```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:24:22 04/16/2007
// Design Name:
// Module Name:    audio_codec
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module audio_codec(clock, reset, frame_enable, audio_in_left, audio_in_right,
audio_out_left, audio_out_right,
                           audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
ac97_bit_clock);

      input  clock;
      input  reset;
      output frame_enable;
      output [17:0] audio_in_left;
      output [17:0] audio_in_right;
      input  [17:0] audio_out_left;
      input  [17:0] audio_out_right;

      output audio_reset_b;
      output ac97_sdata_out;
      input  ac97_sdata_in;
      output ac97_synch;
      input  ac97_bit_clock;

      reg frame_enable;
      reg [19:0] audio_in_left0;
      reg [19:0] audio_in_right0;
      reg [17:0] audio_in_left;
      reg [17:0] audio_in_right;
      reg [19:0] audio_out_left0;
      reg [19:0] audio_out_right0;

      reg audio_reset_b;
      reg ac97_sdata_out;
      reg ac97_synch;

      reg [3:0] frame_count;
      reg [7:0] bit_count;

      reg [23:0] command;
      wire [19:0] command_address;
      wire [19:0] command_data;

      reg [7:0] reset_count;
      reg initializing;

      /*initial begin
            reset_count = 0;
```

```
      // synthesis attribute init of reset_count is "00";
      audio_reset_b = 1'b0;
      // synthesis attribute init of audio_reset_b is "0";
      initializing = 1'b1;
      // synthesis attribute init of initializing is "1";
end*/

always @(posedge clock) begin
      if(reset) begin
            reset_count <= 0;
            audio_reset_b <= 0;
      end else begin
            if(reset_count == 255)
                  audio_reset_b <= 1;
            else
                  reset_count <= reset_count + 1;
      end
end

reg audio_in_frame_valid;
reg audio_in_left_valid;
reg audio_in_right_valid;

/*initial begin
      bit_count = 8'h00;
      // synthesis attribute init of bit_count is "00";
      frame_count = 4'h0;
      // synthesis attribute init of frame_count is "0";
      frame_enable = 1'b0;
      // synthesis attribute init of frame_enable is "0";
      audio_in_left0 = 20'b0;
      // synthesis attribute init of audio_in_left0 is "00000";
      audio_in_right0 = 20'b0;
      // synthesis attribute init of audio_in_right0 is "00000";
      audio_out_left0 = 20'b0;
      // synthesis attribute init of audio_out_left0 is "00000";
      audio_out_right0 = 20'b0;
      // synthesis attribute init of audio_out_right0 is "00000";
      audio_in_frame_valid = 1'b0;
      // synthesis attribute init of audio_in_frame_valid is "0";
      audio_in_left_valid = 1'b0;
      // synthesis attribute init of audio_in_left_valid is "0";
      audio_in_right_valid = 1'b0;
      // synthesis attribute init of audio_in_right_valid is "0";
end*/

always @(posedge ac97_bit_clock) begin
      if(reset) begin
            bit_count <= 8'b0;
            frame_count <= 4'b0;
            frame_enable <= 0;

            audio_in_left0  <= 20'b0;
            audio_in_left   <= 18'b0;
            audio_in_right0 <= 20'b0;
            audio_in_right  <= 18'b0;
            audio_out_left0  <= 20'b0;
            audio_out_right0 <= 20'b0;

            audio_in_frame_valid <= 0;
            audio_in_left_valid <= 0;
            audio_in_right_valid <= 0;
```

```
                ac97_sdata_out <= 0;
                ac97_synch <= 0;

                initializing <= 1;
          end else begin
                if(bit_count == 255) begin
                      if(frame_count == 11)
                            initializing <= 0;

                      frame_count <= frame_count + 1;

                      frame_enable <= 1;

                      audio_in_left  <= audio_in_left0 [18:1];
                      audio_in_right <= audio_in_right0[18:1];
                      audio_out_left0  <= {audio_out_left,  2'b0};
                      audio_out_right0 <= {audio_out_right, 2'b0};
                end else if(bit_count == 16) begin
                      frame_enable <= 0;
                end

                bit_count <= bit_count + 1;

                if(bit_count < 16) begin
                      // Slot 0: Tag bits
                      case(bit_count[3:0])
                            4'h0: ac97_sdata_out <= 1;  // Output frame valid

                            4'h1: begin
                                  ac97_sdata_out <= initializing; // Command address
valid
                                  audio_in_frame_valid <= ac97_sdata_in;  // Input
frame valid
                            end

                            4'h2: ac97_sdata_out <= initializing; // Command data valid

                            4'h3: ac97_sdata_out <= 1;  // Output left channel data
valid

                            4'h4: begin
                                  ac97_sdata_out <= 1;  // Output right channel data
valid
                                  audio_in_left_valid <= ac97_sdata_in;  // Input left
channel valid data
                            end

                            4'h5: audio_in_right_valid <= ac97_sdata_in;  // Input
right channel valid data

                            default: ac97_sdata_out <= 0;
                      endcase
                end else if(bit_count < 36)
                      // Slot 1: Command address
                      ac97_sdata_out <= initializing ? command_address[35 - bit_count]
: 0;
                else if(bit_count < 56)
                      // Slot 2: Command data
                      ac97_sdata_out <= initializing ? command_data[55 - bit_count] :
0;
                else if(bit_count < 76) begin
                      // Slot 3: Left channel data
                      ac97_sdata_out <= audio_out_left0[75 - bit_count];
```

```
                        audio_in_left0[75 - bit_count] <= (audio_in_frame_valid &&
audio_in_left_valid) ? ac97_sdata_in : 0;
                  end else if(bit_count < 96) begin
                        // Slot 4: Right channel data
                        ac97_sdata_out <= audio_out_right0[95 - bit_count];

                        audio_in_right0[95 - bit_count] <= (audio_in_frame_valid &&
audio_in_right_valid) ? ac97_sdata_in : 0;
                  end else
                        ac97_sdata_out <= 0;

                  // Sync signal
                  if(bit_count == 8'd255)
                        ac97_synch <= 1;
                  else if(bit_count == 8'd15)
                        ac97_synch <= 0;
            end
      end

      always @(frame_count) begin
            case(frame_count)
                  4'd0: command = 24'hFC_0000; // Read vendor ID
                  4'd1: command = 24'hFC_0000; // Read vendor ID
                  4'd2: command = 24'hFC_0000; // Read vendor ID
                  4'd3: command = 24'h02_0000; // Unmute line out, no attenuation
                  4'd4: command = 24'h04_0000; // Unmute headphones, no attenuation
                  //4'd5: command = 24'h0E_0000; // Unmute microphone, max volume
                  4'd6: command = 24'h18_0808; // Unmute PCM out, no gain
                  4'd7: command = 24'h1A_0000; // Record select = Mic
                  4'd8: command = 24'h1C_1F1F; // Unmute record gain, no gain //max gain
(+22.5 dB)
                  4'd9: command = 24'h20_0000; // GPR: bypass 3D sound
                  default: command = 24'hFC_0000; // Read vendor ID
            endcase
      end

      // Separate the address and data portions of command and pad them to 20 bits
      assign command_address = {command[23:16], 12'b0};
      assign command_data    = {command[15: 0],  4'b0};

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   15:55:46 04/16/2007
// Design Name:   audio_codec
// Module Name:   U:/Desktop/finalproject/audio_codec_tb.v
// Project Name:  finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: audio_codec
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module audio_codec_tb_v;

	// Inputs
	reg clock;
	reg reset;
	reg [17:0] audio_out_left;
	reg [17:0] audio_out_right;
	reg ac97_sdata_in;
	reg ac97_bit_clock;

	// Outputs
	wire frame_enable;
	wire [17:0] audio_in_left;
	wire [17:0] audio_in_right;
	wire audio_reset_b;
	wire ac97_sdata_out;
	wire ac97_synch;

	// Instantiate the Unit Under Test (UUT)
	audio_codec uut (
		.clock(clock),
		.reset(reset),
		.frame_enable(frame_enable),
		.audio_in_left(audio_in_left),
		.audio_in_right(audio_in_right),
		.audio_out_left(audio_out_left),
		.audio_out_right(audio_out_right),
		.audio_reset_b(audio_reset_b),
		.ac97_sdata_out(ac97_sdata_out),
		.ac97_sdata_in(ac97_sdata_in),
		.ac97_synch(ac97_synch),
		.ac97_bit_clock(ac97_bit_clock)
	);

	always #2 clock = ~clock;
	always #1 ac97_bit_clock = ~ac97_bit_clock;
	always #512 audio_out_left = ~audio_out_left;
	always #512 audio_out_right = ~audio_out_right;
```

```
	reg [7:0] bit_count;
	reg [17:0] frame_count;

	reg prev_ac97_synch;

	always @(posedge ac97_bit_clock)
		prev_ac97_synch <= ac97_synch;

	always @(posedge ac97_bit_clock) begin
		if(ac97_synch && !prev_ac97_synch) begin
			bit_count <= 8'b0;
			frame_count <= frame_count + 1;
		end else
			bit_count <= bit_count + 1;
	end

	always @(bit_count) begin
		if(bit_count == 8'd0 || bit_count == 8'd3 || bit_count == 8'd4)
			ac97_sdata_in = 1;
		else if(bit_count >= 8'd56 && bit_count < 8'd74)
			ac97_sdata_in = frame_count[73 - bit_count];
		else if(bit_count >= 8'd76 && bit_count < 8'd94)
			ac97_sdata_in = frame_count[93 - bit_count];
		else
			ac97_sdata_in = 0;
	end

	initial begin
		// Initialize Inputs
		clock = 0;
		reset = 0;
		audio_out_left = 0;
		audio_out_right = 0;
		ac97_sdata_in = 0;
		ac97_bit_clock = 0;

		frame_count = 18'b0;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here
		reset = 1;
		#100;
		reset = 0;
	end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:05:35 05/04/2007
// Design Name:
// Module Name:    divider
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module divider(clock, reset, rfd, done, dividend, divisor, quotient);

      parameter DIVIDEND_SIZE = 10;
      parameter DIVISOR_SIZE = 10;
      parameter QUOTIENT_FRAC_BITS = 8;
      parameter QUOTIENT_SIZE = DIVIDEND_SIZE + QUOTIENT_FRAC_BITS;
      parameter WORK_SIZE = QUOTIENT_SIZE + 2*DIVISOR_SIZE;

      input clock;
      input reset;
      output rfd;
      output done;
      input [DIVIDEND_SIZE-1:0] dividend;
      input [DIVISOR_SIZE-1:0] divisor;
      output [QUOTIENT_SIZE-1:0] quotient;

      reg done;
      reg [4:0] current_bit;
      reg [WORK_SIZE-1:0] current_value;
      reg [DIVISOR_SIZE-1:0] divisor_latch;
      reg [QUOTIENT_SIZE-1:0] quotient;

      //wire [WORK_SIZE-1:0] next_value;

      //assign next_value = current_value - (divisor_latch << (current_bit +
DIVISOR_SIZE));

      always @(posedge clock) begin
            if(reset) begin
                  current_bit <= 0;
                  current_value <= 0;
                  quotient <= 0;
                  divisor_latch <= 1;
                  done <= 0;
            end else begin
                  if(rfd) begin
                        current_bit <= QUOTIENT_SIZE - 1;
                        current_value <= (dividend << (DIVISOR_SIZE +
QUOTIENT_FRAC_BITS));
                        divisor_latch <= divisor;
                        done <= 1;
                  end else begin
                        current_bit <= current_bit - 1;
```

```
                        done <= 0;
                  end

                  if(current_value >= (divisor_latch << (current_bit + DIVISOR_SIZE)))
begin
                        quotient[current_bit] <= 1;

                        if(!rfd)
                              current_value <= current_value - (divisor_latch <<
(current_bit + DIVISOR_SIZE));
                  end else
                        quotient[current_bit] <= 0;
            end
      end

      assign rfd = (current_bit == 0);

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   15:34:07 05/04/2007
// Design Name:   divider
// Module Name:   U:/Desktop/finalproject3/divider_tb.v
// Project Name:  finalproject3
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: divider
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module divider_tb_v;

	// Inputs
	reg clock;
	reg [9:0] dividend;
	reg [9:0] divisor;

	// Outputs
	wire rfd;
	wire done;
	wire [17:0] quotient;

	// Instantiate the Unit Under Test (UUT)
	divider uut (
		.clock(clock),
		.rfd(rfd),
		.done(done),
		.dividend(dividend),
		.divisor(divisor),
		.quotient(quotient)
	);

	always #1 clock = ~clock;

	initial begin
		// Initialize Inputs
		clock = 0;
		dividend = 0;
		divisor = 0;

		// Wait 100 ns for global reset to finish
		#100;

		uut.current_bit = 0;

		// Add stimulus here
		for(dividend = 100; dividend <= 200; dividend = dividend + 1) begin
			for(divisor = 1; divisor <= 10; divisor = divisor + 1) begin
				#36;
```

```
                end
            end
        end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:49:25 05/07/2007
// Design Name:
// Module Name:    fft_buffer
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module fft_buffer(clock, reset, reset_done, vm_add, vm_index, vm_re, vm_im, fft_index,
fft_re, fft_im);

      input clock;
      input reset;
      output reset_done;
      input vm_add;
      input [9:0] vm_index;
      input [22:0] vm_re;
      input [22:0] vm_im;
      input [9:0] fft_index;
      output [22:0] fft_re;
      output [22:0] fft_im;

      reg resetting;
      reg [8:0] reset_index;
      reg reset_done;

      always @(posedge clock) begin
            if(reset) begin
                  resetting <= 1;
                  reset_index <= 9'b0;
                  reset_done <= 0;
            end else begin
                  if(reset_index == 9'd511) begin
                        resetting <= 0;
                        reset_index <= 9'b0;
                        reset_done <= 1;
                  end else begin
                        if(resetting)
                              reset_index <= reset_index + 1;

                        reset_done <= 0;
                  end
            end
      end

      reg [8:0] vm_index_d1;
      reg [8:0] vm_index_d2;
      reg [22:0] vm_re_d1;
      reg [22:0] vm_im_d1;

      always @(posedge clock) begin
```

```
      vm_index_d1 <= vm_index[8:0];
      vm_index_d2 <= vm_index_d1;
      vm_re_d1 <= vm_re;
      vm_im_d1 <= vm_im;
end

reg [22:0] fft_re;
reg [22:0] fft_im;

//reg [22:0] data_re[511:0];
//reg [22:0] data_im[511:0];

reg data_we;
reg [8:0] write_addr;
reg [22:0] write_re;
reg [22:0] write_im;

wire [8:0] read_addr;
wire [22:0] read_re;
wire [22:0] read_im;

reg [22:0] read_re_d1;
reg [22:0] read_im_d1;

bram_23x512 data_re(
      .clka(clock),
      .wea(data_we),
      .addra(write_addr),
      .dina(write_re),
      .clkb(clock),
      .addrb(read_addr),
      .doutb(read_re));

bram_23x512 data_im(
      .clka(clock),
      .wea(data_we),
      .addra(write_addr),
      .dina(write_im),
      .clkb(clock),
      .addrb(read_addr),
      .doutb(read_im));

wire conjugate;
reg conjugate_d1;
reg conjugate_d2;
wire nyquist;
reg nyquist_d1;
reg nyquist_d2;

assign conjugate = fft_index[9];
assign nyquist = (fft_index == 10'd512);

always @(posedge clock) begin
      conjugate_d1 <= conjugate;
      conjugate_d2 <= conjugate_d1;
      nyquist_d1 <= nyquist;
      nyquist_d2 <= nyquist_d1;
end

reg [22:0] write_re_d1;
reg [22:0] write_im_d1;

always @(posedge clock) begin
```

```
            write_re_d1 <= write_re;
            write_im_d1 <= write_im;
      end

      always @(resetting or reset_index or vm_add or vm_index_d1 or vm_index_d2 or
vm_re_d1 or read_re or vm_im_d1 or read_im or write_re_d1 or write_im_d1) begin
            if(resetting) begin
                  data_we = 1;
                  write_addr = reset_index;
                  write_re = 23'b0;
                  write_im = 23'b0;
            end else begin
                  data_we = vm_add;
                  write_addr = vm_index_d1;

                  if(vm_index_d1 == vm_index_d2) begin
                        write_re = vm_re_d1 + write_re_d1;
                        write_im = vm_im_d1 + write_im_d1;
                  end else begin
                        write_re = vm_re_d1 + read_re;
                        write_im = vm_im_d1 + read_im;
                  end
            end
      end

      assign read_addr = vm_add ? vm_index[8:0] : (conjugate ? -fft_index[8:0] :
fft_index[8:0]);

      always @(posedge clock) begin
            //if(data_we) begin
            //    data_re[write_addr] <= write_re;
            //    data_im[write_addr] <= write_im;
            //end

            //read_re <= data_re[read_addr];
            //read_im <= data_im[read_addr];

            read_re_d1 <= read_re;
            read_im_d1 <= read_im;

            if(nyquist_d2) begin
                  fft_re <= 23'b0;
                  fft_im <= 23'b0;
            end else begin
                  fft_re <= read_re_d1;
                  fft_im <= conjugate_d2 ? -read_im_d1 : read_im_d1;
            end
      end

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   18:24:08 05/09/2007
// Design Name:   forward_fft_module
// Module Name:   U:/Desktop/finalproject3/fft_to_ifft_tb.v
// Project Name:  finalproject3
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: forward_fft_module
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module fft_to_ifft_tb_v;

	// Inputs
	reg clock;
	reg reset;
	reg [17:0] audio_in;
	wire [9:0] fft_index;

	// Outputs
	wire done;
	wire [17:0] fft_re;
	wire [17:0] fft_im;

	// Instantiate the Unit Under Test (UUT)
	forward_fft_module uut (
		.clock(clock),
		.reset(reset),
		.audio_in(audio_in),
		.audio_frame_enable(1'b1),
		.done(done),
		.fft_index(fft_index),
		.fft_re(fft_re),
		.fft_im(fft_im)
	);

	reg [17:0] fft_re_d1;
	reg [17:0] fft_re_d2;
	reg [17:0] fft_im_d1;
	reg [17:0] fft_im_d2;

	always @(posedge clock) begin
		fft_re_d1 <= fft_re;
		fft_re_d2 <= fft_re_d1;
		fft_im_d1 <= fft_im;
		fft_im_d2 <= fft_im_d1;
	end

	reg ifft_start;
	wire ifft_done;
```

```
	wire ifft_edone;
	wire [28:0] ifft_xk_re;

	reg [5:0] done_count;

	always @(posedge clock) begin
		if(done) begin
			done_count <= 6'd1;
			ifft_start <= 0;
		end else if(done_count != 0) begin
			done_count <= done_count + 1;

			if(done_count == 6'd63)
				ifft_start <= 1;
			else
				ifft_start <= 0;
		end else
			ifft_start <= 0;
	end

	wire [17:0] audio_out;

	inverse_fft_module ifft(
		.clock(clock),
		.reset(reset),
		.start(ifft_start),
		.done(ifft_done),
		.edone(ifft_edone),
		.fft_xk_re(ifft_xk_re),
		.audio_frame_enable(1'b1),
		.fft_index(fft_index),
		.fft_re(fft_re_d2),
		.fft_im(fft_im_d2),
		.audio_out(audio_out));

	always #1 clock = ~clock;

	always @(posedge clock)
		audio_in <= audio_in + 1;

	initial begin
		// Initialize Inputs
		clock = 0;
		reset = 0;
		audio_in = 0;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here
		reset = 1;
		#10;
		reset = 0;
		uut.frame_count = 10'd1020;
	end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:12:50 05/09/2007
// Design Name:
// Module Name:    forward_fft_module
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module forward_fft_module(clock, reset, downsample, audio_in, audio_frame_enable, done,
fft_index, fft_re, fft_im);

      input clock;
      input reset;
      input downsample;
      input [17:0] audio_in;
      input audio_frame_enable;
      output done;
      input [9:0] fft_index;
      output [22:0] fft_re;
      output [22:0] fft_im;

      reg [22:0] fft_re;
      reg [22:0] fft_im;
      reg done;

      reg fft_start;
      wire [17:0] data_re;
      reg [17:0] data_re_d1;
      reg [17:0] data_re_d2;
      reg fwd_inv_we;

      reg [12:0] frame_count;

      //reg [17:0] audio_buffer[1023:0];
      reg [22:0] fft_re_buffer[1023:0];
      reg [22:0] fft_im_buffer[1023:0];

      wire [33:0] xk_re;
      wire [33:0] xk_im;
      wire [9:0] xn_index;
      wire [9:0] xk_index;

      wire fft_rfd;
      wire fft_busy;
      wire fft_dv;
      wire fft_edone;
      wire fft_done;

      bram_18x1024 audio_buffer(
            .clka(clock),
            .wea(~reset && audio_frame_enable && ~|frame_count[2:0]),
```

```
	.addra(frame_count[12:3]),
	.dina(audio_in),
	.clkb(clock),
	.addrb(xn_index),
	.doutb(data_re));

fourier_transform fft(
	.xn_re({data_re_d2, 5'b0}),
	.xn_im(23'b0),
	.start(fft_start),
	.unload(1'b1),
	.fwd_inv(1'b1),
	.fwd_inv_we(fwd_inv_we),
	.clk(clock),
	.xk_re(xk_re),
	.xk_im(xk_im),
	.xn_index(xn_index),
	.xk_index(xk_index),
	.rfd(fft_rfd),
	.busy(fft_busy),
	.dv(fft_dv),
	.edone(fft_edone),
	.done(fft_done));

//assign done = fft_done;

always @(posedge clock) begin
	if(reset)
		fwd_inv_we <= 1;
	else
		fwd_inv_we <= 0;
end

wire [12:0] next_frame_count;
assign next_frame_count = frame_count + (downsample ? 1 : 8);

always @(posedge clock) begin
	if(reset) begin
		fft_start <= 0;
		frame_count <= 13'b0;
		//data_re <= 18'b0;
	end else if(audio_frame_enable) begin
		frame_count <= next_frame_count;

		if(next_frame_count == 13'b0)
			fft_start <= 1;
		else
			fft_start <= 0;

		//audio_buffer[frame_count] <= audio_in;
		//data_re <= audio_buffer[xn_index];
	end else
		fft_start <= 0;
end

always @(posedge clock) begin
	data_re_d1 <= data_re;
	data_re_d2 <= data_re_d1;
end

always @(posedge clock) begin
	if(fft_dv) begin
		fft_re_buffer[xk_index] <= xk_re[32:10] + xk_re[9];
```

```
                fft_im_buffer[xk_index] <= xk_im[32:10] + xk_im[9];
            end

            fft_re <= fft_re_buffer[fft_index];
            fft_im <= fft_im_buffer[fft_index];
        end

        reg done_reg;

        always @(posedge clock) begin
            if(reset) begin
                done_reg <= 0;
                done <= 0;
            end else begin
                if(fft_done)
                    done_reg <= 1;
                else if(done_reg && fft_dv && xk_index == 10'd15) begin
                    done_reg <= 0;
                    done <= 1;
                end else
                    done <= 0;
            end
        end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:50:50 04/29/2007
// Design Name:
// Module Name:    ifft_tester
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//   This module tests the inverse Fourier transform module.  This generates a
//   constant vector representing the Fourier transform of a major chord sampled
//   at 48kHz (frequency indices 8, 10, and 12, corresponding to 375 Hz, 468.75 Hz,
//   and 562.5 Hz, which are approximately F#4, A#4, and C#5).
//
//////////////////////////////////////////////////////////////////////////////////
module ifft_tester(clock, reset, switch, frame_enable, ifft_start, index, fft_re,
fft_im);
      input clock;
      input reset;
      input [7:0] switch;
      input frame_enable;
      output ifft_start;
      input [9:0] index;

      output [17:0] fft_re;
      output [17:0] fft_im;

      reg ifft_start;
      reg [17:0] fft_re;
      reg [17:0] fft_im;

      reg [9:0] frame_count;

      always @(posedge clock) begin
            if(reset) begin
                  frame_count <= 10'b0;
                  ifft_start <= 0;
            end else begin
                  if(frame_enable) begin
                        frame_count <= frame_count + 1;

                        if(frame_count == 10'd1023)
                              ifft_start <= 1;
                        else
                              ifft_start <= 0;
                  end else
                        ifft_start <= 0;
            end
      end

      always @(posedge clock) begin
            case(index)
                  8,  1016: fft_re <= switch[7] ? 18'h07FFF : 18'h0;
                  9,  1015: fft_re <= switch[6] ? 18'h07FFF : 18'h0;
```

```
                10, 1014: fft_re <= switch[5] ? 18'h07FFF : 18'h0;
                11, 1013: fft_re <= switch[4] ? 18'h07FFF : 18'h0;
                12, 1012: fft_re <= switch[3] ? 18'h07FFF : 18'h0;
                13, 1011: fft_re <= switch[2] ? 18'h07FFF : 18'h0;
                14, 1010: fft_re <= switch[1] ? 18'h07FFF : 18'h0;
                15, 1009: fft_re <= switch[0] ? 18'h07FFF : 18'h0;

                default: fft_re <= 18'h0;
        endcase
    end

    always @(fft_re) begin
        if(~index[9])
            fft_im = fft_re;
        else
            fft_im = -fft_re;
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:48:34 04/29/2007
// Design Name:
// Module Name:    inverse_fft_module
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module inverse_fft_module(clock, reset, start, done, edone, fft_xk_re,
audio_frame_enable, fft_index, fft_re, fft_im, clamp, audio_out);

      input clock;
      input reset;
      input start;
      output done;
      output edone;
      output [33:0] fft_xk_re;
      input audio_frame_enable;
      output [9:0] fft_index;
      input [22:0] fft_re;
      input [22:0] fft_im;
      input clamp;
      output [17:0] audio_out;

      wire [9:0] fft_index;
      //reg [17:0] audio_out;

      reg ifft_fwd_inv_we;

      reg [9:0] frame_count;

      //reg audio_frame_enable_d1;

      //always @(posedge clock)
      //    audio_frame_enable_d1 <= audio_frame_enable;

      //reg [17:0] audio_data[1023:0];
      /*reg [17:0] audio_sample1;
      wire [17:0] audio_data_out;
      reg [21:0] sample1_weighted;
      reg [21:0] sample2_weighted;
      wire [18:0] sample_weighted_sum;

      always @(posedge clock) begin
            sample1_weighted <= {~audio_sample1[17], audio_sample1[16:0]} * (4'b1000 -
{1'b0, frame_count[2:0]});
            sample2_weighted <= {~audio_data_out[17], audio_data_out[16:0]} * {1'b0,
frame_count[2:0]};
      end

      assign sample_weighted_sum = (sample1_weighted + sample2_weighted) >> 3;//*/
```

```
wire [33:0] data_re;
reg [17:0] data_re_clamped;
wire [33:0] data_im;
wire [9:0] data_index;

wire ifft_rfd;  // Ready for data
wire ifft_busy;
wire ifft_dv;   // Data valid
wire ifft_edone; // Early done (goes high one clock cycle prior to the done signal)
wire ifft_done;

bram_18x1024 audio_data(
      .clka(clock),
      .wea(ifft_dv),
      .addra(data_index),
      .dina(data_re_clamped),
      .clkb(clock),
      .addrb(frame_count),
      .doutb(audio_out));

// Inverse FFT module
fourier_transform ifft(
      .xn_re(fft_re),
      .xn_im(fft_im),
      .start(start),
      .unload(1'b1),
      .fwd_inv(1'b0),
      .fwd_inv_we(ifft_fwd_inv_we),
      .clk(clock),
      .xk_re(data_re),
      .xk_im(data_im),
      .xn_index(fft_index),
      .xk_index(data_index),
      .rfd(ifft_rfd),
      .busy(ifft_busy),
      .dv(ifft_dv),
      .edone(ifft_edone),
      .done(ifft_done));

assign done = ifft_done;
assign edone = ifft_edone;
assign fft_xk_re = data_re;

always @(posedge clock) begin
      if(reset)
            ifft_fwd_inv_we <= 1;
      else
            ifft_fwd_inv_we <= 0;
end

reg prev_ifft_dv;

always @(posedge clock)
      prev_ifft_dv <= ifft_dv;

// Clamp data_re[17:0] to (-2^17, 2^17-1)
always @(data_re or clamp) begin
      if(clamp) begin
            if(!data_re[33] && |data_re[32:24])
                  data_re_clamped = 18'h1FFFF;
            else if(data_re[33] && ~&data_re[32:24])
                  data_re_clamped = 18'h20000;
```

```
                else
                        data_re_clamped = data_re[23:6] + data_re[5];
          end else
                data_re_clamped = data_re[23:6] + data_re[5];
    end

    //always @(posedge clock) begin
    //    if(ifft_dv)
    //          audio_data[data_index] <= data_re_clamped;
    //end

    reg done_reg;

    always @(posedge clock) begin
          if(reset) begin
                frame_count <= 10'b0;
                done_reg <= 0;
          end else begin
                if(ifft_done)
                      done_reg <= 1;
                else if(audio_frame_enable) begin
                      if(done_reg && prev_ifft_dv) begin
                            frame_count <= 0;
                            done_reg <= 0;
                      end else begin
                            frame_count <= frame_count + 1;
                      end
                end //else if(audio_frame_enable_d1) begin
                      //if(frame_count[2:0] == 3'b0)
                      //    audio_sample1 <= audio_data_out;
                      //
                      //audio_out <= {~sample_weighted_sum[17],
sample_weighted_sum[16:0]};
                //end
          end
    end

    //always @(posedge clock) begin
    //    if(reset)
    //          audio_out <= 18'b0;
    //    else if(audio_frame_enable)
    //          audio_out <= audio_data[frame_count];
    //end
endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:42:36 04/30/2007
// Design Name:   inverse_fft_module
// Module Name:   U:/Desktop/finalproject/inverse_fft_tb.v
// Project Name:  finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: inverse_fft_module
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module inverse_fft_tb_v;

	// Inputs
	reg clock;
	reg reset;
	reg start;
	reg audio_frame_enable;
	reg [17:0] fft_re;
	reg [17:0] fft_im;

	// Outputs
	wire [9:0] fft_index;
	wire [17:0] audio_out;

	// Instantiate the Unit Under Test (UUT)
	inverse_fft_module uut (
		.clock(clock),
		.reset(reset),
		.start(start),
		.audio_frame_enable(audio_frame_enable),
		.fft_index(fft_index),
		.fft_re(fft_re),
		.fft_im(fft_im),
		.audio_out(audio_out)
	);

	always #1 clock = ~clock;

	reg [0:0] bit_count;
	reg [9:0] frame_count;

	always @(posedge clock)
		bit_count <= bit_count + 1;

	always @(bit_count)
		audio_frame_enable = (bit_count == 0);

	always @(posedge clock) begin
		if(audio_frame_enable) begin
```

```
                frame_count <= frame_count + 1;

                if(frame_count == 10'h3FF)
                        start <= 1;
                else
                        start <= 0;
        end else
                start <= 0;
    end

    reg [7:0] switch;

    always @(fft_index) begin
        case(fft_index)
                11, 1019: fft_re = switch[7] ? 18'h07FFF : 18'h0;
                12, 1018: fft_re = switch[6] ? 18'h07FFF : 18'h0;
                13, 1017: fft_re = switch[5] ? 18'h07FFF : 18'h0;
                14, 1016: fft_re = switch[4] ? 18'h07FFF : 18'h0;
                15, 1015: fft_re = switch[3] ? 18'h07FFF : 18'h0;
                16, 1014: fft_re = switch[2] ? 18'h07FFF : 18'h0;
                17, 1013: fft_re = switch[1] ? 18'h07FFF : 18'h0;
                18, 1012: fft_re = switch[0] ? 18'h07FFF : 18'h0;

                default: fft_re = 18'h0;
        endcase
    end
    //always @(fft_index)
    //    fft_re = (fft_index == 11 || fft_index == 1019) ? 18'h0FFFF : 18'h00000;

    initial begin
        // Initialize Inputs
        clock = 0;
        reset = 0;
        start = 0;
        audio_frame_enable = 0;
        fft_re = 0;
        fft_im = 0;
        bit_count = 8'b0;
        frame_count = 10'h3C0;
        switch = 8'b0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

        reset = 1;
        #10;
        reset = 0;

        switch = 8'b10000000;
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:39:27 05/15/2007
// Design Name:
// Module Name:    key_frequency_table
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module key_frequency_table(key, freq);

      input [5:0] key;
      output [12:0] freq;

      reg freq;

      always @(key) begin
            case(key)
                  0:  freq = 22;
                  1:  freq = 24;
                  2:  freq = 25;
                  3:  freq = 27;
                  4:  freq = 28;
                  5:  freq = 30;
                  6:  freq = 32;
                  7:  freq = 33;
                  8:  freq = 35;
                  9:  freq = 38;
                  10: freq = 40;
                  11: freq = 42;
                  12: freq = 45;
                  13: freq = 47;
                  14: freq = 50;
                  15: freq = 53;
                  16: freq = 56;
                  17: freq = 60;
                  18: freq = 63;
                  19: freq = 67;
                  20: freq = 71;
                  21: freq = 75;
                  22: freq = 80;
                  23: freq = 84;
                  24: freq = 89;
                  25: freq = 95;
                  26: freq = 100;
                  27: freq = 106;
                  28: freq = 113;
                  29: freq = 119;
                  30: freq = 126;
                  31: freq = 134;
                  32: freq = 142;
                  33: freq = 150;
```

```
34: freq = 159;
35: freq = 169;
36: freq = 179;
37: freq = 189;
38: freq = 200;
39: freq = 212;
40: freq = 225;
41: freq = 238;
42: freq = 253;
43: freq = 268;
44: freq = 284;
45: freq = 300;
46: freq = 318;
47: freq = 337;

/*0:  freq = 10;
1:  freq = 11;
2:  freq = 12;
3:  freq = 13;
4:  freq = 14;
5:  freq = 15;
6:  freq = 16;
7:  freq = 17;
8:  freq = 18;
9:  freq = 19;
10: freq = 20;
11: freq = 21;
12: freq = 22;
13: freq = 24;
14: freq = 25;
15: freq = 27;
16: freq = 28;
17: freq = 30;
18: freq = 32;
19: freq = 33;
20: freq = 35;
21: freq = 38;
22: freq = 40;
23: freq = 42;
24: freq = 45;
25: freq = 47;
26: freq = 50;
27: freq = 53;
28: freq = 56;
29: freq = 60;
30: freq = 63;
31: freq = 67;
32: freq = 71;
33: freq = 75;
34: freq = 80;
35: freq = 84;
36: freq = 89;
37: freq = 95;
38: freq = 100;
39: freq = 106;
40: freq = 113;
41: freq = 119;
42: freq = 126;
43: freq = 134;
44: freq = 142;
45: freq = 150;
46: freq = 159;
47: freq = 169;//*/
```

```
                default: freq = 0;
            endcase
        end

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit
//
//
// Created: March 15, 2007
// Author: Nathan Ickes
//
// This file includes two modules:
//
//   - labkit: the top-level labkit module
//   - debounce: the synchronize/debounce module
//
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
//
// Created: March 15, 2007
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////

module labkit(beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
                  ac97_bit_clock,

                  vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
                  vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
                  vga_out_vsync,

                  tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
                  tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
                  tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

                  tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
                  tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
                  tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
                  tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

                  ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
                  ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

                  ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
                  ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

                  clock_feedback_out, clock_feedback_in,

                  flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
                  flash_reset_b, flash_sts, flash_byte_b,

                  rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

                  mouse_clock, mouse_data, keyboard_clock, keyboard_data,

                  clock_27mhz, clock1, clock2,

                  disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
                  disp_reset_b, disp_data_in,

                  button0, button1, button2, button3, button_enter, button_right,
                  button_left, button_down, button_up,
```

```
            switch,

            led,

            user1, user2, user3, user4,

            daughtercard,

            systemace_data, systemace_address, systemace_ce_b,
            systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

            analyzer1_data, analyzer1_clock,
            analyzer2_data, analyzer2_clock,
            analyzer3_data, analyzer3_clock,
            analyzer4_data, analyzer4_clock);

output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
         vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
         tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
         tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
         tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
         tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
```

```
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
         button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
output [6:0]  systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
input  systemace_irq, systemace_mpbrdy;

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
         analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////////////////////////////////////////
//
// I/O Assignments
//
////////////////////////////////////////////////////////////////////////

// Audio Input and Output
assign beep = 1'b0;
//assign audio_reset_b = 1'b0;
//assign ac97_synch = 1'b0;
//assign ac97_sdata_out = 1'b0;

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'hF;
```

```
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b1;
assign ram1_oe_b = 1'b1;
assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'hF;
assign clock_feedback_out = 1'b0;

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;

// LED Displays
assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
assign disp_data_out = 1'b0;

// Buttons, Switches, and Individual LEDs

// User I/Os
assign user1 = 32'hZ;
assign user2 = 32'hZ;
assign user3 = 32'hZ;
assign user4 = 32'hZ;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;

////////////////////////////////////////////////////////////////////////
//
// Lab Components
//
////////////////////////////////////////////////////////////////////////

//
// Generate a 64.8MHz pixel clock from clock_27mhz
//

wire pclk, pixel_clock;
DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
// synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 3
```

```
	// synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
	// synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
	BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));//*/

	wire clock_buf;
	BUFG clk27mhz_buf(.I(clock_27mhz), .O(clock_buf));

	// Reset synchronizer
	reg reset_d0, reset_d1;
	reg reset_sync;

	always @(posedge clock_buf) begin
		reset_d0 <= ~button0;
		reset_d1 <= reset_d0;
		reset_sync <= reset_d1;
	end

	wire audio_reset;

	debounce audio_reset_debouncer(.clock(clock_buf), .reset(reset_sync),
.noisy(~button1), .clean(audio_reset));

	wire audio_frame_enable;
	wire [17:0] audio_in_left;
	wire [17:0] audio_in_right;
	wire [17:0] audio_out_left;
	wire [17:0] audio_out_right;

	// Audio Codec
	audio_codec the_audio_codec(
		.clock(clock_buf),
		.reset(audio_reset),
		.frame_enable(audio_frame_enable),
		.audio_in_left(audio_in_left),
		.audio_in_right(audio_in_right),
		.audio_out_left(audio_out_left),
		.audio_out_right(audio_out_right),

		.audio_reset_b(audio_reset_b),
		.ac97_sdata_out(ac97_sdata_out),
		.ac97_sdata_in(ac97_sdata_in),
		.ac97_synch(ac97_synch),
		.ac97_bit_clock(ac97_bit_clock));

	reg [31:0] clk_counter;

	reg prev_audio_frame_enable;
	reg real_audio_frame_enable;

	always @(posedge clock_buf) begin
		prev_audio_frame_enable <= audio_frame_enable;

		if(button1)
			real_audio_frame_enable <= audio_frame_enable &&
!prev_audio_frame_enable;
		else
			real_audio_frame_enable <= (clk_counter[1:0] == 0);
	end

	reg [9:0] frame_counter;

	always @(posedge clock_buf) begin
		if(reset_sync)
```

```
                frame_counter <= 10'b0;
            else if(real_audio_frame_enable)
                frame_counter <= frame_counter + 1;
        end

        // Convert stereo to mono
        wire [17:0] audio_in;

        assign audio_in = ~switch[7] ? audio_in_left : ({{4{frame_counter[7]}},
frame_counter[6:0], 7'b0});

        parameter INPUT_MODE_SWITCHES = 2'd0;
        parameter INPUT_MODE_PS2      = 2'd1;
        parameter INPUT_MODE_MIDI     = 2'd2;

        reg [1:0] input_mode;

        wire button_enter_deb;
        reg prev_button_enter_deb;

        debounce button_enter_debouncer(.clock(clock_buf), .reset(reset_sync),
.noisy(~button_enter), .clean(button_enter_deb));

        always @(posedge clock_buf) begin
            prev_button_enter_deb <= button_enter_deb;

            if(reset_sync)
                input_mode <= INPUT_MODE_SWITCHES;
            else if(button_enter_deb && ~prev_button_enter_deb)
                input_mode <= input_mode + 1;
        end

        // PS/2 keyboard stuff
        wire [4:0] ps2_key;
        wire [1:0] octave;
        wire key_state;
        wire ps2_error;
        wire ps2_ready;
        wire key_clear;

        reg [47:0] keys_down;
        wire [5:0] active_key;
        wire active_key_state;
        wire active_ready;

        ps2_controller ps2_ctrl(
            .clock(clock_buf),
            .ps2_clock(keyboard_clock),
            .data(keyboard_data),
            .key(ps2_key),
            .key_state(key_state),
            .octave(octave),
            .error(ps2_error),
            .ready(ps2_ready),
            .clear(key_clear));

        ps2_decoder ps2decoder(
            .clock(clock_buf),
            .key(active_key),
            .key_state(active_key_state),
            .ready(active_ready),
            .ps2_key_state(key_state),
            .ps2_key(ps2_key),
```

```
	.ps2_ready(ps2_ready),
	.octave(octave));

reg [47:0] actual_keys_down;

always @(input_mode or switch or keys_down) begin
	case(input_mode)
		INPUT_MODE_SWITCHES: actual_keys_down = (48'b1 << switch[6:0]);
		INPUT_MODE_PS2:	actual_keys_down = keys_down;
		default: actual_keys_down = 48'b0;
	endcase
end

always @(posedge clock_buf) begin
	if(key_clear)
		keys_down <= 48'b0;
	else if(active_ready)
		keys_down[active_key] <= active_key_state;
end

wire fft_done;
wire voice_mod_start;
wire voice_mod_done;
wire [9:0] fft_index;
wire [22:0] fft_re;
wire [22:0] fft_im;

//wire sa_done;
//wire sa_fft_en;
//wire [9:0] sa_fft_index;
//wire [9:0] input_freq;

wire voice_mod_add;
wire [9:0] voice_mod_in_index;
wire [9:0] voice_mod_out_index;
wire [22:0] voice_mod_out_re;
wire [22:0] voice_mod_out_im;
wire [12:0] output_freq;

wire ifft_buff_reset;
wire ifft_buff_reset_done;
wire ifft_start;
wire ifft_done;
wire ifft_edone;
wire [33:0] ifft_xk_re;
wire [9:0] ifft_fft_index;
wire [22:0] ifft_fft_re;
wire [22:0] ifft_fft_im;

// Forward FFT of audio_in
forward_fft_module fft(
	.clock(clock_buf),
	.reset(reset_sync),
	.downsample(1'b0),
	.audio_in(audio_in),
	.audio_frame_enable(real_audio_frame_enable),
	.done(fft_done),
	.fft_index(fft_index),
	.fft_re(fft_re),
	.fft_im(fft_im));

/*wire fft2_done;
wire [9:0] fft2_index;
```

```
wire [22:0] fft2_re;
wire [22:0] fft2_im;

forward_fft_module fft2(
      .clock(clock_buf),
      .reset(reset_sync),
      .downsample(1'b1),
      .audio_in(audio_in),
      .audio_frame_enable(real_audio_frame_enable),
      .done(fft2_done),
      .fft_index(fft2_index),
      .fft_re(fft2_re),
      .fft_im(fft2_im));//*/

assign fft_index = voice_mod_in_index;
/*assign fft2_index = sa_fft_index;

// Spectrum analyzer (determines fundamental frequency of signal)
spectrum_analyzer analyzer(
      .clock(clock_buf),
      .reset(reset_sync),
      .start(fft2_done),
      .done(sa_done),
      .fft_en(sa_fft_en),
      .fft_index(sa_fft_index),
      .fft_re(fft2_re),
      .fft_im(fft2_im),
      .input_freq(input_freq));//*/

/*/ Inverse FFT tester
ifft_tester ifft_test(
      .clock(clock_buf),
      .reset(reset_sync),
      .switch(8'b00000001),
      .frame_enable(real_audio_frame_enable),
      .ifft_start(fft_done),
      .index(voice_mod_in_index),
      .fft_re(voice_mod_in_re),
      .fft_im(voice_mod_in_im));//*/

voice_modulator_controller voice_mod_ctrl(
      .clock(clock_buf),
      .reset(reset_sync),
      .start(fft_done),
      .keys_down(actual_keys_down),
      .ifft_buff_reset(ifft_buff_reset),
      .ifft_buff_reset_done(ifft_buff_reset_done),
      .vm_start(voice_mod_start),
      .vm_done(voice_mod_done),
      .vm_all_done(ifft_start),
      .vm_out_freq(output_freq));

voice_modulator voice_mod(
      .clock(clock_buf),
      .reset(reset_sync),
      .start(voice_mod_start),
      .done(voice_mod_done),
      .fft_in_index(voice_mod_in_index),
      .fft_in_re(fft_re),
      .fft_in_im(fft_im),
      .input_freq(10'd16),
      .output_freq(output_freq),
      .fft_out_add(voice_mod_add),
```

```
      .fft_out_index(voice_mod_out_index),
      .fft_out_re(voice_mod_out_re),
      .fft_out_im(voice_mod_out_im));

fft_buffer ifft_buff(
      .clock(clock_buf),
      .reset(ifft_buff_reset),
      .reset_done(ifft_buff_reset_done),
      .vm_add(voice_mod_add),
      .vm_index(voice_mod_out_index),
      .vm_re(voice_mod_out_re),
      .vm_im(voice_mod_out_im),
      .fft_index(ifft_fft_index),
      .fft_re(ifft_fft_re),
      .fft_im(ifft_fft_im));

inverse_fft_module ifft(
      .clock(clock_buf),
      .reset(reset_sync),
      .start(ifft_start),
      .done(ifft_done),
      .edone(ifft_edone),
      .fft_xk_re(ifft_xk_re),
      .audio_frame_enable(real_audio_frame_enable),
      .fft_index(ifft_fft_index),
      .fft_re(ifft_fft_re),
      .fft_im(ifft_fft_im),
      .clamp(button2),
      .audio_out(audio_out_left));

assign audio_out_right = audio_out_left;

always @(posedge clock_buf) begin
      if(reset_sync)
            clk_counter <= 0;
      else
            clk_counter <= clk_counter + 1;
end

reg [31:0] ifft_done_times[31:0];
reg [4:0] ifft_next_done_time;
reg [31:0] ifft_done_out;
reg [4:0] ifft_done_out_ptr;

always @(posedge clock_buf) begin
      if(ifft_done)
            ifft_done_times[ifft_next_done_time] <= clk_counter;

      ifft_done_out <= ifft_done_times[ifft_done_out_ptr];
end

always @(posedge clock_buf) begin
      if(ifft_done)
            ifft_next_done_time <= ifft_next_done_time + 1;

      ifft_done_out_ptr <= ifft_done_out_ptr + 1;
end

//assign beep = 1'b0;
//assign audio_reset_b = 1'b0;
//assign ac97_synch = 1'b0;
//assign ac97_sdata_out = 1'b0;
```

```
//
// VGA output signals
//

// Inverting the clock to the DAC provides half a clock period for signals
// to propagate from the FPGA to the DAC.
assign vga_out_pixel_clock = ~pixel_clock;

// The composite sync signal is used to encode sync data in the green
// channel analog voltage for older monitors.  It does not need to be
// implemented for the monitors in the 6.111 lab, and can be left at 1'b1.
assign vga_out_sync_b = 1'b1;
/*
assign vga_out_red = 8'b0;
assign vga_out_green = 8'b0;
assign vga_out_blue = 8'b0;
//assign vga_out_sync_b = 1'b1;
assign vga_out_blank_b = 1'b1;
assign vga_out_pixel_clock = 1'b0;
assign vga_out_hsync = 1'b1;
assign vga_out_vsync = 1'b1;/*/

wire horz_sync;
wire vert_sync;
wire [10:0] pixel_count;
wire [10:0] line_count;

// VGA controller
vga_controller vga_ctrl(
      .clock(pixel_clock),
      .reset(1'b0),
      .blank(vga_out_blank_b),
      .horz_sync(horz_sync),
      .vert_sync(vert_sync),
      .pixel_count(pixel_count),
      .line_count(line_count));

// Parameters for 1024x768, 60Hz display mode
defparam vga_ctrl.WIDTH            = 11'd1024;
defparam vga_ctrl.HORZ_FRONT_PORCH = 11'd24;
defparam vga_ctrl.HORZ_SYNC        = 11'd146;
defparam vga_ctrl.HORZ_BACK_PORCH  = 11'd160;

defparam vga_ctrl.HEIGHT           = 10'd768;
defparam vga_ctrl.VERT_FRONT_PORCH = 10'd3;
defparam vga_ctrl.VERT_SYNC        = 10'd6;
defparam vga_ctrl.VERT_BACK_PORCH  = 10'd29;

// Delay horz_sync and vert_sync by 2 cycles to syncrhonize with DAC
reg horz_sync_d1, horz_sync_d2;
reg vert_sync_d1, vert_sync_d2;

always @(posedge pixel_clock) begin
      horz_sync_d1 <= horz_sync;
      horz_sync_d2 <= horz_sync_d1;

      vert_sync_d1 <= vert_sync;
      vert_sync_d2 <= vert_sync_d1;
end

assign vga_out_hsync = horz_sync_d2;
assign vga_out_vsync = vert_sync_d2;
```

```
   wire [7:0] r1, r2, g1, g2, b1, b2;
   wire [1:0] disp_buff, wave_buff, other_buff;

   trigger_display wd(
   .pixel_clock(pixel_clock),
   .x(pixel_count), .y(line_count),
   .r(r1), .g(g1), .b(b1),
   .bit_clock(clock_buf),
   .wave_enable(real_audio_frame_enable),
   .wave(audio_in),
   .hold(switch[1]),
   .trigger(switch[0]),
   .disp_buff(disp_buff),
   .wave_buff(wave_buff),
   .other_buff(other_buff)
);

defparam wd.Y_MAX = 384;
defparam wd.Y_ZERO = 192;
defparam wd.RED = 8'hFF;
defparam wd.GREEN = 8'hFF;
defparam wd.BLUE = 8'hFF;//*/

trigger_display wd2(
   .pixel_clock(pixel_clock),
   .x(pixel_count), .y(line_count),
   .r(r2), .g(g2), .b(b2),
   .bit_clock(clock_buf),
   .wave_enable(real_audio_frame_enable),
   .wave(audio_out_left),
   .trigger(switch[3]),
   .hold(switch[2])
);
defparam wd2.Y_MIN = 384;
defparam wd2.Y_ZERO = 576;

assign vga_out_red = line_count > 383 ? r2 : r1;
assign vga_out_blue = line_count > 383 ? b2 : b1;
assign vga_out_green = line_count > 383 ? g2 : g1;

   /*
   waveform_viewer the_waveform_viewer(
         .clock(pixel_clock),
         .reset(reset_sync),
         .frame_enable(audio_frame_enable),
         .audio_data_in(audio_in[17:10]),
         .audio_data_out(audio_out_left[17:10]),
         .pixel_count(pixel_count),
         .line_count(line_count),
         .vga_out_red(vga_out_red),
         .vga_out_green(vga_out_green),
         .vga_out_blue(vga_out_blue));*/

   /*reg [6:0] tmp_wave = 7'h0;

   always @(posedge clock_buf)
         tmp_wave <= tmp_wave + switch[7:4];

   wire [7:0] r1, g1, b1, r2, g2, b2;

   wave_display lwave(
         .pixel_clock(pixel_clock),
```

```
            .x(pixel_count),
            .y(line_count),
            .r(r1),
            .g(g1),
            .b(b1),
            .bit_clock(clock_buf),
            .wave_enable(real_audio_frame_enable),
            .wave(audio_out_left),
            .trigger(switch[1]),
            .hold(switch[0]));

      defparam lwave.Y_MAX = 384;
      defparam lwave.Y_ZERO = 192;
      defparam lwave.RED = 8'hF0;
      defparam lwave.GREEN = 8'h55;
      defparam lwave.BLUE = 8'h20;

      wave_display rwave(
            .pixel_clock(pixel_clock),
            .x(pixel_count),
            .y(line_count),
            .r(r2),
            .g(g2),
            .b(b2),
            .bit_clock(clock_buf),
            .wave_enable(real_audio_frame_enable),
            .wave( {tmp_wave, 11'h0} ),
            .trigger(switch[1]),
            .hold(switch[0]));

      defparam rwave.Y_MIN = 384;
      defparam rwave.Y_ZERO = 576;

      wire lregion;
      assign lregion = (line_count < 384);

      assign vga_out_red = lregion ? r1 : r2;
      assign vga_out_blue = lregion ? b1 : b2;
      assign vga_out_green = lregion ? g1 : g2;*/

      reg [3:0] analyzer_mode;
      wire button3_deb;
      reg prev_button3_deb;

      debounce button3_debouncer(.clock(clock_buf), .reset(reset_sync), .noisy(~button3),
.clean(button3_deb));

      always @(posedge clock_buf) begin
            prev_button3_deb <= button3_deb;

            if(reset_sync)
                  analyzer_mode <= 4'b0;
            else if(button3_deb && ~prev_button3_deb)
                  analyzer_mode <= analyzer_mode + 1;
      end

      reg [31:0] analyzer_data;

      always @* begin
            case(analyzer_mode)
                  4'd0:  analyzer_data = {ac97_bit_clock, ac97_synch, audio_reset_b,
ac97_sdata_out, ac97_sdata_in, real_audio_frame_enable, ~audio_in_left[17],
audio_in_left[16:9], ~audio_out_left[17], audio_out_left[16:1]};
```

```
                  4'd1:  analyzer_data = {real_audio_frame_enable, ~audio_out_left[17],
audio_out_left[16:0], output_freq};
                  4'd2:  analyzer_data = {keyboard_clock, keyboard_data, ps2_key,
key_state, octave, ps2_error, ps2_ready, key_clear, active_key, active_ready,
keys_down[47:36]};
                  4'd3:  analyzer_data = keys_down[35:4];
                  //{pixel_clock, vga_out_blank_b, horz_sync, vert_sync, line_count[9:5],
pixel_count[10:6], vga_out_red[7:2], vga_out_green[7:2], vga_out_blue[7:2]};
                  4'd4:  analyzer_data = {ifft_done_out_ptr, ifft_done_out[31:5]};
                  4'd5:  analyzer_data = {clock_buf, ifft_start, ifft_done,
ifft_fft_index, ifft_fft_re[22:4]};
                  4'd6:  analyzer_data = {clock_buf, ifft_start, ifft_done,
ifft_fft_index, ifft_fft_im[22:4]};
                  4'd7:  analyzer_data = {clock_buf, ifft_start, ifft_done,
ifft_xk_re[33:5]};
                  4'd8:  analyzer_data = {clock_buf, fft_done, fft_index, fft_re[22:3]};
                  4'd9:  analyzer_data = {clock_buf, fft_done, fft_index, fft_im[22:3]};
                  4'd10: analyzer_data = {clock_buf, voice_mod_start, voice_mod_done,
voice_mod_add, voice_mod_out_index[9:1], voice_mod_out_re[22:4]};
                  4'd11: analyzer_data = {clock_buf, voice_mod_start, voice_mod_done,
voice_mod_add, voice_mod_out_index[9:1], voice_mod_out_im[22:4]};
            default: analyzer_data = 32'b0;
            endcase
      end

      assign led = 8'hFF;//~input_freq[7:0];

      // Logic Analyzer
      assign analyzer1_data = analyzer_data[31:16];
      assign analyzer1_clock = clock_buf;
      assign analyzer3_data = analyzer_data[15:0];
      assign analyzer3_clock = clock_buf;

      // Unused (no pods)
      assign analyzer2_data = 16'h0;
      assign analyzer2_clock = 1'b1;
      assign analyzer4_data = 16'h0;
      assign analyzer4_clock = 1'b1;

endmodule

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Debounce/Synchronize module
//
//
// Use your system clock for the clock input to produce a synchronous,
// debounced output
//
//////////////////////////////////////////////////////////////////////////////////

module debounce (reset, clock, noisy, clean);
      parameter DELAY = 270000;   // .01 sec with a 27Mhz clock
      input reset, clock, noisy;
      output clean;

      reg [18:0] count;
      reg new, clean;

      always @(posedge clock)
            if(reset) begin
                  count <= 0;
                  new <= noisy;
```

```
                clean <= noisy;
            end else if(noisy != new) begin
                new <= noisy;
                count <= 0;
            end else if(count == DELAY)
                clean <= new;
            else
                count <= count+1;
endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   13:57:07 05/03/2007
// Design Name:   labkit
// Module Name:   U:/Desktop/finalproject2/labkit_tb.v
// Project Name:  finalproject2
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: labkit
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module labkit_tb_v;

      // Inputs
      reg ac97_sdata_in;
      reg ac97_bit_clock;
      reg [19:0] tv_in_ycrcb;
      reg tv_in_data_valid;
      reg tv_in_line_clock1;
      reg tv_in_line_clock2;
      reg tv_in_aef;
      reg tv_in_hff;
      reg tv_in_aff;
      reg clock_feedback_in;
      reg flash_sts;
      reg rs232_rxd;
      reg rs232_cts;
      reg mouse_clock;
      reg mouse_data;
      reg keyboard_clock;
      reg keyboard_data;
      reg clock_27mhz;
      reg clock1;
      reg clock2;
      reg disp_data_in;
      reg button0;
      reg button1;
      reg button2;
      reg button3;
      reg button_enter;
      reg button_right;
      reg button_left;
      reg button_down;
      reg button_up;
      reg [7:0] switch;
      reg systemace_irq;
      reg systemace_mpbrdy;

      // Outputs
      wire beep;
```

```
wire audio_reset_b;
wire ac97_sdata_out;
wire ac97_synch;
wire [7:0] vga_out_red;
wire [7:0] vga_out_green;
wire [7:0] vga_out_blue;
wire vga_out_sync_b;
wire vga_out_blank_b;
wire vga_out_pixel_clock;
wire vga_out_hsync;
wire vga_out_vsync;
wire [9:0] tv_out_ycrcb;
wire tv_out_reset_b;
wire tv_out_clock;
wire tv_out_i2c_clock;
wire tv_out_i2c_data;
wire tv_out_pal_ntsc;
wire tv_out_hsync_b;
wire tv_out_vsync_b;
wire tv_out_blank_b;
wire tv_out_subcar_reset;
wire tv_in_i2c_clock;
wire tv_in_fifo_read;
wire tv_in_fifo_clock;
wire tv_in_iso;
wire tv_in_reset_b;
wire tv_in_clock;
wire [18:0] ram0_address;
wire ram0_adv_ld;
wire ram0_clk;
wire ram0_cen_b;
wire ram0_ce_b;
wire ram0_oe_b;
wire ram0_we_b;
wire [3:0] ram0_bwe_b;
wire [18:0] ram1_address;
wire ram1_adv_ld;
wire ram1_clk;
wire ram1_cen_b;
wire ram1_ce_b;
wire ram1_oe_b;
wire ram1_we_b;
wire [3:0] ram1_bwe_b;
wire clock_feedback_out;
wire [23:0] flash_address;
wire flash_ce_b;
wire flash_oe_b;
wire flash_we_b;
wire flash_reset_b;
wire flash_byte_b;
wire rs232_txd;
wire rs232_rts;
wire disp_blank;
wire disp_data_out;
wire disp_clock;
wire disp_rs;
wire disp_ce_b;
wire disp_reset_b;
wire [7:0] led;
wire [6:0] systemace_address;
wire systemace_ce_b;
wire systemace_we_b;
wire systemace_oe_b;
```

```
wire [15:0] analyzer1_data;
wire analyzer1_clock;
wire [15:0] analyzer2_data;
wire analyzer2_clock;
wire [15:0] analyzer3_data;
wire analyzer3_clock;
wire [15:0] analyzer4_data;
wire analyzer4_clock;

// Bidirs
wire tv_in_i2c_data;
wire [35:0] ram0_data;
wire [35:0] ram1_data;
wire [15:0] flash_data;
wire [31:0] user1;
wire [31:0] user2;
wire [31:0] user3;
wire [31:0] user4;
wire [43:0] daughtercard;
wire [15:0] systemace_data;

// Instantiate the Unit Under Test (UUT)
labkit uut (
      .beep(beep),
      .audio_reset_b(audio_reset_b),
      .ac97_sdata_out(ac97_sdata_out),
      .ac97_sdata_in(ac97_sdata_in),
      .ac97_synch(ac97_synch),
      .ac97_bit_clock(ac97_bit_clock),
      .vga_out_red(vga_out_red),
      .vga_out_green(vga_out_green),
      .vga_out_blue(vga_out_blue),
      .vga_out_sync_b(vga_out_sync_b),
      .vga_out_blank_b(vga_out_blank_b),
      .vga_out_pixel_clock(vga_out_pixel_clock),
      .vga_out_hsync(vga_out_hsync),
      .vga_out_vsync(vga_out_vsync),
      .tv_out_ycrcb(tv_out_ycrcb),
      .tv_out_reset_b(tv_out_reset_b),
      .tv_out_clock(tv_out_clock),
      .tv_out_i2c_clock(tv_out_i2c_clock),
      .tv_out_i2c_data(tv_out_i2c_data),
      .tv_out_pal_ntsc(tv_out_pal_ntsc),
      .tv_out_hsync_b(tv_out_hsync_b),
      .tv_out_vsync_b(tv_out_vsync_b),
      .tv_out_blank_b(tv_out_blank_b),
      .tv_out_subcar_reset(tv_out_subcar_reset),
      .tv_in_ycrcb(tv_in_ycrcb),
      .tv_in_data_valid(tv_in_data_valid),
      .tv_in_line_clock1(tv_in_line_clock1),
      .tv_in_line_clock2(tv_in_line_clock2),
      .tv_in_aef(tv_in_aef),
      .tv_in_hff(tv_in_hff),
      .tv_in_aff(tv_in_aff),
      .tv_in_i2c_clock(tv_in_i2c_clock),
      .tv_in_i2c_data(tv_in_i2c_data),
      .tv_in_fifo_read(tv_in_fifo_read),
      .tv_in_fifo_clock(tv_in_fifo_clock),
      .tv_in_iso(tv_in_iso),
      .tv_in_reset_b(tv_in_reset_b),
      .tv_in_clock(tv_in_clock),
      .ram0_data(ram0_data),
      .ram0_address(ram0_address),
```

```
.ram0_adv_ld(ram0_adv_ld),
.ram0_clk(ram0_clk),
.ram0_cen_b(ram0_cen_b),
.ram0_ce_b(ram0_ce_b),
.ram0_oe_b(ram0_oe_b),
.ram0_we_b(ram0_we_b),
.ram0_bwe_b(ram0_bwe_b),
.ram1_data(ram1_data),
.ram1_address(ram1_address),
.ram1_adv_ld(ram1_adv_ld),
.ram1_clk(ram1_clk),
.ram1_cen_b(ram1_cen_b),
.ram1_ce_b(ram1_ce_b),
.ram1_oe_b(ram1_oe_b),
.ram1_we_b(ram1_we_b),
.ram1_bwe_b(ram1_bwe_b),
.clock_feedback_out(clock_feedback_out),
.clock_feedback_in(clock_feedback_in),
.flash_data(flash_data),
.flash_address(flash_address),
.flash_ce_b(flash_ce_b),
.flash_oe_b(flash_oe_b),
.flash_we_b(flash_we_b),
.flash_reset_b(flash_reset_b),
.flash_sts(flash_sts),
.flash_byte_b(flash_byte_b),
.rs232_txd(rs232_txd),
.rs232_rxd(rs232_rxd),
.rs232_rts(rs232_rts),
.rs232_cts(rs232_cts),
.mouse_clock(mouse_clock),
.mouse_data(mouse_data),
.keyboard_clock(keyboard_clock),
.keyboard_data(keyboard_data),
.clock_27mhz(clock_27mhz),
.clock1(clock1),
.clock2(clock2),
.disp_blank(disp_blank),
.disp_data_out(disp_data_out),
.disp_clock(disp_clock),
.disp_rs(disp_rs),
.disp_ce_b(disp_ce_b),
.disp_reset_b(disp_reset_b),
.disp_data_in(disp_data_in),
.button0(button0),
.button1(button1),
.button2(button2),
.button3(button3),
.button_enter(button_enter),
.button_right(button_right),
.button_left(button_left),
.button_down(button_down),
.button_up(button_up),
.switch(switch),
.led(led),
.user1(user1),
.user2(user2),
.user3(user3),
.user4(user4),
.daughtercard(daughtercard),
.systemace_data(systemace_data),
.systemace_address(systemace_address),
.systemace_ce_b(systemace_ce_b),
```

```
	.systemace_we_b(systemace_we_b),
	.systemace_oe_b(systemace_oe_b),
	.systemace_irq(systemace_irq),
	.systemace_mpbrdy(systemace_mpbrdy),
	.analyzer1_data(analyzer1_data),
	.analyzer1_clock(analyzer1_clock),
	.analyzer2_data(analyzer2_data),
	.analyzer2_clock(analyzer2_clock),
	.analyzer3_data(analyzer3_data),
	.analyzer3_clock(analyzer3_clock),
	.analyzer4_data(analyzer4_data),
	.analyzer4_clock(analyzer4_clock)
);

always #18.5 clock_27mhz = ~clock_27mhz;
//always #40.7 ac97_bit_clock = ~ac97_bit_clock;

/*reg [7:0] count;
always @(posedge clock_27mhz) begin
	uut.real_audio_frame_enable <= (count[2:0] == 0);
	count <= count + 1;
end

assign uut.audio_in = count[7] ? 18'h08000 : 18'h38000;//*/

initial begin
	// Initialize Inputs
	ac97_sdata_in = 0;
	ac97_bit_clock = 0;
	tv_in_ycrcb = 0;
	tv_in_data_valid = 0;
	tv_in_line_clock1 = 0;
	tv_in_line_clock2 = 0;
	tv_in_aef = 0;
	tv_in_hff = 0;
	tv_in_aff = 0;
	clock_feedback_in = 0;
	flash_sts = 0;
	rs232_rxd = 0;
	rs232_cts = 0;
	mouse_clock = 0;
	mouse_data = 0;
	keyboard_clock = 0;
	keyboard_data = 0;
	clock_27mhz = 0;
	clock1 = 0;
	clock2 = 0;
	disp_data_in = 0;
	button0 = 0;
	button1 = 0;
	button2 = 0;
	button3 = 0;
	button_enter = 0;
	button_right = 0;
	button_left = 0;
	button_down = 0;
	button_up = 0;
	switch = 0;
	systemace_irq = 0;
	systemace_mpbrdy = 0;

	//uut.voice_mod.freq_divider.current_bit = 0;
```

```
            //uut.clk_counter = 0;
            //count = 0;

            // Wait 100 ns for global reset to finish
            #100;

            #100;

            // Add stimulus here
            button0 = 1;
            //button1 = 1;
            #100;
            switch = 8'b10001111;
            //uut.ifft_test.frame_count = 10'd1023;
      end

endmodule
```

```
`timescale 1ns / 1ps
//=================
// PS/2 Controller
//=================

// error is high when parity mismatches, or before data is sent
// error is also high when the data is not synchronized properly
// to synchronize, hold data high for > 11 clock cycles

// key_state is high to signal a key being held
// ready goes high when key, key_state can be sampled
module ps2_controller(clock, ps2_clock, data, key, key_state, error, ready, octave,
clear);

   input clock, data, ps2_clock;
   output key_state, error, ready, clear;
   output [4:0] key;
   output [1:0] octave;

   reg [7:0] keycode = 0;
   reg [2:0] bit_count = 0;

   //------------
   // FSM states
   //------------
   parameter START = 0;
   parameter DATA = 1;
   parameter PARITY = 2;
   parameter STOP = 3;

   reg [1:0] state = 0;
   reg error = 0, key_state = 0;
   reg [4:0] key = 0;

   reg frame_ready;

   // data is sampled on the falling edge
   always @(negedge ps2_clock) begin
      case (state)
         START: begin
            // wait for the start bit (always 0)
            if (!data) begin
               state <= DATA; frame_ready <= 0;
               keycode <= 0; bit_count <= 0;
            end
         end

         DATA: begin
            error <= 0;    // clear error flag

            // data is passed in lsb first
            bit_count <= bit_count + 1;
            keycode <= { data, keycode[7:1] };

            // check that we have reached the end of data
            if (bit_count == 7) state <= PARITY;
         end

         // ignore parity bit
         PARITY: state <= STOP;

         STOP: begin
            // check that this bit is 1
```

```
         if (data) begin
            state <= START; frame_ready <= 1;
         end else error <= 1;
      end
   endcase
end

reg last_ready = 0, ready = 0;
reg break = 0;    // is it in break mode?
reg [1:0] octave = 0;
reg clear = 0;    // the "all clear" flag

// output selected key
always @(posedge clock) begin
   last_ready <= frame_ready;
   clear <= 0;
   if (frame_ready && !last_ready) begin
      // defaults
      break <= 0; key_state <= !break;
      ready <= 1;

      case (keycode)
         8'h1A: key <= 0;     // Z
         8'h1B: key <= 1;     // S
         8'h22: key <= 2;     // X
         8'h23: key <= 3;     // D
         8'h21: key <= 4;     // C
         8'h2A: key <= 5;     // V
         8'h34: key <= 6;     // G
         8'h32: key <= 7;     // B
         8'h33: key <= 8;     // H
         8'h31: key <= 9;     // N
         8'h3B: key <= 10;    // J
         8'h3A: key <= 11;    // M
         8'h41: key <= 12;    // ,
         8'h15: key <= 12;    // Q (same note as ,)
         8'h1E: key <= 13;    // 2
         8'h1D: key <= 14;    // W
         8'h26: key <= 15;    // 3
         8'h24: key <= 16;    // E
         8'h2D: key <= 17;    // R
         8'h2E: key <= 18;    // 5
         8'h2C: key <= 19;    // T
         8'h36: key <= 20;    // 6
         8'h35: key <= 21;    // Y
         8'h3D: key <= 22;    // 7
         8'h3C: key <= 23;    // U
         8'h43: key <= 24;    // I

         8'h05: octave <= 0;  // F1
         8'h06: octave <= 1;  // F2
         8'h04: octave <= 2;  // F3

         8'h29: clear <= 1;   // space bar

         8'hF0: begin         // break sequence
            break <= 1;
            ready <= 0;
         end

         default: ready <= 0;
      endcase
   end else ready <= 0;
```

```
    end

endmodule
```

```
`timescale 1ns / 1ps
module ps2_controller_tb_v;

	// Inputs
	reg clock;
	reg ps2_clock;
	reg data;

	// Outputs
	wire [5:0] key;
	wire key_state;
	wire error, ready;

	// Instantiate the Unit Under Test (UUT)
	ps2_controller uut (
		.clock(clock),
		.ps2_clock(ps2_clock),
		.data(data),
		.key(key),
		.key_state(key_state),
		.error(error),
	.ready(ready)
	);

   // internals of uut
   wire [7:0] keycode;
   wire [2:0] bit_count;
   wire [1:0] state;
   wire bit_parity, last_ready, break, frame_ready;

   assign keycode = uut.keycode;
   assign bit_count = uut.bit_count;
   assign state = uut.state;
   assign bit_parity = uut.bit_parity;
   assign last_ready = uut.last_ready;
   assign break = uut.break;
   assign frame_ready = uut.frame_ready;

   // clocks have different speeds
   initial begin
      clock <= 0; ps2_clock <= 0;
   end
   always #2 clock <= ~clock;
   always #3 ps2_clock <= ~ps2_clock;

   reg begin_tests = 0;

      initial begin
		// Initialize Inputs
		data = 1;

		// Wait 100 ns for global reset to finish
		#100;

      // begin tests
      begin_tests = 1;
      end

   wire [10:0] key_v, key_h, key_nothing, key_invalid;
   wire [10:0] key_break;
   assign key_v = 11'b1_1_0010_1010_0;    // should produce key = 5
   assign key_h = 11'b1_0_0011_0011_0;    // should produce key = 8
   assign key_nothing = 11'b1_1_1111_1110_0;
```

```
   assign key_invalid = 11'b1_1_0000_0000_0;
   assign key_break = 11'b1_0_1111_0000_0;

   reg [2:0] in_count = 0;
   reg [3:0] in_bit_count = 0;

   always @(posedge ps2_clock) begin
      if (begin_tests) begin
         // test each input in turn
         case (in_count)
            // high ready, key = 5, key_state = 1
            4'h0: data <= key_v[in_bit_count];

            // low ready, high error
            4'h1: data <= key_invalid[in_bit_count];

            // high ready, key = 8, key_state = 1
            4'h2: data <= key_h[in_bit_count];

            // low ready
            4'h3: data <= key_nothing[in_bit_count];

            // low ready, break = 1
            4'h4: data <= key_break[in_bit_count];

            // high ready, key = 5, key_state = 0, break = 0
            4'h5: data <= key_v[in_bit_count];

            // low ready, break = 1
            4'h6: data <= key_break[in_bit_count];

            // low ready, break = 0
            4'h7: data <= key_nothing[in_bit_count];
         endcase

         if (in_bit_count == 10) begin
            in_count <= in_count + 1;
            in_bit_count <= 0;
         end else in_bit_count <= in_bit_count + 1;
      end
   end

endmodule
```

```
`timescale 1ns / 1ps
//================
// PS/2 Decoder
//================

// converts the key states from keyboard
// into key selection format (key, key_state)
module ps2_decoder(clock, key, ready, key_state, ps2_key_state, ps2_key, ps2_ready,
octave);
   input ps2_ready, clock, ps2_key_state;
   input [1:0] octave;
   input [4:0] ps2_key;
   output ready, key_state;
   output [5:0] key;

   // key output
   // translate the key and octave settings
   // into a single key state (range 0-47)
   reg [5:0] key = 0;
   reg key_state = 0, ready = 0;

   always @(posedge clock) begin
      // one cycle latency before latching key state
      ready <= ps2_ready;

      if (ps2_ready) begin
         case (octave)
            2'h0: key <= {0, ps2_key};
            2'h1: key <= {0, ps2_key} + 12;
            default: key <= {0, ps2_key} + 24;
         endcase

         // remember what the state was
         key_state <= ps2_key_state;
      end
   end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:08:57 05/13/2007
// Design Name:
// Module Name:    spectrum_analyzer
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module spectrum_analyzer(clock, reset, start, done, fft_en, fft_index, fft_re, fft_im,
input_freq);

      input clock;
      input reset;
      input start;
      output done;
      output fft_en;
      output [9:0] fft_index;
      input [22:0] fft_re;
      input [22:0] fft_im;
      output [9:0] input_freq;

      reg done;
      reg [9:0] fft_index;
      reg [9:0] input_freq;

      parameter STATE_N    = 3'd0;
      parameter STATE_2N   = 3'd1;
      parameter STATE_2N_1 = 3'd2;
      parameter STATE_3N   = 3'd3;
      parameter STATE_3N_1 = 3'd4;
      parameter STATE_3N_2 = 3'd5;
      parameter STATE_FINISHING = 3'd6;

      reg busy;
      reg [9:0] current_index;
      reg [2:0] state;
      reg [2:0] finish_count;

      reg [9:0] max_hps_index;
      reg [35:0] max_hps;
      reg [2:0] max_hps_exp;
      reg [35:0] current_hps;
      reg [2:0] current_hps_exp;
      reg [35:0] temp_hps;

      wire signed [17:0] fft_re2;
      wire signed [17:0] fft_im2;
      reg [35:0] fft_norm_sqr;

      assign fft_re2 = fft_re[22:5] + fft_re[4];
      assign fft_im2 = fft_im[22:5] + fft_im[4];
```

```
	always @(posedge clock)
		fft_norm_sqr <= fft_re2 * fft_re2 + fft_im2 * fft_im2;

	wire [35:0] temp_hps_sum;
	wire [71:0] next_hps_scaled;

	assign temp_hps_sum = temp_hps + fft_norm_sqr;
	assign next_hps_scaled = current_hps * temp_hps_sum;

	assign fft_en = busy;

	always @(posedge clock) begin
		if(reset) begin
			done <= 0;
			fft_index <= 10'b0;
			input_freq <= 10'b1;
			busy <= 0;
			current_index <= 10'b0;
			state <= STATE_N;
		end else begin
			if(!busy) begin
				if(start) begin
					busy <= 1;
					fft_index <= 10'd5;
					current_index <= 10'd5;
					state <= STATE_N;
					current_hps <= 36'b0;
					current_hps_exp <= 3'b0;
					temp_hps <= 36'b0;
					max_hps_index <= 10'd4;
					max_hps <= 36'd0;
					max_hps_exp <= 3'b0;
				end

				done <= 0;
			end else case(state)
				STATE_N: begin
					temp_hps <= temp_hps + fft_norm_sqr;
					fft_index <= (fft_index << 1);
					state <= STATE_2N;
				end

				STATE_2N: begin
					if( |next_hps_scaled[71:36]) begin
						current_hps <= (next_hps_scaled >> 36) +
next_hps_scaled[35];
						current_hps_exp <= current_hps_exp + 1;
					end else
						current_hps <= next_hps_scaled[35:0];

					fft_index <= fft_index + 1;
					state <= STATE_2N_1;
				end

				STATE_2N_1: begin
					if(current_hps_exp > max_hps_exp || current_hps_exp ==
max_hps_exp && current_hps > max_hps) begin
						max_hps_index <= current_index - 1;
						max_hps <= current_hps;
						max_hps_exp <= current_hps_exp;
					end
```

```
                        current_hps <= fft_norm_sqr;
                        current_hps_exp <= 3'b0;
                        fft_index <= fft_index - 1 + (fft_index >> 1);
                        state <= STATE_3N;
                  end

                  STATE_3N: begin
                        temp_hps <= fft_norm_sqr;
                        fft_index <= fft_index + 1;
                        state <= STATE_3N_1;
                  end

                  STATE_3N_1: begin
                        if( |next_hps_scaled[71:36]) begin
                              current_hps <= (next_hps_scaled >> 36) +
next_hps_scaled[35];
                              current_hps_exp <= current_hps_exp + 1;
                        end else
                              current_hps <= next_hps_scaled[35:0];

                        fft_index <= fft_index + 1;
                        state <= STATE_3N_2;
                  end

                  STATE_3N_2: begin
                        temp_hps <= fft_norm_sqr;

                        if(current_index == 10'd170) begin
                              fft_index <= 10'b0;
                              state <= STATE_FINISHING;
                              finish_count <= 3'd0;
                        end else begin
                              fft_index <= current_index + 1;
                              current_index <= current_index + 1;
                              state <= STATE_N;
                        end
                  end

                  STATE_FINISHING: begin
                        case(finish_count)
                              3'd0:
                                    temp_hps <= temp_hps + fft_norm_sqr;

                              3'd1: begin
                                    if( |next_hps_scaled[71:36]) begin
                                          current_hps <= (next_hps_scaled >> 36) +
next_hps_scaled[35];
                                          current_hps_exp <= current_hps_exp + 1;
                                    end else
                                          current_hps <= next_hps_scaled[35:0];
                              end

                              3'd2: begin
                                    if(current_hps_exp > max_hps_exp ||
current_hps_exp == max_hps_exp && current_hps > max_hps) begin
                                          max_hps_index <= current_index;
                                          max_hps <= current_hps;
                                          max_hps_exp <= current_hps_exp;
                                    end
                              end

                              3'd3: begin
                                    input_freq <= max_hps_index;
```

```
                                    done <= 1;
                                    current_index <= 10'b0;
                              end
                        endcase

                        if(finish_count == 3'd4) begin
                              done <= 0;
                              busy <= 0;
                        end

                        finish_count <= finish_count + 1;
                  end
            endcase
      end
end

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   23:53:50 05/13/2007
// Design Name:   spectrum_analyzer
// Module Name:   U:/Desktop/finalproject3/spectrum_analyzer_tb.v
// Project Name:  finalproject3
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: spectrum_analyzer
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module spectrum_analyzer_tb_v;

	// Inputs
	reg clock;
	reg reset;
	reg start;
	reg [22:0] fft_re;
	reg [22:0] fft_im;

	// Outputs
	wire done;
	wire fft_en;
	wire [9:0] fft_index;
	wire [9:0] input_freq;

	// Instantiate the Unit Under Test (UUT)
	spectrum_analyzer uut (
		.clock(clock),
		.reset(reset),
		.start(start),
		.done(done),
		.fft_en(fft_en),
		.fft_index(fft_index),
		.fft_re(fft_re),
		.fft_im(fft_im),
		.input_freq(input_freq)
	);

	always #1 clock = ~clock;

	always @(posedge clock) begin
		case(fft_index)
			10, 11, 12, 20, 21, 22, 30, 31, 32: begin
				fft_re <= 23'h020000;
				fft_im <= 23'h7E0000;
			end

			default: begin
				fft_re <= 23'h00200;
```

```
                        fft_im <= 23'h00440;
                end
        endcase
    end

    initial begin
        // Initialize Inputs
        clock = 0;
        reset = 0;
        start = 0;
        fft_re = 0;
        fft_im = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        reset = 1;
        #10;
        reset = 0;
        #10;
        start = 1;
        #10;
        start = 0;
    end

endmodule
```

```
`timescale 1ns / 1ps
//=====================
// Waveform rgb display
//=====================

// Outputs color for given area within display frame.
// - Wave amplitude is shown (solid)
// - Wave is sampled at posedge bit_clock
// - Display can be triggered or held

// To combine multiple viewports, users should use a
// mux to select vga output (switch on coordinates).

// Parameters:
//    FRAME_X, FRAME_Y     -  screen resolution
//    X_MIN, X_MAX         -  x range of viewport
//    Y_MIN, Y_MAX         -  y range of viewport
//    Y_ZERO               -  zero-line
//    RED, GREEN, BLUE     -  wave color
module trigger_display(pixel_clock, x, y, r,g,b, bit_clock, wave_enable, wave, trigger,
hold,
   disp_buff, wave_buff, other_buff, vga_min, vga_max);     // testing (logic analyzer)

   input pixel_clock, bit_clock, trigger, hold, wave_enable;
   input [10:0] x, y;
   input [17:0] wave;
   output [7:0] r,g,b;
   output [1:0] disp_buff, wave_buff, other_buff;
   output [10:0] vga_min, vga_max;

   //---------------------
   // Parameters
   //---------------------

   // VGA frame parameters
   parameter FRAME_X = 1024;
   parameter FRAME_Y = 768;
   parameter TOTAL_X = 1343;     // maximum x value
   parameter TOTAL_Y = 805;      // maximum y value

   // display viewport parameters
   // we only update this section of the VGA frame
   parameter X_MIN = 0;
   parameter X_MAX = 1024;
   parameter X_SIZE = X_MAX - X_MIN - 1;
   parameter Y_MIN = 0;
   parameter Y_MAX = 768;
   parameter Y_ZERO = 384;      // zero-line
   parameter Y_ONE = Y_ZERO + 1;

   // output colors
   parameter RED = 8'h0F;
   parameter GREEN = 8'h0F;
   parameter BLUE = 8'hFF;

   //---------------------
   // Wave/Display Buffers
   //---------------------

   // use 10 bits + sign to store amplitude at x-coordinate
   reg [10:0] buff0[X_SIZE:0], buff1[X_SIZE:0], buff2[X_SIZE:0];
   reg [10:0] buff0m[X_SIZE:0], buff1m[X_SIZE:0], buff2m[X_SIZE:0];
   reg [10:0] vga_addr = 1, wave_addr = 0;    // frame offsets
```

```
// buffers currently in use
// use xnor to find the other buffer
reg [1:0] disp_buff, wave_buff, other_buff;
initial begin
   disp_buff <= 2'b00;
   // synthesis attribute init of disp_buff is "0";
   wave_buff <= 2'b01;
   // synthesis attribute init of wave_buff is "1";
   other_buff <= 2'b10;
   // synthesis attribute init of other_buff is "2";
end

// min/max values for y, given current x position
wire [10:0] vga_min, vga_max;
reg [10:0] vga_min0, vga_min1, vga_min2, vga_max0, vga_max1, vga_max2;
assign vga_min = disp_buff[1] ? vga_min2 :
                 (disp_buff[0] ? vga_min1 :
                                 vga_min0);
assign vga_max = disp_buff[1] ? vga_max2 :
                 (disp_buff[0] ? vga_max1 :
                                 vga_max0);

// display pixel if it is within wave's amplitude
wire hit;
assign hit = (y >= vga_min) && (y <= vga_max);

// colors
assign r = hit ? RED : 8'h0;
assign g = hit ? GREEN : 8'h0;
assign b = hit ? BLUE : 8'h0;

//---------------------
// VGA buffer management
//---------------------

always @(posedge pixel_clock) begin
   // reset address at end of frame
   if (x == FRAME_X && y == FRAME_Y) vga_addr <= 0;

   // increment the vga_addr in time for the second pixel
   else if (x == TOTAL_X && y == TOTAL_Y) vga_addr <= 1;

   // otherwise, increment the buffer addr at each valid pixel
   // it's okay if addr wraps around several times before we
   // get into the valid y-coordinate zone
   else if ((x >= X_MIN) & (x < X_MAX)) vga_addr <= x + 2;

   // update the current min/max values
   vga_min0 <= buff0[vga_addr];
   vga_min1 <= buff1[vga_addr];
   vga_min2 <= buff2[vga_addr];
   vga_max0 <= buff0m[vga_addr];
   vga_max1 <= buff1m[vga_addr];
   vga_max2 <= buff2m[vga_addr];
end

//---------------------
// Wave buffer management
//---------------------

reg disp_ready = 1;  // high: current display has been read from
reg last_zero = 0;   // high: last wave was 0
```

```
   reg triggered = 0;   // high: wave buffer has been triggered

   // is the wave (significant part only) zero?
   wire zero_wave;
   assign zero_wave = wave[16:10] == 0;

   wire [10:0] neg_wave;
   assign neg_wave = { 4'h0, ~wave[16:10] };

   // min/max characteristics
   wire [10:0] wave_min, wave_max;
   assign wave_min = wave[17] ? Y_ZERO : Y_ZERO - { 4'h0, wave[16:10] };
   assign wave_max = wave[17] ? Y_ONE + neg_wave : Y_ZERO;

   always @(posedge bit_clock) begin
      // change display buffers
      if (x >= FRAME_X && y >= FRAME_Y) begin
         // if wave_buff is changing, set disp_buff to the current wave_buff
         // otherwise set it to the completed buffer
         // this prevents wave_buff and disp_buff from being equal
         disp_buff <= hold ? disp_buff :
                      (disp_ready ? ((wave_addr == X_SIZE && wave_enable) ? wave_buff :
other_buff) :
                       disp_buff);

         // unset the flag so we don't cycle through buffers
         disp_ready <= 0;
      end else if (x < FRAME_X && y < FRAME_Y)
         disp_ready <= 1;     // set flag when vga is in valid display space

      // add the new frame to the appropriate buffer
      // we store the minimum and maximum y values for this x-coordinate
      case (wave_buff)
            2'b00: begin
               buff0[wave_addr]  <= wave_min;
               buff0m[wave_addr] <= wave_max;
            end
            2'b01: begin
               buff1[wave_addr]  <= wave_min;
               buff1m[wave_addr] <= wave_max;
            end
            default: begin
               buff2[wave_addr]  <= wave_min;
               buff2m[wave_addr] <= wave_max;
            end
      endcase

      if (wave_enable) begin
         // increment address
         if (wave_addr == X_SIZE) begin
            // wave gets the next available temp buffer
            wave_buff <= other_buff;
            wave_addr <= 0;

            // reset trigger information
            last_zero <= 0; triggered <= 0;
         end else if (~trigger | triggered)
            // already triggered or not concerned with triggering
            // we can increment the address
            wave_addr <= wave_addr + 1;

         // we want to trigger, haven't triggered
         // if zero, we still haven't triggered (but we should mark zero)
```

```
         else if (zero_wave) last_zero <= 1;

         // previously at zero, now at positive => triggered
         else if (last_zero & ~wave[17]) begin
            triggered <= 1;
            wave_addr <= wave_addr + 1;

         // otherwise, unset the previously zero flag
         end else last_zero <= 0;
      end

      // bad, temporary fix
      if ((disp_buff == 3) || (wave_buff == 3)) begin
         disp_buff <= 0; wave_buff <= 1; other_buff <= 2;
      end else other_buff <= disp_buff ~^ wave_buff;

   end

endmodule


////////////////////////////////////////////////////////////////////////////////
//
// generate PCM data for 750hz sine wave (assuming f(ready) = 48khz)
//
////////////////////////////////////////////////////////////////////////////////

module tone750hz (clock, ready, pcm_data);
   input clock;
   input ready;
   output [19:0] pcm_data;

   reg [8:0] index;
   reg [19:0] pcm_data;

   initial begin
      // synthesis attribute init of old_ready is "0";
      index <= 8'h00;
      // synthesis attribute init of index is "00";
      pcm_data <= 20'h00000;
      // synthesis attribute init of pcm_data is "00000";
   end

   always @(posedge clock) begin
      if (ready) index <= index+1;
   end

   // one cycle of a sinewave in 64 20-bit samples
   always @(index) begin
      case (index[5:0])
        6'h00: pcm_data <= 20'h00000;
        6'h01: pcm_data <= 20'h0C8BD;
        6'h02: pcm_data <= 20'h18F8B;
        6'h03: pcm_data <= 20'h25280;
        6'h04: pcm_data <= 20'h30FBC;
        6'h05: pcm_data <= 20'h3C56B;
        6'h06: pcm_data <= 20'h471CE;
        6'h07: pcm_data <= 20'h5133C;
        6'h08: pcm_data <= 20'h5A827;
        6'h09: pcm_data <= 20'h62F20;
        6'h0A: pcm_data <= 20'h6A6D9;
        6'h0B: pcm_data <= 20'h70E2C;
        6'h0C: pcm_data <= 20'h7641A;
```

```
        6'h0D: pcm_data <= 20'h7A7D0;
        6'h0E: pcm_data <= 20'h7D8A5;
        6'h0F: pcm_data <= 20'h7F623;
        6'h10: pcm_data <= 20'h7FFFF;
        6'h11: pcm_data <= 20'h7F623;
        6'h12: pcm_data <= 20'h7D8A5;
        6'h13: pcm_data <= 20'h7A7D0;
        6'h14: pcm_data <= 20'h7641A;
        6'h15: pcm_data <= 20'h70E2C;
        6'h16: pcm_data <= 20'h6A6D9;
        6'h17: pcm_data <= 20'h62F20;
        6'h18: pcm_data <= 20'h5A827;
        6'h19: pcm_data <= 20'h5133C;
        6'h1A: pcm_data <= 20'h471CE;
        6'h1B: pcm_data <= 20'h3C56B;
        6'h1C: pcm_data <= 20'h30FBC;
        6'h1D: pcm_data <= 20'h25280;
        6'h1E: pcm_data <= 20'h18F8B;
        6'h1F: pcm_data <= 20'h0C8BD;
        6'h20: pcm_data <= 20'h00000;
        6'h21: pcm_data <= 20'hF3743;
        6'h22: pcm_data <= 20'hE7075;
        6'h23: pcm_data <= 20'hDAD80;
        6'h24: pcm_data <= 20'hCF044;
        6'h25: pcm_data <= 20'hC3A95;
        6'h26: pcm_data <= 20'hB8E32;
        6'h27: pcm_data <= 20'hAECC4;
        6'h28: pcm_data <= 20'hA57D9;
        6'h29: pcm_data <= 20'h9D0E0;
        6'h2A: pcm_data <= 20'h95927;
        6'h2B: pcm_data <= 20'h8F1D4;
        6'h2C: pcm_data <= 20'h89BE6;
        6'h2D: pcm_data <= 20'h85830;
        6'h2E: pcm_data <= 20'h8275B;
        6'h2F: pcm_data <= 20'h809DD;
        6'h30: pcm_data <= 20'h80000;
        6'h31: pcm_data <= 20'h809DD;
        6'h32: pcm_data <= 20'h8275B;
        6'h33: pcm_data <= 20'h85830;
        6'h34: pcm_data <= 20'h89BE6;
        6'h35: pcm_data <= 20'h8F1D4;
        6'h36: pcm_data <= 20'h95927;
        6'h37: pcm_data <= 20'h9D0E0;
        6'h38: pcm_data <= 20'hA57D9;
        6'h39: pcm_data <= 20'hAECC4;
        6'h3A: pcm_data <= 20'hB8E32;
        6'h3B: pcm_data <= 20'hC3A95;
        6'h3C: pcm_data <= 20'hCF044;
        6'h3D: pcm_data <= 20'hDAD80;
        6'h3E: pcm_data <= 20'hE7075;
        6'h3F: pcm_data <= 20'hF3743;
      endcase // case(index[5:0])
   end // always @ (index)
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:57:16 03/12/2007
// Design Name:
// Module Name:    vga_controller
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module vga_controller(clock, reset, blank, horz_sync, vert_sync, pixel_count,
line_count);
    input clock;
    input reset;
      output blank;
    output horz_sync;
    output vert_sync;
    output [10:0] pixel_count;
    output [9:0] line_count;

      // Display parameters
      parameter WIDTH            = 11'd640;
      parameter HORZ_FRONT_PORCH = 11'd16;
      parameter HORZ_SYNC        = 11'd96;
      parameter HORZ_BACK_PORCH  = 11'd48;

      parameter HEIGHT           = 10'd480;
      parameter VERT_FRONT_PORCH = 10'd11;
      parameter VERT_SYNC        = 10'd2;
      parameter VERT_BACK_PORCH  = 10'd32;

      reg [10:0] pixel_counter;
      reg [9:0] line_counter;

      // Pixel and line counter logic
      always @(posedge clock) begin
            if(reset) begin
                  pixel_counter <= 11'd0;
                  line_counter <= 10'd0;
            end else begin
                  if(pixel_counter == WIDTH + HORZ_FRONT_PORCH + HORZ_SYNC +
HORZ_BACK_PORCH - 1) begin
                        pixel_counter <= 11'd0;

                        if(line_counter == HEIGHT + VERT_FRONT_PORCH + VERT_SYNC +
VERT_BACK_PORCH - 1)
                              line_counter <= 10'd0;
                        else
                              line_counter <= line_counter + 1;
                  end else
                        pixel_counter <= pixel_counter + 1;

            end
```

```
    end

    // Output logic
    assign pixel_count = pixel_counter;
    assign line_count = line_counter;

    assign blank = (pixel_counter < WIDTH) && (line_counter < HEIGHT);
    assign horz_sync = ((pixel_counter < WIDTH + HORZ_FRONT_PORCH) || (pixel_counter >=
WIDTH + HORZ_FRONT_PORCH + HORZ_SYNC));
    assign vert_sync = ((line_counter < HEIGHT + VERT_FRONT_PORCH) || (line_counter >=
HEIGHT + VERT_FRONT_PORCH + VERT_SYNC));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:38:17 05/08/2007
// Design Name:
// Module Name:    voice_modulator_controller
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module voice_modulator_controller(clock, reset, start, keys_down, ifft_buff_reset,
ifft_buff_reset_done, vm_start, vm_done, vm_all_done, vm_out_freq);

	input clock;
	input reset;
	input start;
	input [47:0] keys_down;
	output ifft_buff_reset;
	input ifft_buff_reset_done;
	output vm_start;
	input vm_done;
	output vm_all_done;
	output [12:0] vm_out_freq;

	reg ifft_buff_reset;
	reg vm_start;
	reg vm_all_done;

	parameter IDLE = 2'd0;
	parameter BUFF_RESET = 2'd1;
	parameter MODULATING = 2'd2;

	reg [1:0] state;

	reg [5:0] current_key;

	key_frequency_table key_freq_table(.key(current_key), .freq(vm_out_freq));

	always @(posedge clock) begin
		if(reset) begin
			state <= IDLE;
			ifft_buff_reset <= 0;
			vm_start <= 0;
			vm_all_done <= 0;
		end else begin
			case(state)
				IDLE: begin
					vm_all_done <= 0;

					if(start) begin
						state <= BUFF_RESET;
						ifft_buff_reset <= 1;
					end
```

```
                        end

                        BUFF_RESET: begin
                              ifft_buff_reset <= 0;

                              if(ifft_buff_reset_done) begin
                                    vm_start <= 1;
                                    state <= MODULATING;
                                    current_key <= 6'b0;
                              end
                        end

                        MODULATING: begin
                              if(vm_done || ~keys_down[current_key]) begin
                                    if(current_key == 6'd47) begin
                                          state <= IDLE;
                                          vm_start <= 0;
                                          vm_all_done <= 1;
                                    end else begin
                                          current_key <= current_key + 1;
                                          vm_start <= keys_down[current_key + 1];
                                    end
                              end else
                                    vm_start <= 0;
                        end
                  endcase
            end
      end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:37:52 05/03/2007
// Design Name:
// Module Name:    voice_modulator
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module voice_modulator(clock, reset, start, done, fft_in_index, fft_in_re, fft_in_im,
input_freq, output_freq, fft_out_add, fft_out_index, fft_out_re, fft_out_im);

      input clock;
      input reset;
      input start;
      output done;
      output [9:0] fft_in_index;
      input [22:0] fft_in_re;
      input [22:0] fft_in_im;
      input [9:0] input_freq;
      input [12:0] output_freq;
      output fft_out_add;
      output [9:0] fft_out_index;
      output [22:0] fft_out_re;
      output [22:0] fft_out_im;

      reg done;
      reg [9:0] fft_in_index;
      reg fft_out_add;
      reg [9:0] fft_out_index;
      reg [22:0] fft_out_re;
      reg [22:0] fft_out_im;

      // State values
      parameter IDLE = 2'd0;
      parameter WAITING_FOR_DIVIDER = 2'd1;
      parameter DIVIDING = 2'd2;
      parameter WORKING = 2'd3;

      reg [1:0] state;
      //reg substate;

      wire divider_rfd;
      wire divider_done;

      // Fixed-point, 11 integer bits, 8 fraction bits
      wire [20:0] freq_ratio;

      divider freq_divider(
            .clock(clock),
            .reset(reset),
            .dividend(output_freq),
```

```
	.divisor(input_freq),
	.rfd(divider_rfd),
	.done(divider_done),
	.quotient(freq_ratio));

defparam freq_divider.DIVIDEND_SIZE = 13;

reg [4:0] divide_counter;

// Delay by 1 cycle (BRAM latency)
reg [9:0] fft_in_index_d1;

always @(posedge clock)
	fft_in_index_d1 <= fft_in_index;

wire [14:0] shifted_index;
wire [10:0] shifted_index_frac;

assign {shifted_index, shifted_index_frac} = fft_in_index_d1 * freq_ratio[15:0];

//wire [8:0] weight;

//assign weight = substate ? (9'h100 - shifted_index_frac) : shifted_index_frac;

//wire [30:0] fft_re_weighted;
//wire [30:0] fft_im_weighted;

//assign fft_re_weighted = fft_in_re * weight;
//assign fft_im_weighted = fft_in_im * weight;

always @(posedge clock) begin
	if(reset) begin
		state <= IDLE;
		//substate <= 0;
		done <= 0;
		fft_in_index <= 10'b0;
		fft_out_add <= 0;
		fft_out_index <= 10'b0;
		fft_out_re <= 23'b0;
		fft_out_im <= 23'b0;
		divide_counter <= 5'b0;
	end else begin
		case(state)
			IDLE: begin
				if(start) begin
					state <= WAITING_FOR_DIVIDER;
					divide_counter <= 5'b0;
				end

				done <= 0;
			end

			WAITING_FOR_DIVIDER: begin
				if(divider_rfd)
					state <= DIVIDING;
			end

			DIVIDING: begin
				if(divide_counter == 5'd23) begin
					state <= WORKING;
					//substate <= 0;
					fft_in_index <= 10'b0;
					fft_out_add <= 1;
```

```
                              fft_out_re <= 23'b0;
                              fft_out_im <= 23'b0;
                        end else
                              divide_counter <= divide_counter + 1;
                  end

                  WORKING: begin
                        //substate <= ~substate;

                        //if(substate)
                              fft_out_index <= shifted_index[9:0];
                        //else
                        //    fft_out_index <= fft_out_index + 1;

                        fft_out_re <= fft_in_re;//(fft_re_weighted >> 8);
                        fft_out_im <= fft_in_im;//(fft_im_weighted >> 8);

                        //if(substate) begin
                              if(fft_in_index == 10'd512 || shifted_index >=
14'd512) begin
                                    state <= IDLE;
                                    //substate <= 0;
                                    done <= 1;
                                    fft_in_index <= 10'b0;
                                    fft_out_add <= 0;
                              end else
                                    fft_in_index <= fft_in_index + 1;
                        //end
                  end
            endcase
      end
end

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   16:41:48 05/03/2007
// Design Name:   voice_modulator
// Module Name:   U:/Desktop/finalproject2/voice_modulator_tb.v
// Project Name:  finalproject2
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: voice_modulator
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module voice_modulator_tb_v;

	// Inputs
	reg clock;
	reg reset;
	reg start;
	reg [17:0] fft_in_re;
	reg [17:0] fft_in_im;
	reg [9:0] input_freq;
	reg [9:0] output_freq;

	// Outputs
	wire done;
	wire [9:0] fft_in_index;
	wire fft_out_add;
	wire [9:0] fft_out_index;
	wire [17:0] fft_out_re;
	wire [17:0] fft_out_im;

	// Instantiate the Unit Under Test (UUT)
	voice_modulator uut (
		.clock(clock),
		.reset(reset),
		.start(start),
		.done(done),
		.fft_in_index(fft_in_index),
		.fft_in_re(fft_in_re),
		.fft_in_im(fft_in_im),
		.input_freq(input_freq),
		.output_freq(output_freq),
		.fft_out_add(fft_out_add),
		.fft_out_index(fft_out_index),
		.fft_out_re(fft_out_re),
		.fft_out_im(fft_out_im)
	);

	always #1 clock = ~clock;

	always @(posedge clock) begin
```

```
            fft_in_re <= fft_in_index;
            fft_in_im <= fft_in_index + 1;
      end

      initial begin
            // Initialize Inputs
            clock = 0;
            reset = 0;
            start = 0;
            fft_in_re = 0;
            fft_in_im = 0;
            input_freq = 0;
            output_freq = 0;

            // Wait 100 ns for global reset to finish
            #100;

            uut.freq_divider.current_bit = 0;

            // Add stimulus here
            reset = 1;
            #10;
            reset = 0;

            input_freq = 10;
            output_freq = 15;

            start = 1;
            #2;
            start = 0;
      end

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   12:53:18 05/08/2007
// Design Name:   voice_modulator
// Module Name:   U:/Desktop/finalproject3/voice_modulator_tb2.v
// Project Name:  finalproject3
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: voice_modulator
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module voice_modulator_tb2_v;

	// Inputs
	reg clock;
	reg reset;
	reg start;
	reg [9:0] input_freq;
	reg [9:0] output_freq;

	wire vm_ctrl_start;
	wire voice_mod_start;
	wire voice_mod_done;
	wire voice_mod_all_done;
	wire [9:0] voice_mod_in_index;
	wire [17:0] voice_mod_in_re;
	wire [17:0] voice_mod_in_im;
	wire voice_mod_add;
	wire [9:0] voice_mod_out_index;
	wire [17:0] voice_mod_out_re;
	wire [17:0] voice_mod_out_im;

	wire ifft_buff_reset;
	wire ifft_buff_reset_done;

	wire ifft_start;
	reg [9:0] ifft_fft_index;
	wire [17:0] ifft_fft_re;
	wire [17:0] ifft_fft_im;

	reg [2:0] count;

	always @(posedge clock)
		count <= count + 1;

	// Inverse FFT tester
	ifft_tester ifft_test(
		.clock(clock),
		.reset(reset),
		.switch(8'b00001011),
```

```
	.frame_enable(count == 0),
	.ifft_start(vm_ctrl_start),
	.index(voice_mod_in_index),
	.fft_re(voice_mod_in_re),
	.fft_im(voice_mod_in_im));//*/

voice_modulator_controller voice_mod_ctrl(
	.clock(clock),
	.reset(reset),
	.start(vm_ctrl_start),
	.ifft_buff_reset(ifft_buff_reset),
	.ifft_buff_reset_done(ifft_buff_reset_done),
	.vm_start(voice_mod_start),
	.vm_done(voice_mod_done),
	.vm_all_done(voice_mod_all_done));

voice_modulator voice_mod(
	.clock(clock),
	.reset(reset),
	.start(voice_mod_start),
	.done(voice_mod_done),
	.fft_in_index(voice_mod_in_index),
	.fft_in_re(voice_mod_in_re),
	.fft_in_im(voice_mod_in_im),
	.input_freq(input_freq),
	.output_freq(output_freq),
	.fft_out_add(voice_mod_add),
	.fft_out_index(voice_mod_out_index),
	.fft_out_re(voice_mod_out_re),
	.fft_out_im(voice_mod_out_im));

wire buff_reset_done;

fft_buffer fft_buff(
	.clock(clock),
	.reset(ifft_buff_reset),
	.reset_done(ifft_buff_reset_done),
	.vm_add(voice_mod_add),
	.vm_index(voice_mod_out_index),
	.vm_re(voice_mod_out_re),
	.vm_im(voice_mod_out_im),
	.fft_index(ifft_fft_index),
	.fft_re(ifft_fft_re),
	.fft_im(ifft_fft_im));

always #1 clock = ~clock;

always @(posedge clock)
	ifft_fft_index <= ifft_fft_index + 1;

initial begin
	// Initialize Inputs
	clock = 0;
	reset = 0;
	start = 0;
	input_freq = 0;
	output_freq = 0;
	count = 0;

	// Wait 100 ns for global reset to finish
	#100;

	// Add stimulus here
```

```
            reset = 1;
            input_freq = 21;
            output_freq = 10;
            #100;
            reset = 0;

            ifft_test.frame_count = 1000;
            voice_mod.freq_divider.current_bit = 0;
            ifft_fft_index = 0;
      end

endmodule
```

```
//====================
// Waveform rgb display
//====================

// Outputs color for given area within display frame.
// - Wave amplitude is shown (solid)
// - Wave is sampled at posedge bit_clock
// - Display can be triggered or held

// To combine multiple viewports, users should use a
// mux to select vga output (switch on coordinates).

// Parameters:
//    FRAME_X, FRAME_Y     -  screen resolution
//    X_MIN, X_MAX         -  x range of viewport
//    Y_MIN, Y_MAX         -  y range of viewport
//    Y_ZERO               -  zero-line
//    RED, GREEN, BLUE     -  wave color
module wave_display(pixel_clock, x, y, r,g,b, bit_clock, wave_enable, wave, trigger,
hold,
   disp_buff, wave_buff, other_buff, vga_min, vga_max);

   input pixel_clock, bit_clock, trigger, hold, wave_enable;
   input [10:0] x, y;
   input [17:0] wave;
   output [7:0] r,g,b;
   output [1:0] disp_buff, wave_buff, other_buff;
   output [10:0] vga_min, vga_max;

   //---------------------
   // Parameters
   //---------------------

   // VGA frame parameters
   parameter FRAME_X = 1024;
   parameter FRAME_Y = 768;
   parameter TOTAL_X = 1343;     // maximum x value
   parameter TOTAL_Y = 805;      // maximum y value

   // display viewport parameters
   // we only update this section of the VGA frame
   parameter X_MIN = 0;
   parameter X_MAX = 1024;
   parameter X_SIZE = X_MAX - X_MIN - 1;
   parameter Y_MIN = 0;
   parameter Y_MAX = 768;
   parameter Y_ZERO = 384;      // zero-line
   parameter Y_ONE = Y_ZERO + 1;

   // output colors
   parameter RED = 8'h0F;
   parameter GREEN = 8'h0F;
   parameter BLUE = 8'hFF;

   //---------------------
   // Wave/Display Buffers
   //---------------------

   // use 10 bits + sign to store amplitude at x-coordinate
   reg [10:0] buff0[X_SIZE:0], buff1[X_SIZE:0], buff2[X_SIZE:0];
   reg [10:0] buff0m[X_SIZE:0], buff1m[X_SIZE:0], buff2m[X_SIZE:0];
   reg [10:0] vga_addr = 1, wave_addr = 0;    // frame offsets
```

```
// buffers currently in use
// use xnor to find the other buffer
reg [1:0] disp_buff = 2'b00, wave_buff = 2'b01;
wire [1:0] other_buff;
assign other_buff = disp_buff ~^ wave_buff;

// min/max values for y, given current x position
wire [10:0] vga_min, vga_max;
reg [10:0] vga_min0, vga_min1, vga_min2, vga_max0, vga_max1, vga_max2;
assign vga_min = disp_buff[1] ? vga_min2 :
                 (disp_buff[0] ? vga_min1 :
                                 vga_min0);
assign vga_max = disp_buff[1] ? vga_max2 :
                 (disp_buff[0] ? vga_max1 :
                                 vga_max0);

// display pixel if it is within wave's amplitude
wire hit;
assign hit = (y >= vga_min) && (y <= vga_max);

// colors
assign r = hit ? RED : 8'h0;
assign g = hit ? GREEN : 8'h0;
assign b = hit ? BLUE : 8'h0;

//----------------------
// VGA buffer management
//----------------------

always @(posedge pixel_clock) begin
   // reset address at end of frame
   if (x == FRAME_X && y == FRAME_Y) vga_addr <= 0;

   // increment the vga_addr in time for the second pixel
   else if (x == TOTAL_X && y == TOTAL_Y) vga_addr <= 1;

   // otherwise, increment the buffer addr at each valid pixel
   // it's okay if addr wraps around several times before we
   // get into the valid y-coordinate zone
   else if ((x >= X_MIN) & (x < X_MAX)) vga_addr <= x + 2;

   // update the current min/max values
   vga_min0 <= buff0[vga_addr];
   vga_min1 <= buff1[vga_addr];
   vga_min2 <= buff2[vga_addr];
   vga_max0 <= buff0m[vga_addr];
   vga_max1 <= buff1m[vga_addr];
   vga_max2 <= buff2m[vga_addr];
end

//----------------------
// Wave buffer management
//----------------------

reg disp_ready = 1;  // high: current display has been read from
reg last_zero = 0;   // high: last wave was 0
reg triggered = 0;   // high: wave buffer has been triggered

// is the wave (significant part only) zero?
wire zero_wave;
assign zero_wave = wave[16:10] == 0;

wire [10:0] neg_wave;
```

```
   assign neg_wave = { 3'h0, ~wave[16:10] };

   // min/max characteristics
   wire [10:0] wave_min, wave_max;
   assign wave_min = wave[17] ? Y_ZERO : Y_ZERO - wave[16:10];
   assign wave_max = wave[17] ? Y_ONE + neg_wave : Y_ZERO;

   always @(posedge bit_clock) begin
      // change display buffers
      if (x >= FRAME_X && y >= FRAME_Y) begin
         // if wave_buff is changing, set disp_buff to the current wave_buff
         // otherwise set it to the completed buffer
         // this prevents wave_buff and disp_buff from being equal
         disp_buff <= (disp_ready & ~hold) ? (wave_addr == X_SIZE ? wave_buff :
other_buff) :
                                             disp_buff;

         // unset the flag so we don't cycle through buffers
         disp_ready <= 0;
      end else if (x < FRAME_X && y < FRAME_Y)
         disp_ready <= 1;     // set flag when vga is in valid display space

      // add the new frame to the appropriate buffer
      // we store the minimum and maximum y values for this x-coordinate
      if (wave_enable) begin
         case (wave_buff)
            2'b00: begin
               buff0[wave_addr]  <= wave_min;
               buff0m[wave_addr] <= wave_max;
            end
            2'b01: begin
               buff1[wave_addr]  <= wave_min;
               buff1m[wave_addr] <= wave_max;
            end
            default: begin
               buff2[wave_addr]  <= wave_min;
               buff2m[wave_addr] <= wave_max;
            end
         endcase
         // increment address
         if (wave_addr == X_SIZE) begin
            // wave gets the next available temp buffer
            wave_buff <= other_buff;
            wave_addr <= 0;

            // reset trigger information
            last_zero <= 0; triggered <= 0;
         end else if (~trigger | triggered)
            // already triggered or not concerned with triggering
            // we can increment the address
            wave_addr <= wave_addr + 1;

         // we want to trigger, haven't triggered
         // if zero, we still haven't triggered (but we should mark zero)
         else if (zero_wave) last_zero <= 1;

         // previously at zero, now at positive => triggered
         else if (last_zero & ~wave[17]) begin
            triggered <= 1;
            wave_addr <= wave_addr + 1;

         // otherwise, unset the previously zero flag
         end else last_zero <= 0;
```

```
      end
   end

endmodule
```