error[1:0];
//call fft
FFT u0 (
          .clk          (CLOCK_50),          //    clk.clk
          .reset_n      (~reset_n),      //    rst.reset_n
          .sink_valid   (ready_write),   //   sink.sink_valid
          .sink_ready   (write_out),   //       .sink_readywrite_in <= 10
          .sink_error   (error),   //       .sink_error
          .sink_sop     (sop),     //       .sink_sop
          .sink_eop     (eop),     //       .sink_eop
          .sink_real    (write_in),    //       .sink_real
          .sink_imag    (sink_imag),    //       .sink_imag
          .fftpts_in    (fftpts_in),    //       .fftpts_in
          .inverse      (1'b0),      //       .inverse
          .source_valid (source_valid), // source.source_valid
          .source_ready (done), //       .source_ready
          .source_error (source_error), //       .source_error
          .source_sop   (source_sop),   //       .source_sop
          .source_eop   (source_eop),   //       .source_eop
          .source_real  (source_real),  //       .source_real
          .source_imag  (source_imag),  //       .source_imag
          .fftpts_out   (index)   //       .fftpts_out
);



//check if the filter is ready to start (output to next module)
always @(posedge CLOCK_50)begin
if ((source_sop == 1'b1) && (source_error == 2'b00) && (source_valid)) begin
          start <= 1'b1;
          end
else if ((source_eop == 1'b1) && (source_error == 2'b00) && (source_valid)) begin
```

```
                start <= 1'b0;
                end
        else begin
                start <= 1'b1;
                end
        end

        //find the magnitude of each output number
        always @(posedge CLOCK_50)begin
                mag1 <= (source_real * source_real);
    mag2 <= (source_imag * source_imag);

    end
  assign mag = mag1 + mag2;
endmodule
```

B.1.2 FFT(IP core)

```
// FFT.v

// Generated using ACDS version 18.1 625

`timescale 1 ps / 1 ps
module FFT (
                input  wire        clk,          //    clk.clk
                input  wire        reset_n,      //    rst.reset_n
                input  wire        sink_valid,   //   sink.sink_valid
                output wire        sink_ready,   //       .sink_ready
                input  wire [1:0]  sink_error,   //       .sink_error
                input  wire        sink_sop,     //       .sink_sop
                input  wire        sink_eop,     //       .sink_eop
                input  wire [31:0] sink_real,    //       .sink_real
                input  wire [31:0] sink_imag,    //       .sink_imag
                input  wire [10:0] fftpts_in,    //       .fftpts_in
                input  wire [0:0]  inverse,      //       .inverse
                output wire        source_valid, // source.source_valid
                input  wire        source_ready, //       .source_ready
                output wire [1:0]  source_error, //       .source_error
                output wire        source_sop,   //       .source_sop
                output wire        source_eop,   //       .source_eop
                output wire [31:0] source_real,  //       .source_real
                output wire [31:0] source_imag,  //       .source_imag
                output wire [10:0] fftpts_out    //       .fftpts_out
        );
```

```
	FFT_fft_ii_0 fft_ii_0 (
		.clk          (clk),          //    clk.clk
		.reset_n      (reset_n),      //    rst.reset_n
		.sink_valid   (sink_valid),   //   sink.sink_valid
		.sink_ready   (sink_ready),   //       .sink_ready
		.sink_error   (sink_error),   //       .sink_error
		.sink_sop     (sink_sop),     //       .sink_sop
		.sink_eop     (sink_eop),     //       .sink_eop
		.sink_real    (sink_real),    //       .sink_real
		.sink_imag    (sink_imag),    //       .sink_imag
		.fftpts_in    (fftpts_in),    //       .fftpts_in
		.inverse      (inverse),      //       .inverse
		.source_valid (source_valid), // source.source_valid
		.source_ready (source_ready), //       .source_ready
		.source_error (source_error), //       .source_error
		.source_sop   (source_sop),   //       .source_sop
		.source_eop   (source_eop),   //       .source_eop
		.source_real  (source_real),  //       .source_real
		.source_imag  (source_imag),  //       .source_imag
		.fftpts_out   (fftpts_out)    //       .fftpts_out
	);

endmodule

// (C) 2001-2018 Intel Corporation. All rights reserved.
// Your use of Intel Corporation's design tools, logic functions and other
// software and tools, and its AMPP partner logic functions, and any output
// files from any of the foregoing (including device programming or simulation
// files), and any associated documentation or information are expressly subject
// to the terms and conditions of the Intel Program License Subscription
// Agreement, Intel FPGA IP License Agreement, or other applicable
// license agreement, including, without limitation, that your use is for the
// sole purpose of programming logic devices manufactured by Intel and sold by
// Intel or its authorized distributors.  Please refer to the applicable
// agreement for further details.



module FFT_fft_ii_0 (
  input clk,
  input reset_n,
	input [10 : 0] fftpts_in,
	input    [0 : 0] inverse,
	input    sink_valid,
	input    sink_sop,
	input    sink_eop,
        input     logic [31 : 0] sink_real,
        input     logic [31 : 0] sink_imag,
        input     logic [1 : 0] sink_error,
        input     source_ready,
output [10 : 0] fftpts_out,
        output sink_ready,
        output [1 : 0] source_error,
        output source_sop,
        output source_eop,
        output source_valid,
        output [31 : 0] source_real,
        output [31 : 0] source_imag
        );

        auk_dspip_r22sdf_top #(
                .DEVICE_FAMILY_g("Cyclone V"),
                .MAX_FFTPTS_g(1024),
                .NUM_STAGES_g(5),
                .DATAWIDTH_g(32),
                .TWIDWIDTH_g(32),
                .MAX_GROW_g (0),
                .TWIDROM_BASE_g("FFT_fft_ii_0_"),
                .DSP_ROUNDING_g(0),
                .INPUT_FORMAT_g("NATURAL_ORDER"),
                .OUTPUT_FORMAT_g("NATURAL_ORDER"),
                .REPRESENTATION_g("FIXEDPT"),
                .DSP_ARCH_g(2),
    .PRUNE_g("2,2,3,2,0")
        )
        auk_dspip_r22sdf_top_inst (
                .clk(clk),
                .clk_ena(1'b1),
                .reset_n(reset_n),
                .fftpts_in(fftpts_in),
                .fftpts_out(fftpts_out),
                .inverse(inverse[0]),
                .sink_valid(sink_valid),
                .sink_sop(sink_sop),
                .sink_eop(sink_eop),
                .sink_real(sink_real),
                .sink_imag(sink_imag),
                .sink_ready(sink_ready),
                .sink_error(sink_error),
                .source_error(source_error),
                .source_ready(source_ready),
                .source_sop(source_sop),
                .source_eop(source_eop),
                .source_valid(source_valid),
                .source_real(source_real),
                .source_imag(source_imag)
        );
endmodule
```

B.1.3 playtone

```
module playtone(CLOCK_50,reset, note_num, audio_out, ready_read, ready_out);//KEY,SW,LEDR
        input CLOCK_50;
        input reset;
        //wire reset = KEY[0];

        input [6:0]note_num;
        //wire [5:0]note_num = SW;//A2-22/C3-25/E3-29/G3-32/A3-34

        //connect to audio controller
        input ready_read;// = SW[6];
        output reg [31:0]audio_out;
        //output [9:0]LEDR;
        output reg ready_out;// =1'b1;

        reg [31:0]countdata;// = 11'b00000000000;


        reg [6:0]counter;//number of bits one cycle of note
        reg [7:0]count_note;//count in one cycle

        always @(*) begin
                if(reset == 1'b1) begin
                        counter = 7'b0110111;//half of the cycle
                        end
                else begin
                        if (note_num == 6'b010110)begin
                                counter = 7'b1101101;
                                end
                        else if (note_num == 6'b011001)begin
                                counter = 7'b1011100;
                                end
                        else if (note_num == 6'b011101)begin
                                counter = 7'b1001010;
                                end
                        else if (note_num == 6'b100000)begin
                        counter = 7'b0111101;
                        end
                    else begin
                        counter = 7'b0110111;
                        end
                end//else
            end//always

            always @(posedge CLOCK_50) begin
                if (reset == 1'b1)begin
                    audio_out <= 32'h000000000;
                    ready_out <= 1'b1;
                    end
                else if (!ready_read) begin
                    audio_out <= audio_out;
                    end
                else if (counter == 7'b0110111) begin
                    audio_out <= 32'b0;
                    end
                else begin
                    if (countdata == 32'hFFFFFFFF) begin
                        countdata <= 11'b0;
                        ready_out <= 1'b0;
                        audio_out <= audio_out;
                        end
                    else if (countdata == 32'h0FFFFFFF) begin
                        countdata <=countdata +1;
                        ready_out <= 1'b1;
                        end
                    else begin
                        if (count_note == (counter <<< 1)) begin
                            count_note <= 0;
                            end
                        else begin
                            count_note <= count_note +1;
                            if (count_note == counter) begin
                                audio_out <= 32'h00800000;
                                end
                            else if (count_note == 0)begin
                                audio_out <= 32'hFF7FFFFF;//magnitude of the
        audio

                                end
                            else begin
                                audio_out <= audio_out;

                                end
```

```
                                        end
                                ready_out <= 1'b0;
                                countdata <= countdata +1;
                        end

                end//else*/
        end//always

        //assign LEDR[9:7] = audio_out[25:23];
        //assign LEDR[6] = ready_out;
endmodule
```

B.2 maxFreqDetect (maxFreqDetect / max_finder/ maxFreq/ toneCheck)

B.2.0 maxFreqDetect

```
module maxFreqDetect(start, reset, clk, expected, mag, note, done, pitch_indicator);//, HEX4, HEX5, 
freakMax);//, LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);
        input start;
        input reset;
        input [63:0]mag;
        //input [10:0]pts;
        input clk;
        input [5:0]expected;


        output done;
        output [5:0]note;
        output [1:0]pitch_indicator;
        //output [63:0]freakMax;
        //wire [10:0]p;
        //output [6:0]HEX4, HEX5;
        //output [9:0]LEDR;
        //output [6:0]HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;

        wire [10:0]maxIndex;
        wire MIfound_FDstart;
        wire FDfound_TCstart;
        wire [63:0]max_freq;
        wire [63:0]MaxMag;
```

```
	//wire [1:0]ppppitch;

	//assign freakMax[63:0] = MaxMag[63:0];



	max_finder MFind(.clk(clk), .reset(1'b0), .start(1'b1), .mag(mag), .max_index(maxIndex),
	.done(done), .max_mag(MaxMag[63:0]));//, .LEDR(LEDR[8:0]));, .num_pts(p));

	maxFreq MFreq(.clk(clk), .max_index(maxIndex), .start(start), .reset(1'b0),
	.freq(max_freq[63:0]), .done(FDfound_TCstart));

	toneCheck TC(.clk(clk), .start(1'b1), .freq(max_freq[63:0]), .expected(expected),
	.note_Num(note), .pitch(pitch_indicator), .Done());
	//assign ppppitch[1:0] = pitch_indicator;
	//assign LEDR[9] = mag[27];
	//assign LEDR[3] = maxIndex[10];
	//assign LEDR[3] = MIfound_FDstart;
	//assign LEDR[2] = FDfound_TCstart;
	//assign LEDR[1] = done;
	//assign LEDR[0] = reset;

	/*
	hex_decoder H0(.hex_digit(pitch_indicator[1:0]), .segments(HEX0[6:0]));
	hex_decoder H1(.hex_digit(maxIndex[4:0]), .segments(HEX1[6:0]));
	hex_decoder H2(.hex_digit(maxIndex[9:5]), .segments(HEX2[6:0]));
	hex_decoder H3(.hex_digit(maxIndex[10]), .segments(HEX3[6:0]));
	hex_decoder H4(.hex_digit(note[5:4]), .segments(HEX5[6:0]));
	hex_decoder H5(.hex_digit(note[3:0]), .segments(HEX4[6:0]));
	//hex_decoder H5(.hex_digit(note[31:27]), .segments(HEX5[6:0]));*/



endmodule
```

B.2.1 max_finder

```
module max_finder(clk, reset, start, mag, max_index, done, max_mag ,LEDR);//, num_pts);
	//localparam SHIFT_BITS = 10;
	localparam NUM_PTS = 523;
	localparam HALF_WINDOW = 512;
	//localparam SHIFT_BITS = 3;
	input clk;
	input reset;
	//input [10:0]pts;
```

```
input [31:0]mag;
input start;
output reg [10:0]max_index;
output reg [63:0]max_mag;
output reg done;

output [8:0]LEDR;

assign LEDR[8:0]= counter[8:0];
//output reg [10:0]index0;
//output [10:0]num_pts;
//wire [10:0]num_pts;

reg [63:0]Max;
reg [10:0]MaxIndex = 11'b0;

reg [9:0]counter;

//assign num_pts[10:0] = 1<<SHIFT_BITS;
always @(posedge clk) begin
        if (reset == 1'b1)begin
                MaxIndex <= 0;
                Max <= 0;
                counter <= 0;
                done <= 0;
                max_index <= 0;
                //max_mag <= 0;
                end
        else if(start == 1'b0) begin
                MaxIndex <= 0;
                Max <= 0;
                counter <= 0;
                done <= 0;
                max_index <= 0;
                //max_mag <= 0;
                end
        else begin
                if (counter == 0) begin
                        done <= 1'b0;
                        end
                else if (counter == NUM_PTS) begin
                        done <= 1'b1;
                        counter <= 0;
                        MaxIndex <= 0;
                        Max <= 0;
                        //start <= 1'b0;
```

```
				end
			else if (counter > HALF_WINDOW+2) begin
				done <= 1'b0;
				//if (mag == 0) begin
					//index0 <= counter;
					//end
				if (Max < mag) begin
					MaxIndex <= (counter - HALF_WINDOW);
					Max <= mag;
					end
				else begin
					MaxIndex <= MaxIndex;
					Max <= Max;
				end
				//counter <= counter + 1;
				end//else if
			else begin
				done <= 1'b0;
				MaxIndex <= MaxIndex;
				Max <= Max;
				end
			counter <= counter + 1;
			end//else

		max_index <= MaxIndex;
		max_mag <= Max;
	end//always

endmodule
```

B.2.2 maxFreq

```
module maxFreq(clk, max_index, start, reset, freq, done);
	localparam SAMPLE_FREQ = 48000; //sample frequency
	localparam SHIFT_BITS = 10;
	//localparam SHIFT_BITS = 3;
	input [10:0]max_index;
	//input [10:0]pts; //SHIFT_BITS
	input start;
	input reset;
	input clk;

	output [31:0]freq;
	output done;
```

```
        reg [31:0]freq_reg;
        reg [31:0]mult;
        reg Done = 1'b0;
        //reg [15:0]shifted;

        //bitwise operation
        always@(posedge clk)
        begin
                if(reset == 1'b1) begin
                        freq_reg <= 32'b0;
                        Done <= 1'b0;
                end
                else if(start) begin
                        mult <= SAMPLE_FREQ*max_index;
                        //shifted <= mult >>> pts;
                        freq_reg <= (mult >> SHIFT_BITS);
                        //freq_reg <= (mult >>> pts);
                        Done <= 1'b1;
                end
                else begin
                        freq_reg <= freq_reg;
                        Done <= 1'b0;
                end
        end
        assign done = Done;
        assign freq = freq_reg;
endmodule
```

B.2.3 toneCheck

```
module toneCheck(clk, start, freq, expected, note_Num, pitch, Done);
        input [31:0]freq;
        input start;
        input [5:0]expected;
        input clk;

        output [5:0]note_Num;
        output [1:0]pitch; //low = 00, high = 11, correct = 10;
        output Done;

        reg [5:0]noteNum;
        reg [1:0]isLow_High = 2'b01;
        reg done;
```

```
localparam A_2 = 22;//010110
localparam C_3 = 25;//011001
localparam E_3 = 29;//011101
localparam G_3 = 32;//100000
//localparam DEF = 0;

always@(posedge clk)
begin
        if(start == 1'b1) begin
                if (expected == A_2) begin
                        //if(freq <= 241)begin
                        noteNum <= A_2;
                        if(freq < 210)begin
                                isLow_High <= 2'b00;
                                //done <= 1'b1;
                        end
                        else if(freq > 230)begin
                                isLow_High <= 2'b11;
                                //done <= 1'b1;
                        end
                        else begin
                                isLow_High <= 2'b10;
                                //done <= 1'b0;
                        end
                        done <= 1'b1;
                end
                else if(expected == C_3) begin
                        //else if((freq > 241) && (freq <= 296)) begin
                        noteNum <= C_3;
                        if(freq < 240) begin
                                isLow_High <= 2'b00;
                                //done <= 1'b1;
                        end
                        else if(freq > 280) begin
                                isLow_High <= 2'b11;
                                //done <= 1'b1;
                        end
                        else begin
                                isLow_High <= 2'b10;
                                //done <= 1'b0;
                        end
                        done <= 1'b1;
                end
                else if(expected == E_3) begin
                        //else if((freq > 296) && (freq <= 361)) begin
                        noteNum <= E_3;
                if(freq < 310) begin
                    isLow_High <= 2'b00;
                    //done <= 1'b1;
                end
                else if(freq > 360) begin
                    isLow_High <= 2'b11;
                    //done <= 1'b1;
                end
                else begin
                    isLow_High <= 2'b10;
                    //done <= 1'b0;
                end
                done <= 1'b1;
            end
            else if(expected == G_3) begin
                //else if(freq > 361) begin
                noteNum <= G_3;
                if(freq < 380) begin
                    isLow_High <= 2'b00;
                    //done <= 1'b1;
                end
                else if(freq > 400) begin
                    isLow_High <= 2'b11;
                    //done <= 1'b1;
                end
                else begin
                    isLow_High <= 2'b10;
                    //done <= 1'b0;
                end
                done <= 1'b1;
            end
            else begin
                noteNum <= 0;
                isLow_High <= 2'b01;
                done <= 1'b0;
            end
        end
        else begin
            noteNum <= 6'b0;
            isLow_High <= 2'b01;
            done <= 1'b0;
        end
    end

assign Done = done;
assign note_Num = noteNum;
```

```
	assign pitch = isLow_High;
endmodule
```

B.3 VGA display (mouse_display/ ps2_mouse/ transfer/ram)

B.3.0 mouse_display(datapath, control)

```
module mouse_display(KEY, PS2_CLK, PS2_DAT,
						HEX0, HEX1, HEX2, HEX3,
						note_num2, lowhigh, CLOCK_50, VGA_R,
						VGA_G,
						VGA_B,
						VGA_HS,
						VGA_VS,
						VGA_BLANK_N,
						VGA_SYNC_N,
						VGA_CLK, mouse_note, SW
						);

	input	[0:0]KEY;
	input [0:0]SW;


	inout	PS2_CLK;
	inout	PS2_DAT;


	output [6:0]HEX0;
	output [6:0]HEX1;
	output [6:0]HEX2;
	output [6:0]HEX3;
	//output [1:0]LEDR;


	wire [5:0]note_num1;
	input [5:0]note_num2;
	//assign note_num2[5:0] = SW[5:0];
	input [1:0]lowhigh;
	//assign lowhigh[1:0] = SW[9:8];
	//input ready; //ready to start FSM
	wire reset;
	assign reset = KEY[0];
//input ismouse;
	input CLOCK_50;
```

```
output [7:0] VGA_R;
output [7:0] VGA_G;
output [7:0] VGA_B;
output VGA_HS;
output VGA_VS;
output VGA_BLANK_N;
output VGA_SYNC_N;
output VGA_CLK;

output [5:0]mouse_note;

//input click;

wire [7:0]plotx;
wire [6:0]ploty;
wire [8:0]plotc;

//mouse input
wire [7:0]mx;
wire [6:0]my;
wire [7:0]MX;
wire [6:0]MY;
assign MY = my - 7'h32;
assign MX = mx - 8'h0E;
reg [5:0]note_num;

wire outc;//if the click is valid
reg [1:0]pitch;

reg [16:0]counter;
reg enable;
wire Enable = enable;

wire [8:0]background;

always@(*) begin
        if ((outc) && (SW[0]==1'b0)) begin
                note_num = note_num1;
                pitch = 2'b10;
                end
        else if (SW[0] == 1'b1) begin
                note_num = note_num2;
                pitch[1:0] = lowhigh[1:0];
                end
        else begin
                note_num = 6'b0;
                end
            end

        wire [1:0]PITCH;
        assign PITCH[1:0] = pitch[1:0];
        wire [5:0]NOTE;
        assign NOTE [5:0]= note_num[5:0];

        always @(posedge CLOCK_50) begin
            if (counter == 24999) begin
                counter <= 17'b0;
                enable <= 1'b1;
                end
            else begin
                counter <= counter +1;
                enable <= 1'b0;
                end
        end

        wire ERS;
        wire ERM;
        wire PS;
        wire PM;
        wire [5:0]county;
        wire [7:0]pastmx;
        wire [6:0]pastmy;


        //assign LEDR[0] = enable;
        //assign LEDR[5:0] = note_num1[5:0];
        //assign LEDR[1] = outc;
        assign mouse_note[5:0] = outc ? note_num1[5:0] : 6'b0;

        ram_background R0(.address((pastmy)*160 + pastmx), .clock(CLOCK_50), .wren(1'b0),
.q(background));

        transfer T0(.CLOCK_50(CLOCK_50), .KEY(KEY[0]), .PS2_CLK(PS2_CLK),
.PS2_DAT(PS2_DAT),
        .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3),
        .note_num(note_num1), .outc(outc), .MX(mx), .MY(my));

        control C0(.reset(~KEY[0]), .enable(Enable), .county(county), .clock(CLOCK_50),
.ERS(ERS),
        .ERM(ERM), .PS(PS), .PM(PM));

        datapath D0(.reset(~KEY[0]), .clock(CLOCK_50), .ERS(ERS), .ERM(ERM), .PS(PS),
        .background(background),.PM(PM), .note_num(NOTE), .pastmx(pastmx), .pastmy(pastmy),
        .pitch(PITCH), .mx(MX), .my(MY), .county(county), .outc(outc), .plotx(plotx),
        .ploty(ploty), .plotc(plotc));

        vga_adapter VGA(
                .resetn(reset),
                .clock(CLOCK_50),
                .colour(plotc),
                .x(plotx),
                .y((ERS|PS | ~(ERS|ERM|PS|PM)) ? ploty+county : ploty),
                .plot(1'b1),
                /* Signals for the DAC to drive the monitor. */
                .VGA_R(VGA_R),
                .VGA_G(VGA_G),
                .VGA_B(VGA_B),
                .VGA_HS(VGA_HS),
                .VGA_VS(VGA_VS),
                .VGA_BLANK(VGA_BLANK_N),
                .VGA_SYNC(VGA_SYNC_N),
                .VGA_CLK(VGA_CLK));
            defparam VGA.RESOLUTION = "160x120";
            defparam VGA.MONOCHROME = "FALSE";
            defparam VGA.BITS_PER_COLOUR_CHANNEL = 3;
            defparam VGA.BACKGROUND_IMAGE = "background.mif";
endmodule



module control(reset, enable, county, clock, ERS, ERM, PS, PM);
        input reset;
        input enable;
        input [5:0]county;
        input clock;
        output reg ERS;
        output reg ERM;
        output reg PS;
        output reg PM;

        reg [2:0]current_state, next_state;
        parameter ers = 3'b000,
                                erm = 3'b001,
                                ps = 3'b010,
                                pm = 3'b011,
                                Wait = 3'b100;
        always @(*)
        begin: state_table
        case (current_state)
                ers: if (county == 48) begin
                                next_state = erm;
                                end
                                else begin
                                next_state = ers;
                                end
                erm: next_state = ps;
                ps: if (county == 48) begin
                                next_state = pm;
                                end
                        else begin
                                next_state = ps;
                                end
                pm: next_state = Wait;
                Wait: next_state = enable ? ers : Wait;
        endcase
end

always @(*)
begin: state_t
        case (current_state)
                ers: begin
                        ERS = 1'b1;
                        ERM = 1'b0;
                        PS = 1'b0;
                        PM = 1'b0;
                        end
                erm: begin
                        ERS = 1'b0;
                        ERM = 1'b1;
                        PS = 1'b0;
                        PM = 1'b0;
                        end
                ps: begin
                        ERS = 1'b0;
                        ERM = 1'b0;
                        PS = 1'b1;
                        PM = 1'b0;
                        end
                pm: begin
                        ERS = 1'b0;
                        ERM = 1'b0;
                        PS = 1'b0;
                        PM = 1'b1;
                        end
```

```
                              Wait: begin
                                        ERS = 1'b0;
                                        ERM = 1'b0;
                                        PS = 1'b0;
                                        PM = 1'b0;
                                        end
                    endcase
          end


          always @(posedge clock)
          begin: state_FFs
     if(reset == 1'b1)begin
        current_state <= ers;
        end
     else begin
                              current_state <= next_state;
                              end
   end
endmodule



module datapath(reset, clock, ERS, ERM, PS, PM, background, note_num, pitch,
county, mx, my, pastmx, pastmy, outc, plotx, ploty, plotc);
          input reset;
          input clock;
          input ERS;
          input ERM;
          input PS;
          input PM;
          input [8:0]background;
          input [5:0]note_num;
          input [1:0]pitch;
          input [7:0]mx;
          input [6:0]my;
          input outc;
          output reg[7:0]plotx;
          output reg[6:0]ploty;
          output reg[8:0]plotc;

          output reg [7:0]pastmx;
          output reg [6:0]pastmy;
          reg [7:0]pastx;

          //assign pastx = note_x;
```

```
reg [7:0]note_x;// register x-position of the string
//reg [6:0]note_y; //register y-position of the string
reg [8:0]note_c;
output reg [5:0]county;

always @(posedge clock) begin
        if (note_num == 6'b010110)begin
                note_x <= 8'b01000110;//70
                end
        else if (note_num == 6'b011001)begin
                note_x <= 8'b01001001;//73
                end
        else if (note_num == 6'b011101)begin
                note_x <= 8'b01001100;//76
                end
        else if (note_num == 6'b100000)begin
                note_x <= 8'b01000011;//67
                end
        else begin
                note_x <= pastx;
                end
end

always @(posedge clock) begin

        if (note_num ==  6'b010110 || note_num ==  6'b011001 ||
                note_num ==  6'b011101 || note_num ==  6'b100000 ) begin
                if (pitch == 2'b00) begin//low
                        note_c <= 9'b111001001;
                        end
                else if (pitch == 2'b11) begin//high
                        note_c <= 9'b001010101;
                        end
                else if (pitch == 2'b10) begin
                        note_c <= 9'b010101010;//right
                        end
                else begin
                        note_c <= 9'b111111111;//default-white
                        end
                end
        else begin
                note_c <= 9'b111111111;
                end
        end
```

```
always @(posedge clock)begin
        if (reset == 1'b1) begin
                plotx <= note_x;
                ploty <= 7'b0010000;
                if (county == 48) begin
                        //ploty <= ploty + county;
                        county <= 6'b0;
                        end
                else begin
                        //ploty <= ploty + county;
                        county <= county + 1;
                        end
                plotc <= 9'b111111111;
                end/*
        if (erase) begin
                plotx <= note_x;
                ploty <= 7'b00010000;
                if (county == 48) begin
                        //ploty <= ploty + county;
                        county <= 6'b0;
                        end
                else begin
                        //ploty <= ploty + county;
                        county <= county + 1;
                        end
                plotc <= 9'b111111111;
                end*/
        if (ERS) begin
                plotx <= note_x;
                ploty <= 7'b0010000;
                plotc <= note_c;
                if (county == 48) begin
                        //ploty <= ploty + county;
                        county <= 6'b0;
                        end
                else begin
                        //ploty <= ploty + county;
                        county <= county + 1;
                        end
                end
        else if (ERM) begin
                plotx <= pastmx;
                ploty <= pastmy;
                plotc <= background;
                end
        else if (PS) begin
```

```
                        plotx <= note_x;
                        ploty <= 7'b0010000;
                        pastx <= note_x;
                        plotc <= note_c;
                        if (county == 48) begin
                                //ploty <= ploty + county;
                                county <= 6'b0;
                                end
                        else begin
                                //ploty <= ploty + county;
                                county <= county + 1;
                                end
                        end
                else if (PM) begin
                        plotx <= mx;
                        pastmx <= mx;
                        ploty <= my;
                        pastmy <= my;
                        plotc <= 9'b111110000;
                        end
                /*else begin
                        plotx <= plotx;
                        ploty <= ploty;
                        plotc <= plotc;

                        end*/
                /*if (county == 48) begin
                        county <= 6'b0;
                        end
                else begin
                        county<= county +1;
                        end*/
                end

endmodule



        /*
module control(ready, reset, clock, color);
        input ready;
        input reset;
        input clock;
        output reg color;
        //output [2:0]LEDR;

        reg [22:0]DelayCounter = 23'b0;
```

```
	reg [1:0] current_state, next_state;
	parameter start = 2'b00,
		draw = 2'b01,
		Wait = 2'b10;

	always @(*)
	begin: state_table
	  case (current_state)
				start: next_state = ready? draw : start;
				draw: next_state = Wait;
				Wait: next_state = draw;
				default: next_state = start;
			endcase
	end//fsm

	//output datapath signals
	always @(*) begin
			color = 1'b0;
			case (current_state)
				start: color = 1'b0;//do not change the color
				draw: color = 1'b1;//change the color
				Wait: color = 1'b0;
			endcase
	end//


	always@(posedge clock)
begin: state_FFs
  if(!reset)begin
    current_state <= start;
    end
  else begin
				if (current_state == Wait) begin
			if (DelayCounter == 8333334) begin
					DelayCounter <= 23'b0;
						current_state <= next_state;
						end
					else begin
					  DelayCounter <= DelayCounter +1;
						end
					end
				else begin
					current_state <= next_state;
					end
			end
  end

	//assign LEDR[1:0] = current_state[1:0];

endmodule*/

/*
module datapath(note_num, lowhigh, clock, reset, color, plotx, ploty, plotc, county, mx, my);
	input [5:0]note_num;
	input [1:0]lowhigh;
	input clock;
	input reset;
	input color;

	//output to vga adapter
	output reg [7:0]plotx;
	output reg [6:0]ploty;
	output reg [8:0]plotc;

	output reg [5:0]county;
	output [8:0]LEDR;

	input [7:0]mx;
	input [6:0]my;

	reg [5:0]counter = 6'b0;




	always @(posedge clock) begin
		if (counter == 6'b110010) begin//erase the mouse
			county <= 6'b0;
			plotx <= mx;
			ploty <= my;
			plotc <= 9'b111111111;
			counter <= 6'b0;
			end
		else if (counter == 6'b000000) begin//draw mouse
			county <= 6'b0;
			plotx <= mx;
			ploty <= my;
			plotc <= //background color
			counter <= counter + 1;
			end
		else begin//draw the string
```

```
ploty <= 7'b00010000;
plotc <= 9'b111111111;
if (!reset) begin
        plotc <= 9'b111111111;
        end
else if (click) begin
        plox <= mx;
        plotc <= 9'b010101010;
        end

else begin

//control plot x
        if (note_num == 6'b010110)begin
                plotx <= 8'b01000110;//70
                end
        else if (note_num == 6'b011001)begin
                plotx <= 8'b01001001;//73
                end
        else if (note_num == 6'b011101)begin
                plotx <= 8'b01001100;//76
                end
        else if (note_num == 6'b100000)begin
                plotx <= 8'b01000011;//67
                end

//control plot color
        if (color) begin
                if (lowhigh == 2'b00) begin//low
                        plotc <= 9'b111001001;
                        end
                else if (lowhigh == 2'b11) begin//high
                        plotc <= 9'b001010101;
                        end
                else begin
                        plotc <= 9'b010101010;//right
                        end
                end
        else begin
                plotc <= plotc;
                end
end//end else -plotx,pltc
if (county == 48) begin
        county <= 6'b0;
        end
else begin
```

```
                                        county <= county+1;
                                        end
                                counter <= counter +1;
                                end

                end//always block

endmodule*/
```

B.3.1 ps2_mouse(hex_decoder)

```
module ps2_mouse(CLOCK_50, KEY, PS2_CLK,
        PS2_DAT, HEX0, HEX1, HEX2, HEX3, xpos, ypos, click);
// Inputs
        input   CLOCK_50;
        input   [0:0]KEY;

// Bidirectionals
        inout   PS2_CLK;
        inout   PS2_DAT;

//output
        output [6:0]HEX0;
        output [6:0]HEX1;
        output [6:0]HEX2;
        output [6:0]HEX3;
        //output [0:0]LEDR;


        //output reg [7:0]xpos;
        //output reg [7:0]ypos;
        output reg click;

        wire    [7:0]ps2_key_data;
        wire    ps2_key_pressed;

// Internal Registers
        reg             [7:0]last_data_received;
        reg     [1:0]counter;
        output reg      [7:0]xpos = 8'b0;
        output reg      [7:0]ypos = 8'b0;

        PS2_Controller PS2 (
        // Inputs
```

```
	.CLOCK_50				(CLOCK_50),
	.reset				(~KEY[0]),

	// Bidirectionals
	.PS2_CLK				(PS2_CLK),
	.PS2_DAT				(PS2_DAT),

	// Outputs
	.received_data		(ps2_key_data),
	.received_data_en		(ps2_key_pressed),
);

	always @(posedge CLOCK_50)
	begin
		if (~KEY[0])begin
			last_data_received <= 8'h00;
			xpos <= 8'd00;
			ypos <= 8'd00;
			end
		else if (ps2_key_pressed == 1'b1)begin
			last_data_received <= ps2_key_data;
			if (counter == 2'b00) begin
				click <= last_data_received[0];
				counter <= counter +1;
				end
			else if (counter == 2'b01) begin
				xpos <= xpos + last_data_received;
				counter <= counter +1;
				end
			else if (counter == 2'b10) begin
				ypos <= ypos + last_data_received;
				counter <= 2'b00;
				end
			end
	end


	//assign LEDR[0] = click;
	//assign mx[7:0] = xpos[7:0];
	//assign my[6:0] = ypos[6:0];
	hex_decoder H0(xpos[3:0], HEX0);
	hex_decoder H1(xpos[7:4], HEX1);
	hex_decoder H2(ypos[3:0], HEX2);
	hex_decoder H3(ypos[7:4], HEX3);

endmodule
```

```
module hex_decoder(hex_digit, segments);
   input [3:0] hex_digit;
   output reg [6:0] segments;

   always @(*)
      case (hex_digit)
         4'h0: segments = 7'b100_0000;
         4'h1: segments = 7'b111_1001;
         4'h2: segments = 7'b010_0100;
         4'h3: segments = 7'b011_0000;
         4'h4: segments = 7'b001_1001;
         4'h5: segments = 7'b001_0010;
         4'h6: segments = 7'b000_0010;
         4'h7: segments = 7'b111_1000;
         4'h8: segments = 7'b000_0000;
         4'h9: segments = 7'b001_1000;
         4'hA: segments = 7'b000_1000;
         4'hB: segments = 7'b000_0011;
         4'hC: segments = 7'b100_0110;
         4'hD: segments = 7'b010_0001;
         4'hE: segments = 7'b000_0110;
         4'hF: segments = 7'b000_1110;
         default: segments = 7'h7f;
      endcase
endmodule
```

B.3.2 transfer

```
module transfer(CLOCK_50, KEY, PS2_CLK,
         PS2_DAT, HEX0, HEX1, HEX2, HEX3, note_num, outc , MX, MY);
         input CLOCK_50;
         input    [0:0]KEY;

         inout    PS2_CLK;
         inout    PS2_DAT;


         output [6:0]HEX0;
         output [6:0]HEX1;
         output [6:0]HEX2;
         output [6:0]HEX3;
```

```
//output [0:0]LEDR;
output [7:0]MX;
output [6:0]MY;

wire [7:0]mx;
wire [7:0]my;
wire click;
output reg [5:0]note_num;
output reg outc;


ps2_mouse P0(.CLOCK_50(CLOCK_50), .KEY(KEY[0]), .PS2_CLK(PS2_CLK),
.PS2_DAT(PS2_DAT), .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2),
.HEX3(HEX3), .xpos(mx), .ypos(my), .click(click));

always@(posedge CLOCK_50) begin
        outc <= 1'b0;
        if ((mx == 8'h54) && (my >= 8'h41) && (my <= 8'h71)) begin
                note_num <= 6'b010110;//76
                if (click) begin
                        outc <= 1'b1;
                        end
                else begin
                        outc <= 1'b0;
                        end
                end
        else if ((mx == 8'h57) && (my >= 8'h41) && (my <= 8'h71)) begin
                note_num <= 6'b011001;//73
                if (click) begin
                        outc <= 1'b1;
                        end
                else begin
                        outc <= 1'b0;
                        end
                end
        else if ((mx == 8'h5A) && (my >= 8'h41) && (my <= 8'h71)) begin
                note_num <= 6'b011101;//70
                if (click) begin
                        outc <= 1'b1;
                        end
                else begin
                        outc <= 1'b0;
                        end
                end
        else if ((mx == 8'h51) && (my >= 8'h41) && (my <= 8'h71)) begin
                note_num <= 6'b100000;//67
```

```
				if (click) begin
					outc <= 1'b1;
					end
				else begin
					outc <= 1'b0;
					end
				end
			else begin
				note_num <= 6'b0000000;
				outc <= 1'b0;
				end
			end

			//assign LEDR[5:0] = note_num[5:0];
			//assign LEDR[7] = outc;
			//assign LEDR[7:0] = my[7:0];
			assign MX[7:0] = mx[7:0];
			assign MY[6:0] = my[6:0];
endmodule
```

B.3.3 ram for mouse erase(IP core)

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

// ============================================================
// File Name: ram_background.v
// Megafunction Name(s):
// 			altsyncram
//
// Simulation Library Files(s):
// 			altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 18.1.0 Build 625 09/12/2018 SJ Lite Edition
// ************************************************************


//Copyright (C) 2018  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
```

```
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module ram_background (
	address,
	clock,
	data,
	wren,
	q);

	input	[14:0]  address;
	input	  clock;
	input	[8:0]  data;
	input	  wren;
	output	[8:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	  clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [8:0] sub_wire0;
	wire [8:0] q = sub_wire0[8:0];

	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.data_a (data),
				.wren_a (wren),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
				.address_b (1'b1),
                                        .addressstall_a (1'b0),
                                        .addressstall_b (1'b0),
                                        .byteena_a (1'b1),
                                        .byteena_b (1'b1),
                                        .clock1 (1'b1),
                                        .clocken0 (1'b1),
                                        .clocken1 (1'b1),
                                        .clocken2 (1'b1),
                                        .clocken3 (1'b1),
                                        .data_b (1'b1),
                                        .eccstatus (),
                                        .q_b (),
                                        .rden_a (1'b1),
                                        .rden_b (1'b1),
                                        .wren_b (1'b0));
        defparam
                altsyncram_component.clock_enable_input_a = "BYPASS",
                altsyncram_component.clock_enable_output_a = "BYPASS",
                altsyncram_component.init_file = "background.mif",
                altsyncram_component.intended_device_family = "Cyclone V",
                altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 32768,
                altsyncram_component.operation_mode = "SINGLE_PORT",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a = "UNREGISTERED",
                altsyncram_component.power_up_uninitialized = "FALSE",
                altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
                altsyncram_component.widthad_a = 15,
                altsyncram_component.width_a = 9,
                altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "9"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "background.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "32768"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "15"
// Retrieval info: PRIVATE: WidthData NUMERIC "9"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "background.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "32768"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "15"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "9"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 15 0 INPUT NODEFVAL "address[14..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 9 0 INPUT NODEFVAL "data[8..0]"
// Retrieval info: USED_PORT: q 0 0 9 0 OUTPUT NODEFVAL "q[8..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 15 0 address 0 0 15 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 9 0 data 0 0 9 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 9 0 @q_a 0 0 9 0
// Retrieval info: GEN_FILE: TYPE_NORMAL ram_background.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram_background.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram_background.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram_background.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram_background_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram_background_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
```

B.3.4 VGA_display

```
vga_adapter
/* VGA Adapter
 * ----------------
 *
 * This is an implementation of a VGA Adapter. The adapter uses VGA mode signalling to initiate
 * a 640x480 resolution mode on a computer monitor, with a refresh rate of approximately 60Hz.
 * It is designed for easy use in an early digital logic design course to facilitate student
 * projects on the Altera DE2 Educational board.
 *
 * This implementation of the VGA adapter can display images of varying colour depth at a resolution
of
 * 320x240 or 160x120 superpixels. The concept of superpixels is introduced to reduce the amount of
on-chip
 * memory used by the adapter. The following table shows the number of bits of on-chip memory used
by
 * the adapter in various resolutions and colour depths.
 *
 *
-----------------------------------------------------------------------------------------------------------------
----
 * Resolution | Mono    | 8 colours | 64 colours | 512 colours | 4096 colours | 32768 colours | 262144
colours | 2097152 colours |
 *
-----------------------------------------------------------------------------------------------------------------
----
 * 160x120    |  19200 |     57600 |     115200 |      172800 |       230400 |        288000 |        345600 |
403200 |
 * 320x240    |  78600 |    230400 | ############## Does not fit
################################################################## |
 *
--------------------------------------------------------------------------------------------------------------------
----
 *
 * By default the adapter works at the resolution of 320x240 with 8 colours. To set the adapter in any
of
 * the other modes, the adapter must be instantiated with specific parameters. These parameters are:
 * - RESOLUTION - a string that should be either "320x240" or "160x120".
 * - MONOCHROME - a string that should be "TRUE" if you only want black and white colours, and
"FALSE"
 *             otherwise.
 * - BITS_PER_COLOUR_CHANNEL  - an integer specifying how many bits are available to
describe each colour
 *                    (R,G,B). A default value of 1 indicates that 1 bit will be used for red
 *                    channel, 1 for green channel and 1 for blue channel. This allows 8 colours
 *                    to be used.
 *
 * In addition to the above parameters, a BACKGROUND_IMAGE parameter can be specified. The
parameter
 * refers to a memory initilization file (MIF) which contains the initial contents of video memory.
 * By specifying the initial contents of the memory we can force the adapter to initially display an
 * image of our choice. Please note that the image described by the BACKGROUND_IMAGE file
will only
 * be valid right after your program the DE2 board. If your circuit draws a single pixel on the screen,
 * the video memory will be altered and screen contents will be changed. In order to restore the
background
 * image your circuti will have to redraw the background image pixel by pixel, or you will have to
 * reprogram the DE2 board, thus allowing the video memory to be rewritten.
 *
 * To use the module connect the vga_adapter to your circuit. Your circuit should produce a value for
 * inputs X, Y and plot. When plot is high, at the next positive edge of the input clock the vga_adapter
 * will change the contents of the video memory for the pixel at location (X,Y). At the next redraw
 * cycle the VGA controller will update the contants of the screen by reading the video memory and
copying
 * it over to the screen. Since the monitor screen has no memory, the VGA controller has to copy the
 * contents of the video memory to the screen once every 60th of a second to keep the image stable.
Thus,
 * the video memory should not be used for other purposes as it may interfere with the operation of
the
 * VGA Adapter.
 *
 * As a final note, ensure that the following conditions are met when using this module:
 * 1. You are implementing the the VGA Adapter on the Altera DE2 board. Using another board may
change
 *    the amount of memory you can use, the clock generation mechanism, as well as pin assignments
required
```

```
 *    to properly drive the VGA digital-to-analog converter.
 * 2. Outputs VGA_* should exist in your top level design. They should be assigned pin locations on
the
 *    Altera DE2 board as specified by the DE2_pin_assignments.csv file.
 * 3. The input clock must have a frequency of 50 MHz with a 50% duty cycle. On the Altera DE2
board
 *    PIN_N2 is the source for the 50MHz clock.
 *
 * During compilation with Quartus II you may receive the following warnings:
 * - Warning: Variable or input pin "clocken1" is defined but never used
 * - Warning: Pin "VGA_SYNC" stuck at VCC
 * - Warning: Found xx output pins without output pin load capacitance assignment
 * These warnings can be ignored. The first warning is generated, because the software generated
 * memory module contains an input called "clocken1" and it does not drive logic. The second
warning
 * indicates that the VGA_SYNC signal is always high. This is intentional. The final warning is
 * generated for the purposes of power analysis. It will persist unless the output pins are assigned
 * output capacitance. Leaving the capacitance values at 0 pf did not affect the operation of the
module.
 *
 * If you see any other warnings relating to the vga_adapter, be sure to examine them carefully. They
may
 * cause your circuit to malfunction.
 *
 * NOTES/REVISIONS:
 * July 10, 2007 - Modified the original version of the VGA Adapter written by Sam Vafaee in 2006.
The module
 *                 now supports 2 different resolutions as well as uses half the memory compared to
prior
 *                 implementation. Also, all settings for the module can be specified from the point
 *                 of instantiation, rather than by modifying the source code. (Tomasz S. Czajkowski)
 */

module vga_adapter(
                        resetn,
                        clock,
                        colour,
                        x, y, plot,
                        /* Signals for the DAC to drive the monitor. */
                        VGA_R,
                        VGA_G,
                        VGA_B,
                        VGA_HS,
                        VGA_VS,
                        VGA_BLANK,
                        VGA_SYNC,
                        VGA_CLK);

          parameter BITS_PER_COLOUR_CHANNEL = 1;
          /* The number of bits per colour channel used to represent the colour of each pixel. A value
           * of 1 means that Red, Green and Blue colour channels will use 1 bit each to represent the
intensity
           * of the respective colour channel. For BITS_PER_COLOUR_CHANNEL=1, the adapter
can display 8 colours.
           * In general, the adapter is able to use 2^(3*BITS_PER_COLOUR_CHANNEL ) colours.
The number of colours is
           * limited by the screen resolution and the amount of on-chip memory available on the target
device.
           */

          parameter MONOCHROME = "FALSE";
          /* Set this parameter to "TRUE" if you only wish to use black and white colours. Doing so
will reduce
           * the amount of memory you will use by a factor of 3. */

          parameter RESOLUTION = "320x240";
          /* Set this parameter to "160x120" or "320x240". It will cause the VGA adapter to draw each
dot on
           * the screen by using a block of 4x4 pixels ("160x120" resolution) or 2x2 pixels ("320x240"
resolution).
           * It effectively reduces the screen resolution to an integer fraction of 640x480. It was
necessary
           * to reduce the resolution for the Video Memory to fit within the on-chip memory limits.
           */

          parameter BACKGROUND_IMAGE = "background.mif";
          /* The initial screen displayed when the circuit is first programmed onto the DE2 board can
be
           * defined useing an MIF file. The file contains the initial colour for each pixel on the screen
           * and is placed in the Video Memory (VideoMemory module) upon programming. Note that
resetting the
           * VGA Adapter will not cause the Video Memory to revert to the specified image. */



/*****************************************************************************/
          /* Declare inputs and outputs.                                    */

/*****************************************************************************/
          input resetn;
          input clock;
```

```
	/* The colour input can be either 1 bit or 3*BITS_PER_COLOUR_CHANNEL bits wide,
depending on
	 * the setting of the MONOCHROME parameter.
	 */
	input [((MONOCHROME == "TRUE") ? (0) : (BITS_PER_COLOUR_CHANNEL*3-1)):0]
colour;

	/* Specify the number of bits required to represent an (X,Y) coordinate on the screen for
	 * a given resolution.
	 */
	input [((RESOLUTION == "320x240") ? (8) : (7)):0] x;
	input [((RESOLUTION == "320x240") ? (7) : (6)):0] y;

	/* When plot is high then at the next positive edge of the clock the pixel at (x,y) will change
to
	 * a new colour, defined by the value of the colour input.
	 */
	input plot;

	/* These outputs drive the VGA display. The VGA_CLK is also used to clock the FSM
responsible for
	 * controlling the data transferred to the DAC driving the monitor. */
	output [7:0] VGA_R;
	output [7:0] VGA_G;
	output [7:0] VGA_B;
	output VGA_HS;
	output VGA_VS;
	output VGA_BLANK;
	output VGA_SYNC;
	output VGA_CLK;


/*****************************************************************************/
	/* Declare local signals here.                              */

/*****************************************************************************/

	wire valid_160x120;
	wire valid_320x240;
	/* Set to 1 if the specified coordinates are in a valid range for a given resolution.*/

	wire writeEn;
	/* This is a local signal that allows the Video Memory contents to be changed.
	 * It depends on the screen resolution, the values of X and Y inputs, as well as
	 * the state of the plot signal.
	 */
	wire [((MONOCHROME == "TRUE") ? (0) : (BITS_PER_COLOUR_CHANNEL*3-1)):0]
to_ctrl_colour;
	/* Pixel colour read by the VGA controller */

	wire [((RESOLUTION == "320x240") ? (16) : (14)):0] user_to_video_memory_addr;
	/* This bus specifies the address in memory the user must write
	 * data to in order for the pixel intended to appear at location (X,Y) to be displayed
	 * at the correct location on the screen.
	 */

	wire [((RESOLUTION == "320x240") ? (16) : (14)):0] controller_to_video_memory_addr;
	/* This bus specifies the address in memory the vga controller must read data from
	 * in order to determine the colour of a pixel located at coordinate (X,Y) of the screen.
	 */

	wire clock_25;
	/* 25MHz clock generated by dividing the input clock frequency by 2. */

	wire vcc, gnd;


/*****************************************************************************/
	/* Instances of modules for the VGA adapter.                    */

/*****************************************************************************/
	assign vcc = 1'b1;
	assign gnd = 1'b0;

	vga_address_translator user_input_translator(
					.x(x), .y(y), .mem_address(user_to_video_memory_addr) );
		defparam user_input_translator.RESOLUTION = RESOLUTION;
	/* Convert user coordinates into a memory address. */

	assign valid_160x120 = (({1'b0, x} >= 0) & ({1'b0, x} < 160) & ({1'b0, y} >= 0) & ({1'b0, y}
< 120)) & (RESOLUTION == "160x120");
	assign valid_320x240 = (({1'b0, x} >= 0) & ({1'b0, x} < 320) & ({1'b0, y} >= 0) & ({1'b0, y}
< 240)) & (RESOLUTION == "320x240");
	assign writeEn = (plot) & (valid_160x120 | valid_320x240);
	/* Allow the user to plot a pixel if and only if the (X,Y) coordinates supplied are in a valid
range. */

	/* Create video memory. */
	altsyncram	VideoMemory (
				.wren_a (writeEn),
				.wren_b (gnd),
                              .clock0 (clock), // write clock
                              .clock1 (clock_25), // read clock
                              .clocken0 (vcc), // write enable clock
                              .clocken1 (vcc), // read enable clock
                              .address_a (user_to_video_memory_addr),
                              .address_b (controller_to_video_memory_addr),
                              .data_a (colour), // data in
                              .q_b (to_ctrl_colour)     // data out
                              );
     defparam
          VideoMemory.WIDTH_A = ((MONOCHROME == "FALSE") ?
(BITS_PER_COLOUR_CHANNEL*3) : 1),
          VideoMemory.WIDTH_B = ((MONOCHROME == "FALSE") ?
(BITS_PER_COLOUR_CHANNEL*3) : 1),
          VideoMemory.INTENDED_DEVICE_FAMILY = "Cyclone II",
          VideoMemory.OPERATION_MODE = "DUAL_PORT",
          VideoMemory.WIDTHAD_A = ((RESOLUTION == "320x240") ? (17) : (15)),
          VideoMemory.NUMWORDS_A = ((RESOLUTION == "320x240") ? (76800) :
(19200)),
          VideoMemory.WIDTHAD_B = ((RESOLUTION == "320x240") ? (17) : (15)),
          VideoMemory.NUMWORDS_B = ((RESOLUTION == "320x240") ? (76800) :
(19200)),
          VideoMemory.OUTDATA_REG_B = "CLOCK1",
          VideoMemory.ADDRESS_REG_B = "CLOCK1",
          VideoMemory.CLOCK_ENABLE_INPUT_A = "BYPASS",
          VideoMemory.CLOCK_ENABLE_INPUT_B = "BYPASS",
          VideoMemory.CLOCK_ENABLE_OUTPUT_B = "BYPASS",
          VideoMemory.POWER_UP_UNINITIALIZED = "FALSE",
          VideoMemory.INIT_FILE = BACKGROUND_IMAGE;

     vga_pll mypll(clock, clock_25);
     /* This module generates a clock with half the frequency of the input clock.
      * For the VGA adapter to operate correctly the clock signal 'clock' must be
      * a 50MHz clock. The derived clock, which will then operate at 25MHz, is
      * required to set the monitor into the 640x480@60Hz display mode (also known as
      * the VGA mode).
      */

     wire [9:0] r;
     wire [9:0] g;
     wire [9:0] b;

     /* Assign the MSBs from the controller to the VGA signals */

     assign VGA_R = r[9:2];
     assign VGA_G = g[9:2];
```

```
	assign VGA_B = b[9:2];

	vga_controller controller(
			.vga_clock(clock_25),
			.resetn(resetn),
			.pixel_colour(to_ctrl_colour),
			.memory_address(controller_to_video_memory_addr),
			.VGA_R(r),
			.VGA_G(g),
			.VGA_B(b),
			.VGA_HS(VGA_HS),
			.VGA_VS(VGA_VS),
			.VGA_BLANK(VGA_BLANK),
			.VGA_SYNC(VGA_SYNC),
			.VGA_CLK(VGA_CLK)
		);
		defparam controller.BITS_PER_COLOUR_CHANNEL  =
BITS_PER_COLOUR_CHANNEL ;
		defparam controller.MONOCHROME = MONOCHROME;
		defparam controller.RESOLUTION = RESOLUTION;

endmodule

vga_address_translator
/* This module converts a user specified coordinates into a memory address.
 * The output of the module depends on the resolution set by the user.
 */
module vga_address_translator(x, y, mem_address);

	parameter RESOLUTION = "320x240";
	/* Set this parameter to "160x120" or "320x240". It will cause the VGA adapter to draw each
dot on
	 * the screen by using a block of 4x4 pixels ("160x120" resolution) or 2x2 pixels ("320x240"
resolution).
	 * It effectively reduces the screen resolution to an integer fraction of 640x480. It was
necessary
	 * to reduce the resolution for the Video Memory to fit within the on-chip memory limits.
	 */

	input [((RESOLUTION == "320x240") ? (8) : (7)):0] x;
	input [((RESOLUTION == "320x240") ? (7) : (6)):0] y;
	output reg [((RESOLUTION == "320x240") ? (16) : (14)):0] mem_address;

	/* The basic formula is address = y*WIDTH + x;
	 * For 320x240 resolution we can write 320 as (256 + 64). Memory address becomes
	 * (y*256) + (y*64) + x;
	 * This simplifies multiplication a simple shift and add operation.
	 * A leading 0 bit is added to each operand to ensure that they are treated as unsigned
	 * inputs. By default the use a '+' operator will generate a signed adder.
	 * Similarly, for 160x120 resolution we write 160 as 128+32.
	 */
	wire [16:0] res_320x240 = ({1'b0, y, 8'd0} + {1'b0, y, 6'd0} + {1'b0, x});
	wire [15:0] res_160x120 = ({1'b0, y, 7'd0} + {1'b0, y, 5'd0} + {1'b0, x});

	always @(*)
	begin
		if (RESOLUTION == "320x240")
			mem_address = res_320x240;
		else
			mem_address = res_160x120[14:0];
	end
endmodule



vga_controller
/* This module implements the VGA controller. It assumes a 25MHz clock is supplied as input.
 *
 * General approach:
 * Go through each line of the screen and read the colour each pixel on that line should have from
 * the Video memory. To do that for each (x,y) pixel on the screen convert (x,y) coordinate to
 * a memory_address at which the pixel colour is stored in Video memory. Once the pixel colour is
 * read from video memory its brightness is first increased before it is forwarded to the VGA DAC.
 */
module vga_controller(	vga_clock, resetn, pixel_colour, memory_address,
		VGA_R, VGA_G, VGA_B,
		VGA_HS, VGA_VS, VGA_BLANK,
		VGA_SYNC, VGA_CLK);

	/* Screen resolution and colour depth parameters. */

	parameter BITS_PER_COLOUR_CHANNEL = 1;
	/* The number of bits per colour channel used to represent the colour of each pixel. A value
	 * of 1 means that Red, Green and Blue colour channels will use 1 bit each to represent the
intensity
	 * of the respective colour channel. For BITS_PER_COLOUR_CHANNEL=1, the adapter
can display 8 colours.
	 * In general, the adapter is able to use 2^(3*BITS_PER_COLOUR_CHANNEL) colours.
The number of colours is
	 * limited by the screen resolution and the amount of on-chip memory available on the target
device.
	 */
	parameter MONOCHROME = "FALSE";
	/* Set this parameter to "TRUE" if you only wish to use black and white colours. Doing so 
will reduce
	 * the amount of memory you will use by a factor of 3. */

	parameter RESOLUTION = "320x240";
	/* Set this parameter to "160x120" or "320x240". It will cause the VGA adapter to draw each 
dot on
	 * the screen by using a block of 4x4 pixels ("160x120" resolution) or 2x2 pixels ("320x240" 
resolution).
	 * It effectively reduces the screen resolution to an integer fraction of 640x480. It was 
necessary
	 * to reduce the resolution for the Video Memory to fit within the on-chip memory limits.
	 */

	//--- Timing parameters.
	/* Recall that the VGA specification requires a few more rows and columns are drawn
	 * when refreshing the screen than are actually present on the screen. This is necessary to
	 * generate the vertical and the horizontal syncronization signals. If you wish to use a
	 * display mode other than 640x480 you will need to modify the parameters below as well
	 * as change the frequency of the clock driving the monitor (VGA_CLK).
	 */
	parameter C_VERT_NUM_PIXELS  = 10'd480;
	parameter C_VERT_SYNC_START  = 10'd493;
	parameter C_VERT_SYNC_END    = 10'd494; //(C_VERT_SYNC_START + 2 - 1); 
	parameter C_VERT_TOTAL_COUNT = 10'd525;

	parameter C_HORZ_NUM_PIXELS  = 10'd640;
	parameter C_HORZ_SYNC_START  = 10'd659;
	parameter C_HORZ_SYNC_END    = 10'd754; //(C_HORZ_SYNC_START + 96 - 1); 
	parameter C_HORZ_TOTAL_COUNT = 10'd800;


/*****************************************************************************/
	/* Declare inputs and outputs.                                             */

/*****************************************************************************/

	input vga_clock, resetn;
	input [((MONOCHROME == "TRUE") ? (0) : (BITS_PER_COLOUR_CHANNEL*3-1)):0] 
pixel_colour;
	output [((RESOLUTION == "320x240") ? (16) : (14)):0] memory_address;
	output reg [9:0] VGA_R;
	output reg [9:0] VGA_G;
	output reg [9:0] VGA_B;
	output reg VGA_HS;
	output reg VGA_VS;
	output reg VGA_BLANK;
	output VGA_SYNC, VGA_CLK;


/*****************************************************************************/
	/* Local Signals.                                   */

/*****************************************************************************/

	reg VGA_HS1;
	reg VGA_VS1;
	reg VGA_BLANK1;
	reg [9:0] xCounter, yCounter;
	wire xCounter_clear;
	wire yCounter_clear;
	wire vcc;

	reg [((RESOLUTION == "320x240") ? (8) : (7)):0] x;
	reg [((RESOLUTION == "320x240") ? (7) : (6)):0] y;
	/* Inputs to the converter. */


/*****************************************************************************/
	/* Controller implementation.                               */

/*****************************************************************************/

	assign vcc =1'b1;

	/* A counter to scan through a horizontal line. */
	always @(posedge vga_clock or negedge resetn)
	begin
		if (!resetn)
			xCounter <= 10'd0;
		else if (xCounter_clear)
			xCounter <= 10'd0;
		else
		begin
			xCounter <= xCounter + 1'b1;
		end
	end
	assign xCounter_clear = (xCounter == (C_HORZ_TOTAL_COUNT-1));

	/* A counter to scan vertically, indicating the row currently being drawn. */
	always @(posedge vga_clock or negedge resetn)
	begin
		if (!resetn)
			yCounter <= 10'd0;
		else if (xCounter_clear && yCounter_clear)
			yCounter <= 10'd0;
		else if (xCounter_clear)		//Increment when x counter resets
			yCounter <= yCounter + 1'b1;
	end
	assign yCounter_clear = (yCounter == (C_VERT_TOTAL_COUNT-1));

	/* Convert the xCounter/yCounter location from screen pixels (640x480) to our
	 * local dots (320x240 or 160x120). Here we effectively divide x/y coordinate by 2 or 4,
	 * depending on the resolution. */
	always @(*)
	begin
		if (RESOLUTION == "320x240")
		begin
			x = xCounter[9:1];
			y = yCounter[8:1];
		end
		else
		begin
			x = xCounter[9:2];
			y = yCounter[8:2];
		end
	end

	/* Change the (x,y) coordinate into a memory address. */
	vga_address_translator controller_translator(
					.x(x), .y(y), .mem_address(memory_address) );
		defparam controller_translator.RESOLUTION = RESOLUTION;


	/* Generate the vertical and horizontal synchronization pulses. */
	always @(posedge vga_clock)
	begin
		//- Sync Generator (ACTIVE LOW)
		VGA_HS1 <= ~((xCounter >= C_HORZ_SYNC_START) && (xCounter <=
C_HORZ_SYNC_END));
		VGA_VS1 <= ~((yCounter >= C_VERT_SYNC_START) && (yCounter <=
C_VERT_SYNC_END));

		//- Current X and Y is valid pixel range
		VGA_BLANK1 <= ((xCounter < C_HORZ_NUM_PIXELS) && (yCounter <
C_VERT_NUM_PIXELS));
```

```
		//- Add 1 cycle delay
		VGA_HS <= VGA_HS1;
		VGA_VS <= VGA_VS1;
		VGA_BLANK <= VGA_BLANK1;
	end

	/* VGA sync should be 1 at all times. */
	assign VGA_SYNC = vcc;

	/* Generate the VGA clock signal. */
	assign VGA_CLK = vga_clock;

	/* Brighten the colour output. */
	// The colour input is first processed to brighten the image a little. Setting the top
	// bits to correspond to the R,G,B colour makes the image a bit dull. To brighten the image,
	// each bit of the colour is replicated through the 10 DAC colour input bits. For example,
	// when BITS_PER_COLOUR_CHANNEL is 2 and the red component is set to 2'b10, then
the
	// VGA_R input to the DAC will be set to 10'b1010101010.

	integer index;
	integer sub_index;

	always @(pixel_colour)
	begin
		VGA_R <= 'b0;
		VGA_G <= 'b0;
		VGA_B <= 'b0;
		if (MONOCHROME == "FALSE")
		begin
			for (index = 10-BITS_PER_COLOUR_CHANNEL; index >= 0; index =
index - BITS_PER_COLOUR_CHANNEL)
			begin
				for (sub_index = BITS_PER_COLOUR_CHANNEL - 1; sub_index
>= 0; sub_index = sub_index - 1)
				begin
					VGA_R[sub_index+index] <= pixel_colour[sub_index +
BITS_PER_COLOUR_CHANNEL*2];
					VGA_G[sub_index+index] <= pixel_colour[sub_index +
BITS_PER_COLOUR_CHANNEL];
					VGA_B[sub_index+index] <= pixel_colour[sub_index];
				end
			end
		end
		else
		begin
```

```
                        for (index = 0; index < 10; index = index + 1)
                        begin
                                VGA_R[index] <= pixel_colour[0:0];
                                VGA_G[index] <= pixel_colour[0:0];
                                VGA_B[index] <= pixel_colour[0:0];
                        end
                end
        end

endmodule
```

Vga_pll

```
// megafunction wizard: %ALTPLL%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altpll

// ============================================================
// File Name: VgaPll.v
// Megafunction Name(s):
//                      altpll
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 5.0 Build 168 06/22/2005 SP 1 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2005 Altera Corporation
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, Altera MegaCore Function License
//Agreement, or other applicable license agreement, including,
//without limitation, that your use is for the sole purpose of
//programming logic devices manufactured by Altera and sold by
//Altera or its authorized distributors.  Please refer to the
//applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
```

```
// synopsys translate_on
module vga_pll (
        clock_in,
        clock_out);

        input     clock_in;
        output    clock_out;

        wire [5:0] clock_output_bus;
        wire [1:0] clock_input_bus;
        wire gnd;

        assign gnd = 1'b0;
        assign clock_input_bus = { gnd, clock_in };

        altpll    altpll_component (
                                .inclk (clock_input_bus),
                                .clk (clock_output_bus)
                                );
        defparam
                altpll_component.operation_mode = "NORMAL",
                altpll_component.intended_device_family = "Cyclone II",
                altpll_component.lpm_type = "altpll",
                altpll_component.pll_type = "FAST",
                /* Specify the input clock to be a 50MHz clock. A 50 MHz clock is present
                 * on PIN_N2 on the DE2 board. We need to specify the input clock frequency
                 * in order to set up the PLL correctly. To do this we must put the input clock
                 * period measured in picoseconds in the inclk0_input_frequency parameter.
                 * 1/(20000 ps) = 0.5 * 10^(5) Hz = 50 * 10^(6) Hz = 50 MHz. */
                altpll_component.inclk0_input_frequency = 20000,
                altpll_component.primary_clock = "INCLK0",
                /* Specify output clock parameters. The output clock should have a
                 * frequency of 25 MHz, with 50% duty cycle. */
                altpll_component.compensate_clock = "CLK0",
                altpll_component.clk0_phase_shift = "0",
                altpll_component.clk0_divide_by = 2,
                altpll_component.clk0_multiply_by = 1,
                altpll_component.clk0_duty_cycle = 50;

        assign clock_out = clock_output_bus[0];

endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: MIRROR_CLK0 STRING "0"
// Retrieval info: PRIVATE: PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: OUTPUT_FREQ_UNIT0 STRING "MHz"
// Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_COMBO STRING "MHz"
// Retrieval info: PRIVATE: SPREAD_USE STRING "0"
// Retrieval info: PRIVATE: SPREAD_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: GLOCKED_COUNTER_EDIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: GLOCK_COUNTER_EDIT NUMERIC "1048575"
// Retrieval info: PRIVATE: SRC_SYNCH_COMP_RADIO STRING "0"
// Retrieval info: PRIVATE: DUTY_CYCLE0 STRING "50.00000000"
// Retrieval info: PRIVATE: PHASE_SHIFT0 STRING "0.00000000"
// Retrieval info: PRIVATE: MULT_FACTOR0 NUMERIC "1"
// Retrieval info: PRIVATE: OUTPUT_FREQ_MODE0 STRING "1"
// Retrieval info: PRIVATE: SPREAD_PERCENT STRING "0.500"
// Retrieval info: PRIVATE: LOCKED_OUTPUT_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_ARESET_CHECK STRING "0"
// Retrieval info: PRIVATE: STICKY_CLK0 STRING "1"
// Retrieval info: PRIVATE: BANDWIDTH STRING "1.000"
// Retrieval info: PRIVATE: BANDWIDTH_USE_CUSTOM STRING "0"
// Retrieval info: PRIVATE: DEVICE_SPEED_GRADE STRING "Any"
// Retrieval info: PRIVATE: SPREAD_FREQ STRING "50.000"
// Retrieval info: PRIVATE: BANDWIDTH_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: LONG_SCAN_RADIO STRING "1"
// Retrieval info: PRIVATE: PLL_ENHPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE_DIRTY NUMERIC "0"
// Retrieval info: PRIVATE: USE_CLK0 STRING "1"
// Retrieval info: PRIVATE: INCLK1_FREQ_EDIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: SCAN_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: ZERO_DELAY_RADIO STRING "0"
// Retrieval info: PRIVATE: PLL_PFDENA_CHECK STRING "0"
// Retrieval info: PRIVATE: CREATE_CLKBAD_CHECK STRING "0"
// Retrieval info: PRIVATE: INCLK1_FREQ_EDIT STRING "50.000"
// Retrieval info: PRIVATE: CUR_DEDICATED_CLK STRING "c0"
// Retrieval info: PRIVATE: PLL_FASTPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: ACTIVECLK_CHECK STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH_FREQ_UNIT STRING "MHz"
// Retrieval info: PRIVATE: INCLK0_FREQ_UNIT_COMBO STRING "MHz"
// Retrieval info: PRIVATE: GLOCKED_MODE_CHECK STRING "0"
// Retrieval info: PRIVATE: NORMAL_MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: CUR_FBIN_CLK STRING "e0"
// Retrieval info: PRIVATE: DIV_FACTOR0 NUMERIC "1"
// Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: HAS_MANUAL_SWITCHOVER STRING "1"
// Retrieval info: PRIVATE: EXT_FEEDBACK_RADIO STRING "0"
// Retrieval info: PRIVATE: PLL_AUTOPLL_CHECK NUMERIC "1"
// Retrieval info: PRIVATE: CLKLOSS_CHECK STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH_USE_AUTO STRING "1"
// Retrieval info: PRIVATE: SHORT_SCAN_RADIO STRING "0"
// Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE STRING "Not Available"
// Retrieval info: PRIVATE: CLKSWITCH_CHECK STRING "1"
// Retrieval info: PRIVATE: SPREAD_FREQ_UNIT STRING "KHz"
// Retrieval info: PRIVATE: PLL_ENA_CHECK STRING "0"
// Retrieval info: PRIVATE: INCLK0_FREQ_EDIT STRING "50.000"
// Retrieval info: PRIVATE: CNX_NO_COMPENSATE_RADIO STRING "0"
// Retrieval info: PRIVATE: INT_FEEDBACK__MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: OUTPUT_FREQ0 STRING "25.000"
// Retrieval info: PRIVATE: PRIMARY_CLK_COMBO STRING "inclk0"
// Retrieval info: PRIVATE: CREATE_INCLK1_CHECK STRING "0"
// Retrieval info: PRIVATE: SACN_INPUTS_CHECK STRING "0"
// Retrieval info: PRIVATE: DEV_FAMILY STRING "Cyclone II"
// Retrieval info: PRIVATE: SWITCHOVER_COUNT_EDIT NUMERIC "1"
// Retrieval info: PRIVATE: SWITCHOVER_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: BANDWIDTH_PRESET STRING "Low"
// Retrieval info: PRIVATE: GLOCKED_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: USE_CLKENA0 STRING "0"
// Retrieval info: PRIVATE: LVDS_PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: CLKBAD_SWITCHOVER_CHECK STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH_USE_PRESET STRING "0"
// Retrieval info: PRIVATE: PLL_LVDS_PLL_CHECK NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLK0_DUTY_CYCLE NUMERIC "50"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altpll"
// Retrieval info: CONSTANT: CLK0_MULTIPLY_BY NUMERIC "1"
// Retrieval info: CONSTANT: INCLK0_INPUT_FREQUENCY NUMERIC "20000"
// Retrieval info: CONSTANT: CLK0_DIVIDE_BY NUMERIC "2"
// Retrieval info: CONSTANT: PLL_TYPE STRING "FAST"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "NORMAL"
// Retrieval info: CONSTANT: COMPENSATE_CLOCK STRING "CLK0"
// Retrieval info: CONSTANT: CLK0_PHASE_SHIFT STRING "0"
// Retrieval info: USED_PORT: c0 0 0 0 0 OUTPUT VCC "c0"
// Retrieval info: USED_PORT: @clk 0 0 6 0 OUTPUT VCC "@clk[5..0]"
// Retrieval info: USED_PORT: inclk0 0 0 0 0 INPUT GND "inclk0"
// Retrieval info: USED_PORT: @extclk 0 0 4 0 OUTPUT VCC "@extclk[3..0]"
// Retrieval info: CONNECT: @inclk 0 0 1 0 inclk0 0 0 0 0
// Retrieval info: CONNECT: c0 0 0 0 0 @clk 0 0 1 0
// Retrieval info: CONNECT: @inclk 0 0 1 1 GND 0 0 0 0
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll.v TRUE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll.inc FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll.cmp FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll.bsf FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll_inst.v FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll_bb.v FALSE FALSE
```

B.4 Audio_controller(Eecg.toronto.edu, 2019)

B.4.1 Altera_UP_Audio_Bit_Counter

```
/*****************************************************************************
 *                                                                           *
 * Module:       Altera_UP_Audio_Bit_Counter                                 *
 * Description:                                                              *
 *      This module counts which bits for serial audio transfers. The module *
 *   assume that the data format is I2S, as it is described in the audio     *
 *   chip's datasheet.                                                       *
 *                                                                           *
 *****************************************************************************/

module Altera_UP_Audio_Bit_Counter (
	// Inputs
	clk,
	reset,

	bit_clk_rising_edge,
	bit_clk_falling_edge,
	left_right_clk_rising_edge,
	left_right_clk_falling_edge,

	// Bidirectionals

	// Outputs
	counting
);

/*****************************************************************************
 *                           Parameter Declarations                          *
 *****************************************************************************/

parameter BIT_COUNTER_INIT	= 5'd31;

/*****************************************************************************
 *                             Port Declarations                             *
 *****************************************************************************/

// Inputs
input				clk;
```

```
input                       reset;

input                       bit_clk_rising_edge;
input                       bit_clk_falling_edge;
input                       left_right_clk_rising_edge;
input                       left_right_clk_falling_edge;

// Bidirectionals

// Outputs
output  reg                 counting;

/*****************************************************************************
 *                    Constant Declarations                    *
 *****************************************************************************/


/*****************************************************************************
 *              Internal wires and registers Declarations              *
 *****************************************************************************/

// Internal Wires
wire                        reset_bit_counter;

// Internal Registers
reg                 [4:0]   bit_counter;

// State Machine Registers


/*****************************************************************************
 *                   Finite State Machine(s)                   *
 *****************************************************************************/


/*****************************************************************************
 *                      Sequential logic                      *
 *****************************************************************************/

always @(posedge clk)
begin
	if (reset == 1'b1)
		bit_counter <= 5'h00;
	else if (reset_bit_counter == 1'b1)
		bit_counter <= BIT_COUNTER_INIT;
	else if ((bit_clk_falling_edge == 1'b1) && (bit_counter != 5'h00))
```

```
			bit_counter <= bit_counter - 5'h01;
end

always @(posedge clk)
begin
	if (reset == 1'b1)
		counting <= 1'b0;
	else if (reset_bit_counter == 1'b1)
		counting <= 1'b1;
	else if ((bit_clk_falling_edge == 1'b1) && (bit_counter == 5'h00))
		counting <= 1'b0;
end

/*****************************************************************************
 *                         Combinational logic                               *
 *****************************************************************************/

assign reset_bit_counter = left_right_clk_rising_edge |
							left_right_clk_falling_edge;

/*****************************************************************************
 *                         Internal Modules                                  *
 *****************************************************************************/

endmodule
```

B.4.2 Altera_UP_Audio_In_Deserializer

```
/*****************************************************************************
 *                                                                           *
 * Module:       Altera_UP_Audio_In_Deserializer                             *
 * Description:                                                              *
 *      This module read data from the Audio ADC on the Altera DE2 board.    *
 *                                                                           *
 *****************************************************************************/

module Altera_UP_Audio_In_Deserializer (
	// Inputs
	clk,
	reset,

	bit_clk_rising_edge,
	bit_clk_falling_edge,
	left_right_clk_rising_edge,
	left_right_clk_falling_edge,
	done_channel_sync,

	serial_audio_in_data,

	read_left_audio_data_en,
	read_right_audio_data_en,

	// Bidirectionals

	// Outputs
	left_audio_fifo_read_space,
	right_audio_fifo_read_space,

	left_channel_data,
	right_channel_data
);

/*****************************************************************************
 *                           Parameter Declarations                          *
 *****************************************************************************/

parameter AUDIO_DATA_WIDTH	= 32;
parameter BIT_COUNTER_INIT	= 5'd31;

/*****************************************************************************
 *                             Port Declarations                             *
 *****************************************************************************/
// Inputs
input				clk;
input				reset;

input				bit_clk_rising_edge;
input				bit_clk_falling_edge;
input				left_right_clk_rising_edge;
input				left_right_clk_falling_edge;

input				done_channel_sync;

input				serial_audio_in_data;

input				read_left_audio_data_en;
input				read_right_audio_data_en;

// Bidirectionals

// Outputs
```

```
output   reg       [7:0]     left_audio_fifo_read_space;
output   reg       [7:0]     right_audio_fifo_read_space;

output             [AUDIO_DATA_WIDTH:1]     left_channel_data;
output             [AUDIO_DATA_WIDTH:1]     right_channel_data;

/*****************************************************************************
 *              Internal wires and registers Declarations                    *
 *****************************************************************************/
// Internal Wires
wire                              valid_audio_input;

wire                              left_channel_fifo_is_empty;
wire                              right_channel_fifo_is_empty;

wire                              left_channel_fifo_is_full;
wire                              right_channel_fifo_is_full;

wire               [6:0]     left_channel_fifo_used;
wire               [6:0]     right_channel_fifo_used;

// Internal Registers
reg                          [AUDIO_DATA_WIDTH:1]     data_in_shift_reg;

// State Machine Registers


/*****************************************************************************
 *                    Finite State Machine(s)                                *
 *****************************************************************************/


/*****************************************************************************
 *                        Sequential logic                                   *
 *****************************************************************************/

always @(posedge clk)
begin
	if (reset == 1'b1)
		left_audio_fifo_read_space				<= 8'h00;
	else
	begin
		left_audio_fifo_read_space[7]		<= left_channel_fifo_is_full;
		left_audio_fifo_read_space[6:0]		<= left_channel_fifo_used;
	end
end
```

```
always @(posedge clk)
begin
	if (reset == 1'b1)
		right_audio_fifo_read_space			<= 8'h00;
	else
	begin
		right_audio_fifo_read_space[7]		<= right_channel_fifo_is_full;
		right_audio_fifo_read_space[6:0]		<= right_channel_fifo_used;
	end
end



always @(posedge clk)
begin
	if (reset == 1'b1)
		data_in_shift_reg		<= {AUDIO_DATA_WIDTH{1'b0}};
	else if (bit_clk_rising_edge & valid_audio_input)
		data_in_shift_reg		<=
			{data_in_shift_reg[(AUDIO_DATA_WIDTH - 1):1],
			 serial_audio_in_data};
end

/*****************************************************************************
 *                        Combinational logic                        *
 *****************************************************************************/


/*****************************************************************************
 *                         Internal Modules                         *
 *****************************************************************************/

Altera_UP_Audio_Bit_Counter Audio_Out_Bit_Counter (
	// Inputs
	.clk							(clk),
	.reset							(reset),

	.bit_clk_rising_edge			(bit_clk_rising_edge),
	.bit_clk_falling_edge			(bit_clk_falling_edge),
	.left_right_clk_rising_edge		(left_right_clk_rising_edge),
	.left_right_clk_falling_edge	(left_right_clk_falling_edge),

	// Bidirectionals
	// Outputs
	.counting						(valid_audio_input)
);
defparam
	Audio_Out_Bit_Counter.BIT_COUNTER_INIT= BIT_COUNTER_INIT;

Altera_UP_SYNC_FIFO Audio_In_Left_Channel_FIFO(
	// Inputs
	.clk			(clk),
	.reset			(reset),

	.write_en		(left_right_clk_falling_edge & ~left_channel_fifo_is_full &
done_channel_sync),
	.write_data		(data_in_shift_reg),

	.read_en		(read_left_audio_data_en & ~left_channel_fifo_is_empty),

	// Bidirectionals

	// Outputs
	.fifo_is_empty	(left_channel_fifo_is_empty),
	.fifo_is_full	(left_channel_fifo_is_full),
	.words_used		(left_channel_fifo_used),

	.read_data		(left_channel_data)
);
defparam
	Audio_In_Left_Channel_FIFO.DATA_WIDTH = AUDIO_DATA_WIDTH,
	Audio_In_Left_Channel_FIFO.DATA_DEPTH = 128,
	Audio_In_Left_Channel_FIFO.ADDR_WIDTH = 7;

Altera_UP_SYNC_FIFO Audio_In_Right_Channel_FIFO(
	// Inputs
	.clk			(clk),
	.reset			(reset),

	.write_en		(left_right_clk_rising_edge & ~right_channel_fifo_is_full &
done_channel_sync),
	.write_data		(data_in_shift_reg),

	.read_en		(read_right_audio_data_en & ~right_channel_fifo_is_empty),

	// Bidirectionals

	// Outputs
	.fifo_is_empty	(right_channel_fifo_is_empty),
	.fifo_is_full		(right_channel_fifo_is_full),
	.words_used		(right_channel_fifo_used),

	.read_data		(right_channel_data)
);
defparam
	Audio_In_Right_Channel_FIFO.DATA_WIDTH		= AUDIO_DATA_WIDTH,
	Audio_In_Right_Channel_FIFO.DATA_DEPTH		= 128,
	Audio_In_Right_Channel_FIFO.ADDR_WIDTH		= 7;

endmodule
```

B.4.3 Altera_UP_Audio_Out_Serializer

```
/*****************************************************************************
 *                                                                           *
 * Module:       Altera_UP_Audio_Out_Serializer                              *
 * Description:                                                              *
 *      This module writes data to the Audio DAC on the Altera DE2 board.    *
 *                                                                           *
 *****************************************************************************/

module Altera_UP_Audio_Out_Serializer (
	// Inputs
	clk,
	reset,

	bit_clk_rising_edge,
	bit_clk_falling_edge,
	left_right_clk_rising_edge,
	left_right_clk_falling_edge,

	left_channel_data,
	left_channel_data_en,

	right_channel_data,
	right_channel_data_en,

	// Bidirectionals

	// Outputs
	left_channel_fifo_write_space,
	right_channel_fifo_write_space,

	serial_audio_out_data
);
```

```
/*****************************************************************************
 *                        Parameter Declarations                         *
 *****************************************************************************/

parameter AUDIO_DATA_WIDTH      = 32;

/*****************************************************************************
 *                        Port Declarations                           *
 *****************************************************************************/
// Inputs
input                           clk;
input                           reset;

input                           bit_clk_rising_edge;
input                           bit_clk_falling_edge;
input                           left_right_clk_rising_edge;
input                           left_right_clk_falling_edge;

input           [AUDIO_DATA_WIDTH:1]            left_channel_data;
input                           left_channel_data_en;

input           [AUDIO_DATA_WIDTH:1]            right_channel_data;
input                           right_channel_data_en;

// Bidirectionals

// Outputs
output  reg     [7:0]   left_channel_fifo_write_space;
output  reg     [7:0]   right_channel_fifo_write_space;

output  reg                     serial_audio_out_data;


/*****************************************************************************
 *              Internal wires and registers Declarations              *
 *****************************************************************************/

// Internal Wires
wire                            read_left_channel;
wire                            read_right_channel;

wire                            left_channel_fifo_is_empty;
wire                            right_channel_fifo_is_empty;

wire                            left_channel_fifo_is_full;
wire                            right_channel_fifo_is_full;

wire            [6:0]    left_channel_fifo_used;
wire            [6:0]    right_channel_fifo_used;

wire            [AUDIO_DATA_WIDTH:1]            left_channel_from_fifo;
wire            [AUDIO_DATA_WIDTH:1]            right_channel_from_fifo;

// Internal Registers
reg                                     left_channel_was_read;
reg                     [AUDIO_DATA_WIDTH:1]    data_out_shift_reg;

// State Machine Registers

/*****************************************************************************
 *                      Finite State Machine(s)                              *
 *****************************************************************************/


/*****************************************************************************
 *                         Sequential logic                              *
 *****************************************************************************/

always @(posedge clk)
begin
        if (reset == 1'b1)
                left_channel_fifo_write_space <= 8'h00;
        else
                left_channel_fifo_write_space <= 8'h80 - 
{left_channel_fifo_is_full,left_channel_fifo_used};
end

always @(posedge clk)
begin
        if (reset == 1'b1)
                right_channel_fifo_write_space <= 8'h00;
        else
                right_channel_fifo_write_space <= 8'h80 - 
{right_channel_fifo_is_full,right_channel_fifo_used};
end


always @(posedge clk)
begin
        if (reset == 1'b1)
                serial_audio_out_data <= 1'b0;
```

```
		else
			serial_audio_out_data <= data_out_shift_reg[AUDIO_DATA_WIDTH];
end


always @(posedge clk)
begin
	if (reset == 1'b1)
		left_channel_was_read <= 1'b0;
	else if (read_left_channel)
		left_channel_was_read <=1'b1;
	else if (read_right_channel)
		left_channel_was_read <=1'b0;
end


always @(posedge clk)
begin
	if (reset == 1'b1)
		data_out_shift_reg	<= {AUDIO_DATA_WIDTH{1'b0}};
	else if (read_left_channel)
		data_out_shift_reg	<= left_channel_from_fifo;
	else if (read_right_channel)
		data_out_shift_reg	<= right_channel_from_fifo;
	else if (left_right_clk_rising_edge | left_right_clk_falling_edge)
		data_out_shift_reg	<= {AUDIO_DATA_WIDTH{1'b0}};
	else if (bit_clk_falling_edge)
		data_out_shift_reg	<=
			{data_out_shift_reg[(AUDIO_DATA_WIDTH - 1):1], 1'b0};
end

/*****************************************************************************
 *                         Combinational logic                         *
 *****************************************************************************/

assign read_left_channel	= left_right_clk_rising_edge &
								~left_channel_fifo_is_empty &
								~right_channel_fifo_is_empty;
assign read_right_channel	= left_right_clk_falling_edge &
								left_channel_was_read;

/*****************************************************************************
 *                          Internal Modules                          *
 *****************************************************************************/

Altera_UP_SYNC_FIFO Audio_Out_Left_Channel_FIFO(
	// Inputs
	.clk			(clk),
	.reset			(reset),

	.write_en		(left_channel_data_en & ~left_channel_fifo_is_full),
	.write_data		(left_channel_data),

	.read_en		(read_left_channel),

	// Bidirectionals

	// Outputs
	.fifo_is_empty	(left_channel_fifo_is_empty),
	.fifo_is_full	(left_channel_fifo_is_full),
	.words_used		(left_channel_fifo_used),

	.read_data		(left_channel_from_fifo)
);
defparam
	Audio_Out_Left_Channel_FIFO.DATA_WIDTH		= AUDIO_DATA_WIDTH,
	Audio_Out_Left_Channel_FIFO.DATA_DEPTH		= 128,
	Audio_Out_Left_Channel_FIFO.ADDR_WIDTH		= 7;

Altera_UP_SYNC_FIFO Audio_Out_Right_Channel_FIFO(
	// Inputs
	.clk			(clk),
	.reset			(reset),

	.write_en		(right_channel_data_en & ~right_channel_fifo_is_full),
	.write_data		(right_channel_data),

	.read_en		(read_right_channel),

	// Bidirectionals

	// Outputs
	.fifo_is_empty	(right_channel_fifo_is_empty),
	.fifo_is_full	(right_channel_fifo_is_full),
	.words_used		(right_channel_fifo_used),

	.read_data		(right_channel_from_fifo)
);
defparam
	Audio_Out_Right_Channel_FIFO.DATA_WIDTH		= AUDIO_DATA_WIDTH,
	Audio_Out_Right_Channel_FIFO.DATA_DEPTH		= 128,
	Audio_Out_Right_Channel_FIFO.ADDR_WIDTH		= 7;
```

```
endmodule
```

B.4.4 Altera_UP_Clock_Edge

```
/*****************************************************************************
 *                                                                           *
 * Module:       Altera_UP_Clock_Edge                                        *
 * Description:                                                              *
 *      This module finds clock edges of one clock at the frquency of        *
 *   another clock.                                                          *
 *                                                                           *
 *****************************************************************************/

module Altera_UP_Clock_Edge (
	// Inputs
	clk,
	reset,
	
	test_clk,
	
	// Bidirectionals

	// Outputs
	rising_edge,
	falling_edge
);

/*****************************************************************************
 *                           Parameter Declarations                          *
 *****************************************************************************/


/*****************************************************************************
 *                             Port Declarations                             *
 *****************************************************************************/

// Inputs
input				clk;
input				reset;

input				test_clk;

// Bidirectionals
// Outputs
output                          rising_edge;
output                          falling_edge;

/**************************************************************************
 *                  Constant Declarations                    *
 **************************************************************************/

/**************************************************************************
 *            Internal wires and registers Declarations              *
 **************************************************************************/

// Internal Wires
wire                            found_edge;

// Internal Registers
reg                                     cur_test_clk;
reg                                     last_test_clk;

// State Machine Registers

/**************************************************************************
 *                  Finite State Machine(s)                  *
 **************************************************************************/


/**************************************************************************
 *                    Sequential logic                    *
 **************************************************************************/

always @(posedge clk)
        cur_test_clk    <= test_clk;

always @(posedge clk)
        last_test_clk   <= cur_test_clk;

/**************************************************************************
 *                   Combinational logic                   *
 **************************************************************************/

// Output Assignments
assign rising_edge      = found_edge & cur_test_clk;
assign falling_edge     = found_edge & last_test_clk;

// Internal Assignments
assign found_edge       = last_test_clk ^ cur_test_clk;
```

```
/*****************************************************************************
 *                        Internal Modules                        *
 *****************************************************************************/

endmodule
```

B.4.5 Altera_UP_SYNC_FIFO

```
/*****************************************************************************
 *                                                                *
 * Module:       Altera_UP_SYNC_FIFO                             *
 * Description:                                                 *
 *      This module is a FIFO with same clock for both reads and writes.     *
 *                                                                *
 *****************************************************************************/

module Altera_UP_SYNC_FIFO (
	// Inputs
	clk,
	reset,

	write_en,
	write_data,

	read_en,

	// Bidirectionals

	// Outputs
	fifo_is_empty,
	fifo_is_full,
	words_used,

	read_data
);

/*****************************************************************************
 *                      Parameter Declarations                      *
 *****************************************************************************/

parameter	DATA_WIDTH	= 32;
parameter	DATA_DEPTH= 128;
parameter	ADDR_WIDTH	= 7;

/*****************************************************************************
```

```
 *                        Port Declarations                        *
 *****************************************************************************/

// Inputs
input                           clk;
input                           reset;

input                           write_en;
input             [DATA_WIDTH:1]      write_data;

input                           read_en;

// Bidirectionals

// Outputs
output                          fifo_is_empty;
output                          fifo_is_full;
output            [ADDR_WIDTH:1]      words_used;

output            [DATA_WIDTH:1]      read_data;

/*****************************************************************************
 *              Internal wires and registers Declarations              *
 *****************************************************************************/

// Internal Wires

// Internal Registers

// State Machine Registers

/*****************************************************************************
 *                       Finite State Machine(s)                       *
 *****************************************************************************/


/*****************************************************************************
 *                        Sequential logic                        *
 *****************************************************************************/


/*****************************************************************************
 *                       Combinational logic                       *
 *****************************************************************************/
```

```
/*****************************************************************************
 *                         Internal Modules                          *
 *****************************************************************************/


scfifo    Sync_FIFO (
          // Inputs
          .clock                    (clk),
          .sclr                     (reset),

          .data                     (write_data),
          .wrreq                    (write_en),

          .rdreq                    (read_en),

          // Bidirectionals

          // Outputs
          .empty                    (fifo_is_empty),
          .full                     (fifo_is_full),
          .usedw                    (words_used),

          .q                                  (read_data)

          // Unused
          // synopsys translate_off
          ,
          .aclr                     (),
          .almost_empty   (),
          .almost_full       ()
          // synopsys translate_on
);
defparam
          Sync_FIFO.add_ram_output_register      = "OFF",
          Sync_FIFO.intended_device_family       = "Cyclone II",
          Sync_FIFO.lpm_numwords                               = DATA_DEPTH,
          Sync_FIFO.lpm_showahead                              = "ON",
          Sync_FIFO.lpm_type                                   = "scfifo",
          Sync_FIFO.lpm_width                                  = DATA_WIDTH,
          Sync_FIFO.lpm_widthu                       = ADDR_WIDTH,
          Sync_FIFO.overflow_checking                = "OFF",
          Sync_FIFO.underflow_checking               = "OFF",
          Sync_FIFO.use_eab                                    = "ON";

endmodule
```

B.4.6 Audio_Clock

```
// megafunction wizard: %ALTPLL%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altpll

// ============================================================
// File Name: Audio_Clock.v
// Megafunction Name(s):
//                  altpll
//
// Simulation Library Files(s):
//                  altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 7.2 Build 151 09/26/2007 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2007 Altera Corporation
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, Altera MegaCore Function License
//Agreement, or other applicable license agreement, including,
//without limitation, that your use is for the sole purpose of
//programming logic devices manufactured by Altera and sold by
//Altera or its authorized distributors.  Please refer to the
//applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module Audio_Clock (
        areset,
        inclk0,
        c0,
        locked);
```

```
input      areset;
input      inclk0;
output     c0;
output     locked;

wire [5:0] sub_wire0;
wire  sub_wire2;
wire [0:0] sub_wire5 = 1'h0;
wire [0:0] sub_wire1 = sub_wire0[0:0];
wire  c0 = sub_wire1;
wire  locked = sub_wire2;
wire  sub_wire3 = inclk0;
wire [1:0] sub_wire4 = {sub_wire5, sub_wire3};

altpll     altpll_component (
                            .inclk (sub_wire4),
                            .areset (areset),
                            .clk (sub_wire0),
                            .locked (sub_wire2),
                            .activeclock (),
                            .clkbad (),
                            .clkena ({6{1'b1}}),
                            .clkloss (),
                            .clkswitch (1'b0),
                            .configupdate (1'b0),
                            .enable0 (),
                            .enable1 (),
                            .extclk (),
                            .extclkena ({4{1'b1}}),
                            .fbin (1'b1),
                            .fbmimicbidir (),
                            .fbout (),
                            .pfdena (1'b1),
                            .phasecounterselect ({4{1'b1}}),
                            .phasedone (),
                            .phasestep (1'b1),
                            .phaseupdown (1'b1),
                            .pllena (1'b1),
                            .scanaclr (1'b0),
                            .scanclk (1'b0),
                            .scanclkena (1'b1),
                            .scandata (1'b0),
                            .scandataout (),
                            .scandone (),
                            .scanread (1'b0),
                            .scanwrite (1'b0),
                            .sclkout0 (),
                            .sclkout1 (),
                            .vcooverrange (),
                            .vcounderrange ());
defparam
        altpll_component.clk0_divide_by = 4,
        altpll_component.clk0_duty_cycle = 50,
        altpll_component.clk0_multiply_by = 1,
        altpll_component.clk0_phase_shift = "0",
        altpll_component.compensate_clock = "CLK0",
        altpll_component.gate_lock_signal = "NO",
        altpll_component.inclk0_input_frequency = 20000,
        altpll_component.intended_device_family = "Cyclone II",
        altpll_component.invalid_lock_multiplier = 5,
        altpll_component.lpm_hint = "CBX_MODULE_PREFIX=Audio_Clock",
        altpll_component.lpm_type = "altpll",
        altpll_component.operation_mode = "NORMAL",
        altpll_component.port_activeclock = "PORT_UNUSED",
        altpll_component.port_areset = "PORT_USED",
        altpll_component.port_clkbad0 = "PORT_UNUSED",
        altpll_component.port_clkbad1 = "PORT_UNUSED",
        altpll_component.port_clkloss = "PORT_UNUSED",
        altpll_component.port_clkswitch = "PORT_UNUSED",
        altpll_component.port_configupdate = "PORT_UNUSED",
        altpll_component.port_fbin = "PORT_UNUSED",
        altpll_component.port_inclk0 = "PORT_USED",
        altpll_component.port_inclk1 = "PORT_UNUSED",
        altpll_component.port_locked = "PORT_USED",
        altpll_component.port_pfdena = "PORT_UNUSED",
        altpll_component.port_phasecounterselect = "PORT_UNUSED",
        altpll_component.port_phasedone = "PORT_UNUSED",
        altpll_component.port_phasestep = "PORT_UNUSED",
        altpll_component.port_phaseupdown = "PORT_UNUSED",
        altpll_component.port_pllena = "PORT_UNUSED",
        altpll_component.port_scanaclr = "PORT_UNUSED",
        altpll_component.port_scanclk = "PORT_UNUSED",
        altpll_component.port_scanclkena = "PORT_UNUSED",
        altpll_component.port_scandata = "PORT_UNUSED",
        altpll_component.port_scandataout = "PORT_UNUSED",
        altpll_component.port_scandone = "PORT_UNUSED",
        altpll_component.port_scanread = "PORT_UNUSED",
        altpll_component.port_scanwrite = "PORT_UNUSED",
        altpll_component.port_clk0 = "PORT_USED",
        altpll_component.port_clk1 = "PORT_UNUSED",
        altpll_component.port_clk2 = "PORT_UNUSED",
                altpll_component.port_clk3 = "PORT_UNUSED",
                altpll_component.port_clk4 = "PORT_UNUSED",
                altpll_component.port_clk5 = "PORT_UNUSED",
                altpll_component.port_clkena0 = "PORT_UNUSED",
                altpll_component.port_clkena1 = "PORT_UNUSED",
                altpll_component.port_clkena2 = "PORT_UNUSED",
                altpll_component.port_clkena3 = "PORT_UNUSED",
                altpll_component.port_clkena4 = "PORT_UNUSED",
                altpll_component.port_clkena5 = "PORT_UNUSED",
                altpll_component.port_extclk0 = "PORT_UNUSED",
                altpll_component.port_extclk1 = "PORT_UNUSED",
                altpll_component.port_extclk2 = "PORT_UNUSED",
                altpll_component.port_extclk3 = "PORT_UNUSED",
                altpll_component.valid_lock_multiplier = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ACTIVECLK_CHECK STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH STRING "1.000"
// Retrieval info: PRIVATE: BANDWIDTH_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH_FREQ_UNIT STRING "MHz"
// Retrieval info: PRIVATE: BANDWIDTH_PRESET STRING "Low"
// Retrieval info: PRIVATE: BANDWIDTH_USE_AUTO STRING "1"
// Retrieval info: PRIVATE: BANDWIDTH_USE_CUSTOM STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH_USE_PRESET STRING "0"
// Retrieval info: PRIVATE: CLKBAD_SWITCHOVER_CHECK STRING "0"
// Retrieval info: PRIVATE: CLKLOSS_CHECK STRING "0"
// Retrieval info: PRIVATE: CLKSWITCH_CHECK STRING "1"
// Retrieval info: PRIVATE: CNX_NO_COMPENSATE_RADIO STRING "0"
// Retrieval info: PRIVATE: CREATE_CLKBAD_CHECK STRING "0"
// Retrieval info: PRIVATE: CREATE_INCLK1_CHECK STRING "0"
// Retrieval info: PRIVATE: CUR_DEDICATED_CLK STRING "c0"
// Retrieval info: PRIVATE: CUR_FBIN_CLK STRING "e0"
// Retrieval info: PRIVATE: DEVICE_SPEED_GRADE STRING "6"
// Retrieval info: PRIVATE: DIV_FACTOR0 NUMERIC "4"
// Retrieval info: PRIVATE: DUTY_CYCLE0 STRING "50.00000000"
// Retrieval info: PRIVATE: EXPLICIT_SWITCHOVER_COUNTER STRING "0"
// Retrieval info: PRIVATE: EXT_FEEDBACK_RADIO STRING "0"
// Retrieval info: PRIVATE: GLOCKED_COUNTER_EDIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: GLOCKED_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: GLOCKED_MODE_CHECK STRING "0"
// Retrieval info: PRIVATE: GLOCK_COUNTER_EDIT NUMERIC "1048575"
// Retrieval info: PRIVATE: HAS_MANUAL_SWITCHOVER STRING "1"
// Retrieval info: PRIVATE: INCLK0_FREQ_EDIT STRING "50.000"
// Retrieval info: PRIVATE: INCLK0_FREQ_UNIT_COMBO STRING "MHz"
// Retrieval info: PRIVATE: INCLK1_FREQ_EDIT STRING "100.000"
// Retrieval info: PRIVATE: INCLK1_FREQ_EDIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_COMBO STRING "MHz"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
// Retrieval info: PRIVATE: INT_FEEDBACK__MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: LOCKED_OUTPUT_CHECK STRING "1"
// Retrieval info: PRIVATE: LONG_SCAN_RADIO STRING "1"
// Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE STRING "Not Available"
// Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE_DIRTY NUMERIC "0"
// Retrieval info: PRIVATE: LVDS_PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: MIRROR_CLK0 STRING "0"
// Retrieval info: PRIVATE: MULT_FACTOR0 NUMERIC "1"
// Retrieval info: PRIVATE: NORMAL_MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: OUTPUT_FREQ0 STRING "100.00000000"
// Retrieval info: PRIVATE: OUTPUT_FREQ_MODE0 STRING "0"
// Retrieval info: PRIVATE: OUTPUT_FREQ_UNIT0 STRING "MHz"
// Retrieval info: PRIVATE: PHASE_RECONFIG_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: PHASE_RECONFIG_INPUTS_CHECK STRING "0"
// Retrieval info: PRIVATE: PHASE_SHIFT0 STRING "0.00000000"
// Retrieval info: PRIVATE: PHASE_SHIFT_STEP_ENABLED_CHECK STRING "0"
// Retrieval info: PRIVATE: PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: PLL_ADVANCED_PARAM_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_ARESET_CHECK STRING "1"
// Retrieval info: PRIVATE: PLL_AUTOPLL_CHECK NUMERIC "1"
// Retrieval info: PRIVATE: PLL_ENA_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_ENHPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_FASTPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_FBMIMIC_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_LVDS_PLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_PFDENA_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_TARGET_HARCOPY_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PRIMARY_CLK_COMBO STRING "inclk0"
// Retrieval info: PRIVATE: RECONFIG_FILE STRING "Audio_Clock.mif"
// Retrieval info: PRIVATE: SACN_INPUTS_CHECK STRING "0"
// Retrieval info: PRIVATE: SCAN_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: SELF_RESET_LOCK_LOSS STRING "0"
// Retrieval info: PRIVATE: SHORT_SCAN_RADIO STRING "0"
// Retrieval info: PRIVATE: SPREAD_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: SPREAD_FREQ STRING "50.000"
// Retrieval info: PRIVATE: SPREAD_FREQ_UNIT STRING "KHz"
// Retrieval info: PRIVATE: SPREAD_PERCENT STRING "0.500"
// Retrieval info: PRIVATE: SPREAD_USE STRING "0"
// Retrieval info: PRIVATE: SRC_SYNCH_COMP_RADIO STRING "0"
// Retrieval info: PRIVATE: STICKY_CLK0 STRING "1"
// Retrieval info: PRIVATE: SWITCHOVER_COUNT_EDIT NUMERIC "1"
// Retrieval info: PRIVATE: SWITCHOVER_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: USE_CLK0 STRING "1"
// Retrieval info: PRIVATE: USE_CLKENA0 STRING "0"
// Retrieval info: PRIVATE: USE_MIL_SPEED_GRADE NUMERIC "0"
// Retrieval info: PRIVATE: ZERO_DELAY_RADIO STRING "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLK0_DIVIDE_BY NUMERIC "4"
// Retrieval info: CONSTANT: CLK0_DUTY_CYCLE NUMERIC "50"
// Retrieval info: CONSTANT: CLK0_MULTIPLY_BY NUMERIC "1"
// Retrieval info: CONSTANT: CLK0_PHASE_SHIFT STRING "0"
// Retrieval info: CONSTANT: COMPENSATE_CLOCK STRING "CLK0"
// Retrieval info: CONSTANT: GATE_LOCK_SIGNAL STRING "NO"
// Retrieval info: CONSTANT: INCLK0_INPUT_FREQUENCY NUMERIC "20000"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
// Retrieval info: CONSTANT: INVALID_LOCK_MULTIPLIER NUMERIC "5"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altpll"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "NORMAL"
// Retrieval info: CONSTANT: PORT_ACTIVECLOCK STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_ARESET STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_CLKBAD0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKBAD1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKLOSS STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKSWITCH STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CONFIGUPDATE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_FBIN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_INCLK0 STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_INCLK1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_LOCKED STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_PFDENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASECOUNTERSELECT STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASEDONE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASESTEP STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASEUPDOWN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PLLENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANACLR STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANCLK STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANCLKENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDATA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDATAOUT STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDONE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANREAD STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANWRITE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk0 STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_clk1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk4 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk5 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena4 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena5 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: VALID_LOCK_MULTIPLIER NUMERIC "1"
// Retrieval info: USED_PORT: @clk 0 0 6 0 OUTPUT_CLK_EXT VCC "@clk[5..0]"
// Retrieval info: USED_PORT: @extclk 0 0 4 0 OUTPUT_CLK_EXT VCC "@extclk[3..0]"
// Retrieval info: USED_PORT: areset 0 0 0 0 INPUT GND "areset"
// Retrieval info: USED_PORT: c0 0 0 0 0 OUTPUT_CLK_EXT VCC "c0"
// Retrieval info: USED_PORT: inclk0 0 0 0 0 INPUT_CLK_EXT GND "inclk0"
// Retrieval info: USED_PORT: locked 0 0 0 0 OUTPUT GND "locked"
// Retrieval info: CONNECT: locked 0 0 0 0 @locked 0 0 0 0
// Retrieval info: CONNECT: @inclk 0 0 1 0 inclk0 0 0 0 0
// Retrieval info: CONNECT: c0 0 0 0 0 @clk 0 0 1 0
// Retrieval info: CONNECT: @inclk 0 0 1 1 GND 0 0 0 0
// Retrieval info: CONNECT: @areset 0 0 0 0 areset 0 0 0 0
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock.v TRUE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock.ppf TRUE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock.inc FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock.cmp FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock.bsf FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock_inst.v FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock_bb.v TRUE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock_waveforms.html TRUE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL Audio_Clock_wave*.jpg FALSE FALSE
// Retrieval info: LIB_FILE: altera_mf
// Retrieval info: CBX_MODULE_PREFIX: ON
```

B.4.7 Altera_UP_Avalon_Audio

```
/*****************************************************************************
 *                                                   *
 * Module:       Altera_UP_Avalon_Audio                              *
 * Description:                                                    *
 *     This module reads and writes data to the Audio chip on Altera's DE2  *
 *   Development and Education Board. The audio chip must be in master mode  *
 *   and the digital format must be left justified.                    *
 *                                                          *
 ***************************************************************************/

module Audio_Controller(
        // Inputs
        CLOCK_50,
        reset,

        clear_audio_in_memory,
        read_audio_in,

        clear_audio_out_memory,
        left_channel_audio_out,
        right_channel_audio_out,
        write_audio_out,

        AUD_ADCDAT,

        // Bidirectionals
        AUD_BCLK,
        AUD_ADCLRCK,
        AUD_DACLRCK,

        // Outputs
        left_channel_audio_in,
        right_channel_audio_in,
        audio_in_available,

        audio_out_allowed,

        AUD_XCK,
        AUD_DACDAT
);

/***************************************************************************
 *                    Parameter Declarations                    *
 ***************************************************************************/

localparam AUDIO_DATA_WIDTH     = 32;
localparam BIT_COUNTER_INIT     = 5'd31;

/***************************************************************************
```

```
 *                         Port Declarations                         *
 *****************************************************************************/
// Inputs
input                           CLOCK_50;
input                           reset;

input                           clear_audio_in_memory;
input                           read_audio_in;

input                           clear_audio_out_memory;
input           [AUDIO_DATA_WIDTH:1]    left_channel_audio_out;
input           [AUDIO_DATA_WIDTH:1]    right_channel_audio_out;
input                           write_audio_out;

input                           AUD_ADCDAT;

// Bidirectionals
inout                           AUD_BCLK;
inout                           AUD_ADCLRCK;
inout                           AUD_DACLRCK;

// Outputs
output  reg                     audio_in_available;
output          [AUDIO_DATA_WIDTH:1]    left_channel_audio_in;
output          [AUDIO_DATA_WIDTH:1]    right_channel_audio_in;

output  reg                     audio_out_allowed;

output                          AUD_XCK;
output                          AUD_DACDAT;

/*****************************************************************************
 *              Internal wires and registers Declarations              *
 *****************************************************************************/

// Internal Wires
wire                            bclk_rising_edge;
wire                            bclk_falling_edge;

wire                            adc_lrclk_rising_edge;
wire                            adc_lrclk_falling_edge;

wire                            dac_lrclk_rising_edge;
wire                            dac_lrclk_falling_edge;

wire            [7:0]   left_channel_read_available;
wire                [7:0]    right_channel_read_available;

wire                [7:0]    left_channel_write_space;
wire                [7:0]    right_channel_write_space;

// Internal Registers
reg                                     done_adc_channel_sync;
reg                                     done_dac_channel_sync;

// State Machine Registers


/*****************************************************************************
 *                       Finite State Machine(s)                       *
 *****************************************************************************/


/*****************************************************************************
 *                          Sequential logic                        *
 *****************************************************************************/

// Output Registers
always @ (posedge CLOCK_50)
begin
        if (reset == 1'b1)
                audio_in_available <= 1'b0;
        else if ((left_channel_read_available[7] | left_channel_read_available[6])
                        & (right_channel_read_available[7] | right_channel_read_available[6]))
                audio_in_available <= 1'b1;
        else
                audio_in_available <= 1'b0;
end

always @ (posedge CLOCK_50)
begin
        if (reset == 1'b1)
                audio_out_allowed <= 1'b0;
        else if ((left_channel_write_space[7] | left_channel_write_space[6])
                        & (right_channel_write_space[7] | right_channel_write_space[6]))
                audio_out_allowed <= 1'b1;
        else
                audio_out_allowed <= 1'b0;
end

// Internal Registers
always @ (posedge CLOCK_50)
```

```
begin
	if (reset == 1'b1)
		done_adc_channel_sync <= 1'b0;
	else if (adc_lrclk_rising_edge == 1'b1)
		done_adc_channel_sync <= 1'b1;
end

always @ (posedge CLOCK_50)
begin
	if (reset == 1'b1)
		done_dac_channel_sync <= 1'b0;
	else if (dac_lrclk_falling_edge == 1'b1)
		done_dac_channel_sync <= 1'b1;
end

/*****************************************************************************
 *                      Combinational logic                          *
 *****************************************************************************/

assign AUD_BCLK             = 1'bZ;
assign AUD_ADCLRCK          = 1'bZ;
assign AUD_DACLRCK          = 1'bZ;


/*****************************************************************************
 *                        Internal Modules                         *
 *****************************************************************************/

Altera_UP_Clock_Edge Bit_Clock_Edges (
	// Inputs
	.clk                  (CLOCK_50),
	.reset                (reset),

	.test_clk             (AUD_BCLK),

	// Bidirectionals

	// Outputs
	.rising_edge    (bclk_rising_edge),
	.falling_edge   (bclk_falling_edge)
);

Altera_UP_Clock_Edge ADC_Left_Right_Clock_Edges (
	// Inputs
	.clk                  (CLOCK_50),
	.reset                (reset),
	.test_clk			(AUD_ADCLRCK),

	// Bidirectionals

	// Outputs
	.rising_edge	(adc_lrclk_rising_edge),
	.falling_edge	(adc_lrclk_falling_edge)
);

Altera_UP_Clock_Edge DAC_Left_Right_Clock_Edges (
	// Inputs
	.clk			(CLOCK_50),
	.reset			(reset),

	.test_clk			(AUD_DACLRCK),

	// Bidirectionals

	// Outputs
	.rising_edge	(dac_lrclk_rising_edge),
	.falling_edge	(dac_lrclk_falling_edge)
);

Altera_UP_Audio_In_Deserializer Audio_In_Deserializer (
	// Inputs
	.clk						(CLOCK_50),
	.reset						(reset | clear_audio_in_memory),

	.bit_clk_rising_edge			(bclk_rising_edge),
	.bit_clk_falling_edge			(bclk_falling_edge),
	.left_right_clk_rising_edge		(adc_lrclk_rising_edge),
	.left_right_clk_falling_edge	(adc_lrclk_falling_edge),

	.done_channel_sync				(done_adc_channel_sync),

	.serial_audio_in_data			(AUD_ADCDAT),

	.read_left_audio_data_en		(read_audio_in & audio_in_available),
	.read_right_audio_data_en		(read_audio_in & audio_in_available),

	// Bidirectionals

	// Outputs
	.left_audio_fifo_read_space		(left_channel_read_available),
	.right_audio_fifo_read_space	(right_channel_read_available),
	.left_channel_data				(left_channel_audio_in),
	.right_channel_data				(right_channel_audio_in)
);
defparam
	Audio_In_Deserializer.AUDIO_DATA_WIDTH = AUDIO_DATA_WIDTH,
	Audio_In_Deserializer.BIT_COUNTER_INIT = BIT_COUNTER_INIT;

Altera_UP_Audio_Out_Serializer Audio_Out_Serializer (
	// Inputs
	.clk						(CLOCK_50),
	.reset						(reset | clear_audio_out_memory),

	.bit_clk_rising_edge			(bclk_rising_edge),
	.bit_clk_falling_edge			(bclk_falling_edge),
	.left_right_clk_rising_edge		(done_dac_channel_sync & dac_lrclk_rising_edge),
	.left_right_clk_falling_edge	(done_dac_channel_sync & dac_lrclk_falling_edge),

	.left_channel_data				(left_channel_audio_out),
	.left_channel_data_en			(write_audio_out & audio_out_allowed),

	.right_channel_data				(right_channel_audio_out),
	.right_channel_data_en			(write_audio_out & audio_out_allowed),

	// Bidirectionals

	// Outputs
	.left_channel_fifo_write_space	(left_channel_write_space),
	.right_channel_fifo_write_space	(right_channel_write_space),

	.serial_audio_out_data			(AUD_DACDAT)
);
defparam
	Audio_Out_Serializer.AUDIO_DATA_WIDTH = AUDIO_DATA_WIDTH;

Audio_Clock Audio_Clock (
	// Inputs
	.inclk0			(CLOCK_50),
	.areset			(),

	// Outputs
	.c0				(AUD_XCK),
	.locked			()
);

endmodule
```

B.4.8 avconf (Eecg.toronto.edu, 2019)

```
module avconf (          //          Host Side
                                                        CLOCK_50,
                                                        reset,
                                                        //          I2C Side
                                                        FPGA_I2C_SCLK,
                                                        FPGA_I2C_SDAT        );
//          Host Side
input               CLOCK_50;
input               reset;
//          I2C Side
output              FPGA_I2C_SCLK;
inout               FPGA_I2C_SDAT;
//          Internal Registers/Wires
reg         [15:0]  mI2C_CLK_DIV;
reg         [23:0]  mI2C_DATA;
reg                         mI2C_CTRL_CLK;
reg                         mI2C_GO;
wire                mI2C_END;
wire                mI2C_ACK;
wire                iRST_N = !reset;
reg         [15:0]  LUT_DATA;
reg         [5:0]   LUT_INDEX;
reg         [3:0]   mSetup_ST;

parameter USE_MIC_INPUT               = 1'b0;

parameter AUD_LINE_IN_LC  = 9'd24;
parameter AUD_LINE_IN_RC  = 9'd24;
parameter AUD_LINE_OUT_LC          = 9'd119;
parameter AUD_LINE_OUT_RC          = 9'd119;
parameter AUD_ADC_PATH             = 9'd17;
parameter AUD_DAC_PATH             = 9'd6;
parameter AUD_POWER                         = 9'h000;
parameter AUD_DATA_FORMAT          = 9'd77;
parameter AUD_SAMPLE_CTRL          = 9'd0;
parameter AUD_SET_ACTIVE = 9'h001;

//          Clock Setting
parameter           CLK_Freq      =        50000000;        //         50        MHz
parameter           I2C_Freq      =        20000;           //         20        KHz
//          LUT Data Number
parameter           LUT_SIZE      =        50;
//          Audio Data Index
```

```
parameter	SET_LIN_L	=	0;
parameter	SET_LIN_R	=	1;
parameter	SET_HEAD_L =	2;
parameter	SET_HEAD_R =	3;
parameter	A_PATH_CTRL	=	4;
parameter	D_PATH_CTRL	=	5;
parameter	POWER_ON	=	6;
parameter	SET_FORMAT=	7;
parameter	SAMPLE_CTRL	=	8;
parameter	SET_ACTIVE	=	9;
//	Video Data Index
parameter	SET_VIDEO	=	10;

/////////////////////	I2C Control Clock	////////////////////////
always@(posedge CLOCK_50 or negedge iRST_N)
begin
	if(!iRST_N)
	begin
		mI2C_CTRL_CLK	<=	0;
		mI2C_CLK_DIV	<=	0;
	end
	else
	begin
		if( mI2C_CLK_DIV	< (CLK_Freq/I2C_Freq) )
		mI2C_CLK_DIV	<=	mI2C_CLK_DIV+1;
		else
		begin
			mI2C_CLK_DIV	<=	0;
			mI2C_CTRL_CLK	<=	~mI2C_CTRL_CLK;
		end
	end
end
////////////////////////////////////////////////////////////////////
I2C_Controller	u0	(	.CLOCK(mI2C_CTRL_CLK),		//	Controller Work
Clock
					.FPGA_I2C_SCLK(FPGA_I2C_SCLK),
	//	I2C CLOCK
					.FPGA_I2C_SDAT(FPGA_I2C_SDAT),
	//	I2C DATA
					.I2C_DATA(mI2C_DATA),		//
DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
					.GO(mI2C_GO),			//	GO
transfor
					.END(mI2C_END),				//
	END transfor
```

```
							.ACK(mI2C_ACK),				//
	ACK
							.RESET(iRST_N)	);
////////////////////////////////////////////////////////////////////
//////////////////////  Config Control  ////////////////////////////
always@(posedge mI2C_CTRL_CLK or negedge iRST_N)
begin
	if(!iRST_N)
	begin
		LUT_INDEX	<=	0;
		mSetup_ST	<=	0;
		mI2C_GO		<=	0;
	end
	else
	begin
		if(LUT_INDEX<LUT_SIZE)
		begin
			case(mSetup_ST)
			0:	begin
					if(LUT_INDEX<SET_VIDEO)
					mI2C_DATA	<=	{8'h34,LUT_DATA};
					else
					mI2C_DATA	<=	{8'h40,LUT_DATA};
					mI2C_GO		<=	1;
					mSetup_ST	<=	1;
				end
			1:	begin
					if(mI2C_END)
					begin
						if(!mI2C_ACK)
						mSetup_ST	<=	2;
						else
						mSetup_ST	<=	0;

						mI2C_GO		<=	0;
					end
				end
			2:	begin
					LUT_INDEX	<=	LUT_INDEX+1;
					mSetup_ST	<=	0;
				end
			endcase
		end
	end
end
////////////////////////////////////////////////////////////////////
```

```
/////////////////////	Config Data LUT		//////////////////////////
always
begin
	LUT_DATA	<=	16'h0000;
	case(LUT_INDEX)
	//	Audio Config Data
	SET_LIN_L	:	LUT_DATA	<=	{7'h0, AUD_LINE_IN_LC};
	SET_LIN_R	:	LUT_DATA	<=	{7'h1, AUD_LINE_IN_RC};
	SET_HEAD_L	:	LUT_DATA	<=	{7'h2, AUD_LINE_OUT_LC};
	SET_HEAD_R	:	LUT_DATA	<=	{7'h3, AUD_LINE_OUT_RC};
	A_PATH_CTRL	:	LUT_DATA	<=	{7'h4, AUD_ADC_PATH} + 
(16'h0004 * USE_MIC_INPUT);
	D_PATH_CTRL	:	LUT_DATA	<=	{7'h5, AUD_DAC_PATH};
	POWER_ON	:	LUT_DATA	<=	{7'h6, AUD_POWER};
	SET_FORMAT	:	LUT_DATA	<=	{7'h7, AUD_DATA_FORMAT};
	SAMPLE_CTRL	:	LUT_DATA	<=	{7'h8, AUD_SAMPLE_CTRL};
	SET_ACTIVE	:	LUT_DATA	<=	{7'h9, AUD_SET_ACTIVE};
	//	Video Config Data
	SET_VIDEO+0:	LUT_DATA	<=	16'h1500;
	SET_VIDEO+1:	LUT_DATA	<=	16'h1741;
	SET_VIDEO+2:	LUT_DATA	<=	16'h3a16;
	SET_VIDEO+3:	LUT_DATA	<=  16'h503f; // 16'h5004;
	SET_VIDEO+4:	LUT_DATA	<=	16'hc305;
	SET_VIDEO+5:	LUT_DATA	<=	16'hc480;
	SET_VIDEO+6:	LUT_DATA	<=	16'h0e80;
	SET_VIDEO+7:	LUT_DATA	<=	16'h503f; // 16'h5020;
	SET_VIDEO+8:	LUT_DATA	<=	16'h5218;
	SET_VIDEO+9:	LUT_DATA	<=	16'h58ed;
	SET_VIDEO+10:	LUT_DATA	<=	16'h77c5;
	SET_VIDEO+11:	LUT_DATA	<=	16'h7c93;
	SET_VIDEO+12:	LUT_DATA	<=	16'h7d00;
	SET_VIDEO+13:	LUT_DATA	<=	16'hd048;
	SET_VIDEO+14:	LUT_DATA	<=	16'hd5a0;
	SET_VIDEO+15:	LUT_DATA	<=	16'hd7ea;
	SET_VIDEO+16:	LUT_DATA	<=	16'he43e;
	SET_VIDEO+17:	LUT_DATA	<=	16'hea0f;
	SET_VIDEO+18:	LUT_DATA	<=	16'h3112;
	SET_VIDEO+19:	LUT_DATA	<=	16'h3281;
	SET_VIDEO+20:	LUT_DATA	<=	16'h3384;
	SET_VIDEO+21:	LUT_DATA	<=	16'h37A0;
	SET_VIDEO+22:	LUT_DATA	<=	16'he580;
	SET_VIDEO+23:	LUT_DATA	<=	16'he603;
	SET_VIDEO+24:	LUT_DATA	<=	16'he785;
	SET_VIDEO+25:	LUT_DATA	<=	16'h2778; // 16'h503f; // 16'h5000;
	SET_VIDEO+26:	LUT_DATA	<=	16'h5100;
	SET_VIDEO+27:	LUT_DATA	<=	16'h0050;
```

```
		SET_VIDEO+28:		LUT_DATA	<=	16'h1000;
		SET_VIDEO+29:		LUT_DATA	<=	16'h0402;
		SET_VIDEO+30:		LUT_DATA	<=	16'h0860;
		SET_VIDEO+31:		LUT_DATA	<=	16'h0a18;
		SET_VIDEO+32:		LUT_DATA	<=	16'h1100;
		SET_VIDEO+33:		LUT_DATA	<=	16'h2b00;
		SET_VIDEO+34:		LUT_DATA	<=	16'h2c8c;
		SET_VIDEO+35:		LUT_DATA	<=	16'h2df8;
		SET_VIDEO+36:		LUT_DATA	<=	16'h2eee;
		SET_VIDEO+37:		LUT_DATA	<=	16'h2ff4;
		SET_VIDEO+38:		LUT_DATA	<=	16'h30d2;
		SET_VIDEO+39:		LUT_DATA	<=	16'h0e05;
		endcase
end
////////////////////////////////////////////////////////////////////
endmodule
```

B.4.9 i2c controller

```
// --------------------------------------------------------------------
// Copyright (c) 2005 by Terasic Technologies Inc.
// --------------------------------------------------------------------
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altrea Development
//   Kits made by Terasic.  Other use of this code, including the selling
//   ,duplication, or modification of any portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL or Verilog source code is intended as a design reference
//   which illustrates how these types of functions can be implemented.
//   It is the user's responsibility to verify their design for
//   consistency and functionality through the use of formal
//   verification methods.  Terasic provides no warranty regarding the use
//   or functionality of this code.
//
// --------------------------------------------------------------------
//
//                     Terasic Technologies Inc
//                     356 Fu-Shin E. Rd Sec. 1. JhuBei City,
//                     HsinChu County, Taiwan
//                     302
```

```
//
//                  web: http://www.terasic.com/
//                  email: support@terasic.com
//
// --------------------------------------------------------------------
//
// Major Functions:i2c controller
//
// --------------------------------------------------------------------
//
// Revision History :
// --------------------------------------------------------------------
//   Ver  :| Author           :| Mod. Date :| Changes Made:
//   V1.0 :| Joe Yang         :| 05/07/10  :|      Initial Revision
//   V2.0 :| Paul Chow        :| 10/31/17  :| For DE1_SoC
// --------------------------------------------------------------------
module I2C_Controller (
	CLOCK,
	FPGA_I2C_SCLK,//I2C CLOCK
	FPGA_I2C_SDAT,//I2C DATA
	I2C_DATA,//DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
	GO,      //GO transfor
	END,     //END transfor
	W_R,     //W_R
	ACK,      //ACK
	RESET,
	//TEST
	SD_COUNTER,
	SDO
);
	input  CLOCK;
	input  [23:0]I2C_DATA;
	input  GO;
	input  RESET;
	input  W_R;
	inout  FPGA_I2C_SDAT;
	output FPGA_I2C_SCLK;
	output END;
	output ACK;

//TEST
	output [5:0] SD_COUNTER;
	output SDO;


reg SDO;
reg SCLK;
reg END;
reg [23:0]SD;
reg [5:0]SD_COUNTER;

wire FPGA_I2C_SCLK=SCLK | ( ((SD_COUNTER >= 4) & (SD_COUNTER <=30))? ~CLOCK :0
);
wire FPGA_I2C_SDAT=SDO?1'bz:0 ;

reg ACK1,ACK2,ACK3;
wire ACK=ACK1 | ACK2 |ACK3;

//--I2C COUNTER
always @(negedge RESET or posedge CLOCK ) begin
if (!RESET) SD_COUNTER=6'b111111;
else begin
if (GO==0)
        SD_COUNTER=0;
        else
        if (SD_COUNTER < 6'b111111) SD_COUNTER=SD_COUNTER+1;
end
end
//----

always @(negedge RESET or  posedge CLOCK ) begin
if (!RESET) begin SCLK=1;SDO=1; ACK1=0;ACK2=0;ACK3=0; END=1; end
else
case (SD_COUNTER)
        6'd0  : begin ACK1=0 ;ACK2=0 ;ACK3=0 ; END=0; SDO=1; SCLK=1;end
        //start
        6'd1  : begin SD=I2C_DATA;SDO=0;end
        6'd2  : SCLK=0;
        //SLAVE ADDR
        6'd3  : SDO=SD[23];
        6'd4  : SDO=SD[22];
        6'd5  : SDO=SD[21];
        6'd6  : SDO=SD[20];
        6'd7  : SDO=SD[19];
        6'd8  : SDO=SD[18];
        6'd9  : SDO=SD[17];
        6'd10 : SDO=SD[16];
        6'd11 : SDO=1'b1;//ACK

        //SUB ADDR
        6'd12  : begin SDO=SD[15]; ACK1=FPGA_I2C_SDAT; end
        6'd13  : SDO=SD[14];
        6'd14  : SDO=SD[13];
        6'd15  : SDO=SD[12];
        6'd16  : SDO=SD[11];
        6'd17  : SDO=SD[10];
        6'd18  : SDO=SD[9];
        6'd19  : SDO=SD[8];
        6'd20  : SDO=1'b1;//ACK

        //DATA
        6'd21  : begin SDO=SD[7]; ACK2=FPGA_I2C_SDAT; end
        6'd22  : SDO=SD[6];
        6'd23  : SDO=SD[5];
        6'd24  : SDO=SD[4];
        6'd25  : SDO=SD[3];
        6'd26  : SDO=SD[2];
        6'd27  : SDO=SD[1];
        6'd28  : SDO=SD[0];
        6'd29  : SDO=1'b1;//ACK


        //stop
    6'd30 : begin SDO=1'b0;     SCLK=1'b0; ACK3=FPGA_I2C_SDAT; end
    6'd31 : SCLK=1'b1;
    6'd32 : begin SDO=1'b1; END=1; end

endcase
end



endmodule
```

B.5 PS2_controller (Eecg.toronto.edu, 2019)

B.5.1 Altera_UP_PS2

```
/*****************************************************************************
 *                                                                           *
 * Module:       Altera_UP_PS2                                               *
 * Description:                                                              *
 *      This module communicates with the PS2 core.                          *
 *                                                                           *
 *****************************************************************************/
```

```
module PS2_Controller #(parameter INITIALIZE_MOUSE = 1) (
	// Inputs
	CLOCK_50,
	reset,

	the_command,
	send_command,

	// Bidirectionals
	PS2_CLK,					// PS2 Clock
	PS2_DAT,					// PS2 Data

	// Outputs
	command_was_sent,
	error_communication_timed_out,

	received_data,
	received_data_en			// If 1 - new data has been received
);

/*****************************************************************************
 *                             Parameter Declarations                        *
 *****************************************************************************/


/*****************************************************************************
 *                             Port Declarations                             *
 *****************************************************************************/
// Inputs
input			CLOCK_50;
input			reset;

input	[7:0]	the_command;
input			send_command;

// Bidirectionals
inout			PS2_CLK;
inout			PS2_DAT;

// Outputs
output			command_was_sent;
output			error_communication_timed_out;

output	[7:0]	received_data;
output			received_data_en;
```

```
wire [7:0] the_command_w;
wire send_command_w, command_was_sent_w, error_communication_timed_out_w;

generate
        if(INITIALIZE_MOUSE) begin
                assign the_command_w = init_done ? the_command : 8'hf4;
                assign send_command_w = init_done ? send_command : (!command_was_sent_w
&& !error_communication_timed_out_w);
                assign command_was_sent = init_done ? command_was_sent_w : 0;
                assign error_communication_timed_out = init_done ?
error_communication_timed_out_w : 1;

                reg init_done;

                always @(posedge CLOCK_50)
                        if(reset) init_done <= 0;
                        else if(command_was_sent_w) init_done <= 1;

        end else begin
                assign the_command_w = the_command;
                assign send_command_w = send_command;
                assign command_was_sent = command_was_sent_w;
                assign error_communication_timed_out = error_communication_timed_out_w;
        end
endgenerate

/*****************************************************************************
 *                    Constant Declarations                     *
 *****************************************************************************/
// states
localparam      PS2_STATE_0_IDLE                = 3'h0,
                        PS2_STATE_1_DATA_IN                     = 3'h1,
                        PS2_STATE_2_COMMAND_OUT                 = 3'h2,
                        PS2_STATE_3_END_TRANSFER        = 3'h3,
                        PS2_STATE_4_END_DELAYED                 = 3'h4;

/*****************************************************************************
 *             Internal wires and registers Declarations              *
 *****************************************************************************/
// Internal Wires
wire                    ps2_clk_posedge;
wire                    ps2_clk_negedge;

wire                    start_receiving_data;
wire                    wait_for_incoming_data;
```

```
// Internal Registers
reg             [7:0]   idle_counter;

reg                     ps2_clk_reg;
reg                     ps2_data_reg;
reg                     last_ps2_clk;

// State Machine Registers
reg             [2:0]   ns_ps2_transceiver;
reg             [2:0]   s_ps2_transceiver;

/*****************************************************************************
 *                 Finite State Machine(s)                   *
 *****************************************************************************/

always @(posedge CLOCK_50)
begin
        if (reset == 1'b1)
                s_ps2_transceiver <= PS2_STATE_0_IDLE;
        else
                s_ps2_transceiver <= ns_ps2_transceiver;
end

always @(*)
begin
        // Defaults
        ns_ps2_transceiver = PS2_STATE_0_IDLE;

   case (s_ps2_transceiver)
        PS2_STATE_0_IDLE:
                begin
                        if ((idle_counter == 8'hFF) &&
                                        (send_command == 1'b1))
                                ns_ps2_transceiver = PS2_STATE_2_COMMAND_OUT;
                        else if ((ps2_data_reg == 1'b0) && (ps2_clk_posedge == 1'b1))
                                ns_ps2_transceiver = PS2_STATE_1_DATA_IN;
                        else
                                ns_ps2_transceiver = PS2_STATE_0_IDLE;
                end
        PS2_STATE_1_DATA_IN:
                begin
                        if ((received_data_en == 1'b1)/* && (ps2_clk_posedge == 1'b1)*/)
                                ns_ps2_transceiver = PS2_STATE_0_IDLE;
                        else
                                ns_ps2_transceiver = PS2_STATE_1_DATA_IN;
                end
	PS2_STATE_2_COMMAND_OUT:
		begin
			if ((command_was_sent == 1'b1) ||
				(error_communication_timed_out == 1'b1))
				ns_ps2_transceiver = PS2_STATE_3_END_TRANSFER;
			else
				ns_ps2_transceiver = PS2_STATE_2_COMMAND_OUT;
		end
	PS2_STATE_3_END_TRANSFER:
		begin
			if (send_command == 1'b0)
				ns_ps2_transceiver = PS2_STATE_0_IDLE;
			else if ((ps2_data_reg == 1'b0) && (ps2_clk_posedge == 1'b1))
				ns_ps2_transceiver = PS2_STATE_4_END_DELAYED;
			else
				ns_ps2_transceiver = PS2_STATE_3_END_TRANSFER;
		end
	PS2_STATE_4_END_DELAYED:
		begin
			if (received_data_en == 1'b1)
			begin
				if (send_command == 1'b0)
					ns_ps2_transceiver = PS2_STATE_0_IDLE;
				else
					ns_ps2_transceiver = PS2_STATE_3_END_TRANSFER;
			end
			else
				ns_ps2_transceiver = PS2_STATE_4_END_DELAYED;
		end
	default:
			ns_ps2_transceiver = PS2_STATE_0_IDLE;
	endcase
end

/*****************************************************************************
 *                        Sequential logic                            *
 *****************************************************************************/

always @(posedge CLOCK_50)
begin
	if (reset == 1'b1)
	begin
		last_ps2_clk	<= 1'b1;
		ps2_clk_reg		<= 1'b1;

		ps2_data_reg	<= 1'b1;
		end
		else
		begin
			last_ps2_clk	<= ps2_clk_reg;
			ps2_clk_reg			<= PS2_CLK;

			ps2_data_reg	<= PS2_DAT;
		end
end

always @(posedge CLOCK_50)
begin
	if (reset == 1'b1)
		idle_counter <= 6'h00;
	else if ((s_ps2_transceiver == PS2_STATE_0_IDLE) &&
			(idle_counter != 8'hFF))
		idle_counter <= idle_counter + 6'h01;
	else if (s_ps2_transceiver != PS2_STATE_0_IDLE)
		idle_counter <= 6'h00;
end

/*****************************************************************************
 *                         Combinational logic                               *
 *****************************************************************************/

assign ps2_clk_posedge =
			((ps2_clk_reg == 1'b1) && (last_ps2_clk == 1'b0)) ? 1'b1 : 1'b0;
assign ps2_clk_negedge =
			((ps2_clk_reg == 1'b0) && (last_ps2_clk == 1'b1)) ? 1'b1 : 1'b0;

assign start_receiving_data		= (s_ps2_transceiver == PS2_STATE_1_DATA_IN);
assign wait_for_incoming_data	=
			(s_ps2_transceiver == PS2_STATE_3_END_TRANSFER);

/*****************************************************************************
 *                          Internal Modules                                 *
 *****************************************************************************/

Altera_UP_PS2_Data_In PS2_Data_In (
	// Inputs
	.clk								(CLOCK_50),
	.reset								(reset),

	.wait_for_incoming_data				(wait_for_incoming_data),
	.start_receiving_data			(start_receiving_data),
	.ps2_clk_posedge			(ps2_clk_posedge),
	.ps2_clk_negedge			(ps2_clk_negedge),
	.ps2_data				(ps2_data_reg),

	// Bidirectionals

	// Outputs
	.received_data			(received_data),
	.received_data_en			(received_data_en)
);

Altera_UP_PS2_Command_Out PS2_Command_Out (
	// Inputs
	.clk					(CLOCK_50),
	.reset				(reset),

	.the_command			(the_command_w),
	.send_command			(send_command_w),

	.ps2_clk_posedge			(ps2_clk_posedge),
	.ps2_clk_negedge			(ps2_clk_negedge),

	// Bidirectionals
	.PS2_CLK				(PS2_CLK),
	.PS2_DAT				(PS2_DAT),

	// Outputs
	.command_was_sent			(command_was_sent_w),
	.error_communication_timed_out		(error_communication_timed_out_w)
);

endmodule
```

B.5.2 Altera_UP_PS2_Data_In

```
/*****************************************************************************
 *                                                                           *
 * Module:       Altera_UP_PS2_Data_In                                       *
 * Description:                                                              *
 *      This module accepts incoming data from a PS2 core.                   *
 *                                                                           *
 *****************************************************************************/


module Altera_UP_PS2_Data_In (
	// Inputs
	clk,
	reset,

	wait_for_incoming_data,
	start_receiving_data,

	ps2_clk_posedge,
	ps2_clk_negedge,
	ps2_data,

	// Bidirectionals

	// Outputs
	received_data,
	received_data_en			// If 1 - new data has been received
);


/*****************************************************************************
 *                           Parameter Declarations                          *
 *****************************************************************************/


/*****************************************************************************
 *                             Port Declarations                             *
 *****************************************************************************/
// Inputs
input				clk;
input				reset;

input				wait_for_incoming_data;
input				start_receiving_data;

input				ps2_clk_posedge;
input				ps2_clk_negedge;
input				ps2_data;

// Bidirectionals

// Outputs
output reg	[7:0]	received_data;

output reg			received_data_en;

/*****************************************************************************
 *                           Constant Declarations                           *
 ***************************************************************************/
// states
localparam      PS2_STATE_0_IDLE                    = 3'h0,
                        PS2_STATE_1_WAIT_FOR_DATA   = 3'h1,
                        PS2_STATE_2_DATA_IN                 = 3'h2,
                        PS2_STATE_3_PARITY_IN           = 3'h3,
                        PS2_STATE_4_STOP_IN                 = 3'h4;

/***************************************************************************
 *              Internal wires and registers Declarations              *
 ***************************************************************************/
// Internal Wires
reg                     [3:0]   data_count;
reg                     [7:0]   data_shift_reg;

// State Machine Registers
reg                     [2:0]   ns_ps2_receiver;
reg                     [2:0]   s_ps2_receiver;

/***************************************************************************
 *                      Finite State Machine(s)                        *
 ***************************************************************************/

always @(posedge clk)
begin
        if (reset == 1'b1)
                s_ps2_receiver <= PS2_STATE_0_IDLE;
        else
                s_ps2_receiver <= ns_ps2_receiver;
end

always @(*)
begin
        // Defaults
        ns_ps2_receiver = PS2_STATE_0_IDLE;

   case (s_ps2_receiver)
        PS2_STATE_0_IDLE:
                begin
                        if ((wait_for_incoming_data == 1'b1) &&
                                        (received_data_en == 1'b0))
                                ns_ps2_receiver = PS2_STATE_1_WAIT_FOR_DATA;
                        else if ((start_receiving_data == 1'b1) &&
                                        (received_data_en == 1'b0))
                                ns_ps2_receiver = PS2_STATE_2_DATA_IN;
                        else
```

```
                    ns_ps2_receiver = PS2_STATE_0_IDLE;
            end
        PS2_STATE_1_WAIT_FOR_DATA:
            begin
                if ((ps2_data == 1'b0) && (ps2_clk_posedge == 1'b1))
                    ns_ps2_receiver = PS2_STATE_2_DATA_IN;
                else if (wait_for_incoming_data == 1'b0)
                    ns_ps2_receiver = PS2_STATE_0_IDLE;
                else
                    ns_ps2_receiver = PS2_STATE_1_WAIT_FOR_DATA;
            end
        PS2_STATE_2_DATA_IN:
            begin
                if ((data_count == 3'h7) && (ps2_clk_posedge == 1'b1))
                    ns_ps2_receiver = PS2_STATE_3_PARITY_IN;
                else
                    ns_ps2_receiver = PS2_STATE_2_DATA_IN;
            end
        PS2_STATE_3_PARITY_IN:
            begin
                if (ps2_clk_posedge == 1'b1)
                    ns_ps2_receiver = PS2_STATE_4_STOP_IN;
                else
                    ns_ps2_receiver = PS2_STATE_3_PARITY_IN;
            end
        PS2_STATE_4_STOP_IN:
            begin
                if (ps2_clk_posedge == 1'b1)
                    ns_ps2_receiver = PS2_STATE_0_IDLE;
                else
                    ns_ps2_receiver = PS2_STATE_4_STOP_IN;
            end
        default:
            begin
                ns_ps2_receiver = PS2_STATE_0_IDLE;
            end
        endcase
end

/*****************************************************************************
 *                     Sequential logic                          *
 *****************************************************************************/


always @(posedge clk)
begin
```

```
        if (reset == 1'b1)
                data_count        <= 3'h0;
        else if ((s_ps2_receiver == PS2_STATE_2_DATA_IN) &&
                        (ps2_clk_posedge == 1'b1))
                data_count        <= data_count + 3'h1;
        else if (s_ps2_receiver != PS2_STATE_2_DATA_IN)
                data_count        <= 3'h0;
end

always @(posedge clk)
begin
        if (reset == 1'b1)
                data_shift_reg                    <= 8'h00;
        else if ((s_ps2_receiver == PS2_STATE_2_DATA_IN) &&
                        (ps2_clk_posedge == 1'b1))
                data_shift_reg   <= {ps2_data, data_shift_reg[7:1]};
end

always @(posedge clk)
begin
        if (reset == 1'b1)
                received_data              <= 8'h00;
        else if (s_ps2_receiver == PS2_STATE_4_STOP_IN)
                received_data    <= data_shift_reg;
end

always @(posedge clk)
begin
        if (reset == 1'b1)
                received_data_en                  <= 1'b0;
        else if ((s_ps2_receiver == PS2_STATE_4_STOP_IN) &&
                        (ps2_clk_posedge == 1'b1))
                received_data_en          <= 1'b1;
        else
                received_data_en          <= 1'b0;
end

/*****************************************************************************
 *                           Combinational logic                            *
 *****************************************************************************/


/*****************************************************************************
 *                             Internal Modules                             *
 *****************************************************************************/
```

```
endmodule
```

B.5.3 Altera_UP_PS2_Command_Out

```
/*****************************************************************************
 *                                                                           *
 * Module:       Altera_UP_PS2_Command_Out                                   *
 * Description:                                                              *
 *      This module sends commands out to the PS2 core.                      *
 *                                                                           *
 *****************************************************************************/


module Altera_UP_PS2_Command_Out (
	// Inputs
	clk,
	reset,

	the_command,
	send_command,

	ps2_clk_posedge,
	ps2_clk_negedge,

	// Bidirectionals
	PS2_CLK,
	PS2_DAT,

	// Outputs
	command_was_sent,
	error_communication_timed_out
);

/*****************************************************************************
 *                          Parameter Declarations                           *
 *****************************************************************************/

// Timing info for initiating Host-to-Device communication
//   when using a 50MHz system clock
parameter	CLOCK_CYCLES_FOR_101US		= 5050;
parameter	NUMBER_OF_BITS_FOR_101US	= 13;
parameter	COUNTER_INCREMENT_FOR_101US	= 13'h0001;

//parameter	CLOCK_CYCLES_FOR_101US		= 50;
```

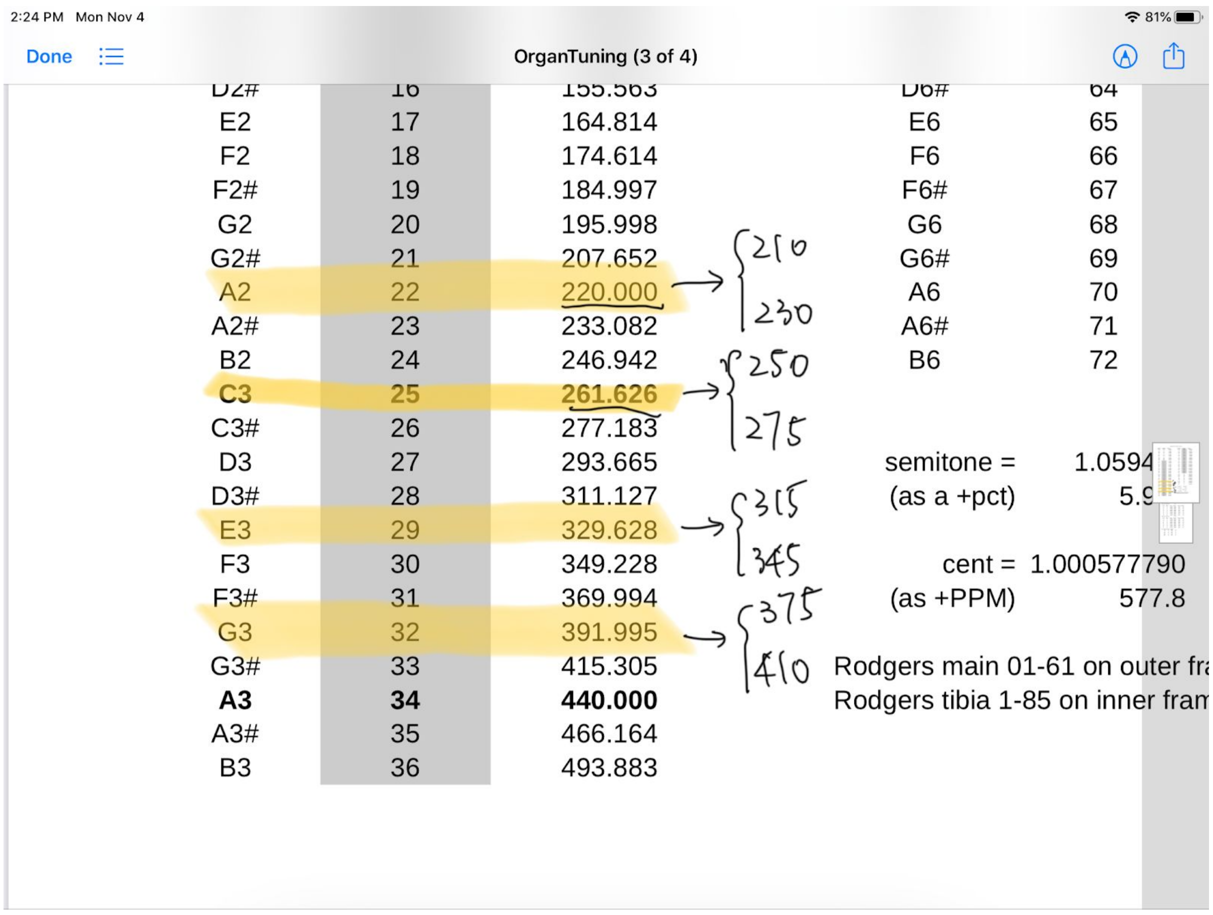
```
//parameter      NUMBER_OF_BITS_FOR_101US        = 6;
//parameter      COUNTER_INCREMENT_FOR_101US           = 6'h01;

// Timing info for start of transmission error
//   when using a 50MHz system clock
parameter        CLOCK_CYCLES_FOR_15MS                 = 750000;
parameter        NUMBER_OF_BITS_FOR_15MS               = 20;
parameter        COUNTER_INCREMENT_FOR_15MS            = 20'h00001;

// Timing info for sending data error
//   when using a 50MHz system clock
parameter        CLOCK_CYCLES_FOR_2MS            = 100000;
parameter        NUMBER_OF_BITS_FOR_2MS                = 17;
parameter        COUNTER_INCREMENT_FOR_2MS  = 17'h00001;

/*****************************************************************************
 *                        Port Declarations                      *
 *****************************************************************************/
// Inputs
input                           clk;
input                           reset;

input            [7:0]     the_command;
input                           send_command;

input                           ps2_clk_posedge;
input                           ps2_clk_negedge;

// Bidirectionals
inout                           PS2_CLK;
inout                           PS2_DAT;

// Outputs
output   reg                    command_was_sent;
output   reg                    error_communication_timed_out;

/*****************************************************************************
 *                       Constant Declarations                     *
 *****************************************************************************/
// states
parameter        PS2_STATE_0_IDLE                                  = 3'h0,
                      PS2_STATE_1_INITIATE_COMMUNICATION         = 3'h1,
                      PS2_STATE_2_WAIT_FOR_CLOCK                 = 3'h2,
                      PS2_STATE_3_TRANSMIT_DATA                  = 3'h3,
                      PS2_STATE_4_TRANSMIT_STOP_BIT              = 3'h4,
                      PS2_STATE_5_RECEIVE_ACK_BIT                = 3'h5,
                    PS2_STATE_6_COMMAND_WAS_SENT            = 3'h6,
                    PS2_STATE_7_TRANSMISSION_ERROR          = 3'h7;

/*****************************************************************************
 *              Internal wires and registers Declarations              *
 *****************************************************************************/
// Internal Wires

// Internal Registers
reg                 [3:0]   cur_bit;
reg                 [8:0]   ps2_command;

reg                 [NUMBER_OF_BITS_FOR_101US:1]  command_initiate_counter;

reg                 [NUMBER_OF_BITS_FOR_15MS:1]         waiting_counter;
reg                 [NUMBER_OF_BITS_FOR_2MS:1]          transfer_counter;

// State Machine Registers
reg                 [2:0]   ns_ps2_transmitter;
reg                 [2:0]   s_ps2_transmitter;

/*****************************************************************************
 *                  Finite State Machine(s)                  *
 *****************************************************************************/

always @(posedge clk)
begin
        if (reset == 1'b1)
                s_ps2_transmitter <= PS2_STATE_0_IDLE;
        else
                s_ps2_transmitter <= ns_ps2_transmitter;
end

always @(*)
begin
        // Defaults
        ns_ps2_transmitter = PS2_STATE_0_IDLE;

   case (s_ps2_transmitter)
        PS2_STATE_0_IDLE:
                begin
                        if (send_command == 1'b1)
                                ns_ps2_transmitter =
PS2_STATE_1_INITIATE_COMMUNICATION;
                        else
                                ns_ps2_transmitter = PS2_STATE_0_IDLE;
```

```
		end
	PS2_STATE_1_INITIATE_COMMUNICATION:
		begin
			if (command_initiate_counter == CLOCK_CYCLES_FOR_101US)
				ns_ps2_transmitter = PS2_STATE_2_WAIT_FOR_CLOCK;
			else
				ns_ps2_transmitter =
PS2_STATE_1_INITIATE_COMMUNICATION;
		end
	PS2_STATE_2_WAIT_FOR_CLOCK:
		begin
			if (ps2_clk_negedge == 1'b1)
				ns_ps2_transmitter = PS2_STATE_3_TRANSMIT_DATA;
			else if (waiting_counter == CLOCK_CYCLES_FOR_15MS)
				ns_ps2_transmitter = PS2_STATE_7_TRANSMISSION_ERROR;
			else
				ns_ps2_transmitter = PS2_STATE_2_WAIT_FOR_CLOCK;
		end
	PS2_STATE_3_TRANSMIT_DATA:
		begin
			if ((cur_bit == 4'd8) && (ps2_clk_negedge == 1'b1))
				ns_ps2_transmitter = PS2_STATE_4_TRANSMIT_STOP_BIT;
			else if (transfer_counter == CLOCK_CYCLES_FOR_2MS)
				ns_ps2_transmitter = PS2_STATE_7_TRANSMISSION_ERROR;
			else
				ns_ps2_transmitter = PS2_STATE_3_TRANSMIT_DATA;
		end
	PS2_STATE_4_TRANSMIT_STOP_BIT:
		begin
			if (ps2_clk_negedge == 1'b1)
				ns_ps2_transmitter = PS2_STATE_5_RECEIVE_ACK_BIT;
			else if (transfer_counter == CLOCK_CYCLES_FOR_2MS)
				ns_ps2_transmitter = PS2_STATE_7_TRANSMISSION_ERROR;
			else
				ns_ps2_transmitter = PS2_STATE_4_TRANSMIT_STOP_BIT;
		end
	PS2_STATE_5_RECEIVE_ACK_BIT:
		begin
			if (ps2_clk_posedge == 1'b1)
				ns_ps2_transmitter = PS2_STATE_6_COMMAND_WAS_SENT;
			else if (transfer_counter == CLOCK_CYCLES_FOR_2MS)
				ns_ps2_transmitter = PS2_STATE_7_TRANSMISSION_ERROR;
			else
				ns_ps2_transmitter = PS2_STATE_5_RECEIVE_ACK_BIT;
		end
	PS2_STATE_6_COMMAND_WAS_SENT:
				begin
					if (send_command == 1'b0)
						ns_ps2_transmitter = PS2_STATE_0_IDLE;
					else
						ns_ps2_transmitter = PS2_STATE_6_COMMAND_WAS_SENT;
				end
			PS2_STATE_7_TRANSMISSION_ERROR:
				begin
					if (send_command == 1'b0)
						ns_ps2_transmitter = PS2_STATE_0_IDLE;
					else
						ns_ps2_transmitter = PS2_STATE_7_TRANSMISSION_ERROR;
				end
			default:
				begin
					ns_ps2_transmitter = PS2_STATE_0_IDLE;
				end
			endcase
		end

/*****************************************************************************
 *                             Sequential logic                              *
 *****************************************************************************/

always @(posedge clk)
begin
	if (reset == 1'b1)
		ps2_command <= 9'h000;
	else if (s_ps2_transmitter == PS2_STATE_0_IDLE)
		ps2_command <= {(^the_command) ^ 1'b1, the_command};
end

always @(posedge clk)
begin
	if (reset == 1'b1)
		command_initiate_counter <= {NUMBER_OF_BITS_FOR_101US{1'b0}};
	else if ((s_ps2_transmitter == PS2_STATE_1_INITIATE_COMMUNICATION) &&
			(command_initiate_counter != CLOCK_CYCLES_FOR_101US))
		command_initiate_counter <=
			command_initiate_counter + COUNTER_INCREMENT_FOR_101US;
	else if (s_ps2_transmitter != PS2_STATE_1_INITIATE_COMMUNICATION)
		command_initiate_counter <= {NUMBER_OF_BITS_FOR_101US{1'b0}};
end

always @(posedge clk)
begin
```

```
        if (reset == 1'b1)
                waiting_counter <= {NUMBER_OF_BITS_FOR_15MS{1'b0}};
        else if ((s_ps2_transmitter == PS2_STATE_2_WAIT_FOR_CLOCK) &&
                        (waiting_counter != CLOCK_CYCLES_FOR_15MS))
                waiting_counter <= waiting_counter + COUNTER_INCREMENT_FOR_15MS;
        else if (s_ps2_transmitter != PS2_STATE_2_WAIT_FOR_CLOCK)
                waiting_counter <= {NUMBER_OF_BITS_FOR_15MS{1'b0}};
end

always @(posedge clk)
begin
        if (reset == 1'b1)
                transfer_counter <= {NUMBER_OF_BITS_FOR_2MS{1'b0}};
        else
        begin
                if ((s_ps2_transmitter == PS2_STATE_3_TRANSMIT_DATA) ||
                        (s_ps2_transmitter == PS2_STATE_4_TRANSMIT_STOP_BIT) ||
                        (s_ps2_transmitter == PS2_STATE_5_RECEIVE_ACK_BIT))
                begin
                        if (transfer_counter != CLOCK_CYCLES_FOR_2MS)
                                transfer_counter <= transfer_counter +
COUNTER_INCREMENT_FOR_2MS;
                end
                else
                        transfer_counter <= {NUMBER_OF_BITS_FOR_2MS{1'b0}};
        end
end

always @(posedge clk)
begin
        if (reset == 1'b1)
                cur_bit <= 4'h0;
        else if ((s_ps2_transmitter == PS2_STATE_3_TRANSMIT_DATA) &&
                        (ps2_clk_negedge == 1'b1))
                cur_bit <= cur_bit + 4'h1;
        else if (s_ps2_transmitter != PS2_STATE_3_TRANSMIT_DATA)
                cur_bit <= 4'h0;
end

always @(posedge clk)
begin
        if (reset == 1'b1)
                command_was_sent <= 1'b0;
        else if (s_ps2_transmitter == PS2_STATE_6_COMMAND_WAS_SENT)
                command_was_sent <= 1'b1;
        else if (send_command == 1'b0)
```

```
                              command_was_sent <= 1'b0;
end

always @(posedge clk)
begin
          if (reset == 1'b1)
                    error_communication_timed_out <= 1'b0;
          else if (s_ps2_transmitter == PS2_STATE_7_TRANSMISSION_ERROR)
                    error_communication_timed_out <= 1'b1;
          else if (send_command == 1'b0)
                    error_communication_timed_out <= 1'b0;
end

/*****************************************************************************
 *                       Combinational logic                     *
 *****************************************************************************/

assign PS2_CLK           =
          (s_ps2_transmitter == PS2_STATE_1_INITIATE_COMMUNICATION) ?
                    1'b0 :
                    1'bz;

assign PS2_DAT           =
          (s_ps2_transmitter == PS2_STATE_3_TRANSMIT_DATA) ? ps2_command[cur_bit] :
          (s_ps2_transmitter == PS2_STATE_2_WAIT_FOR_CLOCK) ? 1'b0 :
          ((s_ps2_transmitter == PS2_STATE_1_INITIATE_COMMUNICATION) &&
                    (command_initiate_counter[NUMBER_OF_BITS_FOR_101US] == 1'b1)) ? 1'b0 :
                              1'bz;

/*****************************************************************************
 *                         Internal Modules                      *
 *****************************************************************************/


endmodule
```

Appendix C



2:24 PM Mon Nov 4

Done

OrganTuning (3 of 4)

D2#	16	155.563		D6#	64
E2	17	164.814		E6	65
F2	18	174.614		F6	66
F2#	19	184.997		F6#	67
G2	20	195.998		G6	68
G2#	21	207.652	210	G6#	69
A2	22	220.000	→	A6	70
A2#	23	233.082	230	A6#	71
B2	24	246.942	250	B6	72
C3	25	261.626	→		
C3#	26	277.183	275		
D3	27	293.665			
D3#	28	311.127	315		
E3	29	329.628	→		
F3	30	349.228	345		
F3#	31	369.994	375		
G3	32	391.995	→		
G3#	33	415.305	410		
A3	**34**	**440.000**			
A3#	35	466.164			
B3	36	493.883			

semitone = 1.0594
(as a +pct) 5.9

cent = 1.000577790
(as +PPM) 577.8

Rodgers main 01-61 on outer fra
Rodgers tibia 1-85 on inner fram

Table 1 (frequency)

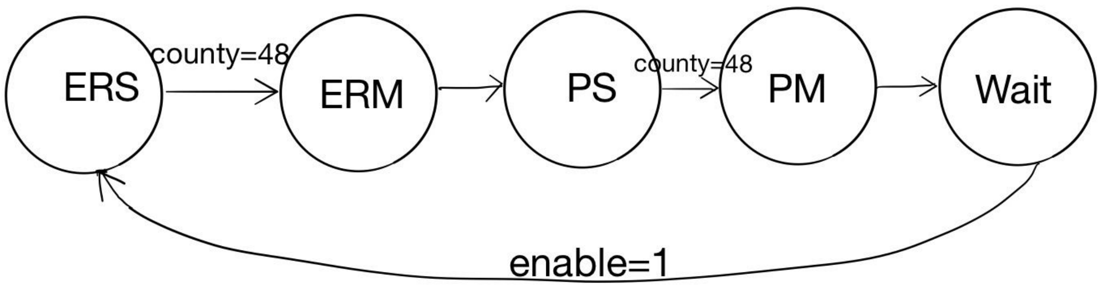


Table 2 (state diagram)

Reference

1. Eecg.toronto.edu. (2019). *PS/2 Controller*. [online] Available at: http://www.eecg.toronto.edu/~jayar/ece241_08F/AudioVideoCores/ps2/ps2.html [Accessed 2 Dec. 2019].

2. Eecg.toronto.edu. (2019). *Audio and Video-in configuration module*. [online] Available at: http://www.eecg.toronto.edu/~jayar/ece241_08F/AudioVideoCores/avconf/avconf.html [Accessed 2 Dec. 2019].
3. Eecg.toronto.edu. (2019). *Audio Controller*. [online] Available at: http://www.eecg.toronto.edu/~jayar/ece241_08F/AudioVideoCores/audio/audio.html [Accessed 2 Dec. 2019].