```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:46:41 05/07/06
// Design Name:
// Module Name:    add3
// Project Name:
// Target Device:
// Tool versions:
// Description: Module used to add 3    to a 4 bit number
//                                      Look up table used for the
binary8tobcd module
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
http://www.engr.udayton.edu/faculty/jloomis/ece314/notes/devices/binary_to_BCD/bin_t
o_BCD.html
//
//////////////////////////////////////////////////////////////////////////////////
module add3(in,out);
input [3:0] in;
output [3:0] out;
reg [3:0] out;


always @ (in)
        case (in)
        4'b0000: out <= 4'b0000;
```



```
        4'b0001: out <= 4'b0001;
        4'b0010: out <= 4'b0010;
        4'b0011: out <= 4'b0011;
        4'b0100: out <= 4'b0100;
        4'b0101: out <= 4'b1000;
        4'b0110: out <= 4'b1001;
        4'b0111: out <= 4'b1010;
        4'b1000: out <= 4'b1011;
        4'b1001: out <= 4'b1100;
        default: out <= 4'b0000;
        endcase
endmodule
```

```
module audio (reset, clock_27mhz, audio_reset_b, ac97_sdata_out, ac97_sdata_in,
sample,
              ac97_synch, ac97_bit_clock);

   input reset, clock_27mhz;
   output audio_reset_b;
   output ac97_sdata_out;
        input [19:0] sample;    //input from tomrom

   input ac97_sdata_in;
   output ac97_synch;
   input ac97_bit_clock;



   wire ready;
   wire [7:0] command_address;
   wire [15:0] command_data;
   wire command_valid;
   // AC'97 spec requires 20 bit values for slots.

        // But the LM4550 only uses the 18 most significant bits.

        reg [19:0] left_out_data, right_out_data;
   wire [19:0] left_in_data, right_in_data;

   //
   // Reset controller.  This requires an external clock such as clock_27mHz
   //

   reg audio_reset_b;
   reg [9:0] reset_count;

   always @(posedge clock_27mhz) begin
      if (reset)
        begin
           audio_reset_b = 1'b0;
           reset_count = 0;
        end
      else if (reset_count == 1023)
        audio_reset_b = 1'b1;
      else
        reset_count = reset_count+1;
   end

   ac97 ac97(ready, command_address, command_data, command_valid,
             left_out_data, 1'b1, right_out_data, 1'b1, left_in_data,
             right_in_data, ac97_sdata_out, ac97_sdata_in, ac97_synch,
             ac97_bit_clock);

   ac97commands cmds(clock_27mhz, ready, command_address, command_data,
                     command_valid);

   always @(sample)
                begin

                        left_out_data = sample;
                        right_out_data = sample;
/*                 // digital loopback.  20 bit sample values declared above.
                        left_out_data = left_in_data;
                        right_out_data = right_in_data;*/
        end
endmodule
```



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

        Finite state machine which loops though an audio frame based on the
bit_clock.

        During this loop it serializes values for digital to analog conversion and

        constructs a parallel value from the serialized analog to digital
conversion.

*/
module ac97 (ready,
             command_address, command_data, command_valid,
             left_data, left_valid,
             right_data, right_valid,
             left_in_data, right_in_data,
             ac97_sdata_out, ac97_sdata_in, ac97_synch, ac97_bit_clock);

   output ready;
   input [7:0] command_address;
   input [15:0] command_data;
   input command_valid;
   input [19:0] left_data, right_data;
   input left_valid, right_valid;
   output [19:0] left_in_data, right_in_data;

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

   reg ready;

   reg ac97_sdata_out;
   reg ac97_synch;

   reg [7:0] bit_count;

   reg [19:0] l_cmd_addr;
   reg [19:0] l_cmd_data;
   reg [19:0] l_left_data, l_right_data;
   reg l_cmd_v, l_left_v, l_right_v;
   reg [19:0] left_in_data, right_in_data;


   /*

         Evil trick to initialize for simulation via initial block

              command and FPGA via specialized comments.

        */
   initial begin
      ready <= 1'b0;
      // synthesis attribute init of ready is "0";
      ac97_sdata_out <= 1'b0;
      // synthesis attribute init of ac97_sdata_out is "0";
      ac97_synch <= 1'b0;
      // synthesis attribute init of ac97_synch is "0";
```

```
   bit_count <= 8'h00;
   // synthesis attribute init of bit_count is "0000";
   l_cmd_v <= 1'b0;
   // synthesis attribute init of l_cmd_v is "0";
   l_left_v <= 1'b0;
   // synthesis attribute init of l_left_v is "0";
   l_right_v <= 1'b0;
   // synthesis attribute init of l_right_v is "0";

   left_in_data <= 20'h00000;
   // synthesis attribute init of left_in_data is "00000";
   right_in_data <= 20'h00000;
   // synthesis attribute init of right_in_data is "00000";
end


     // Construct a frame bit by bit.  Note parallel to serial conversion.
always @(posedge ac97_bit_clock) begin
   // Generate the sync signal
   if (bit_count == 255)
     ac97_synch <= 1'b1;
   if (bit_count == 15)
     ac97_synch <= 1'b0;

   // Generate the ready signal
   if (bit_count == 128)
     ready <= 1'b1;
   if (bit_count == 2)
     ready <= 1'b0;

   // Latch user data at the end of each frame. This ensures that the
   // first frame after reset will be empty.
   if (bit_count == 255)
     begin
        l_cmd_addr <= {command_address, 12'h000};
        l_cmd_data <= {command_data, 4'h0};
        l_cmd_v <= command_valid;
        l_left_data <= left_data;
        l_left_v <= left_valid;
        l_right_data <= right_data;
        l_right_v <= right_valid;
     end

   if ((bit_count >= 0) && (bit_count <= 15))
     // Slot 0: Tags
     case (bit_count[3:0])
       4'h0: ac97_sdata_out <= 1'b1;      // Frame valid
       4'h1: ac97_sdata_out <= l_cmd_v;   // Command address valid
       4'h2: ac97_sdata_out <= l_cmd_v;   // Command data valid
       4'h3: ac97_sdata_out <= l_left_v;  // Left data valid
       4'h4: ac97_sdata_out <= l_right_v; // Right data valid
       default: ac97_sdata_out <= 1'b0;
     endcase

   else if ((bit_count >= 16) && (bit_count <= 35))
     // Slot 1: Command address (8-bits, left justified)
     ac97_sdata_out <= l_cmd_v ? l_cmd_addr[35-bit_count] : 1'b0;

   else if ((bit_count >= 36) && (bit_count <= 55))
     // Slot 2: Command data (16-bits, left justified)
     ac97_sdata_out <= l_cmd_v ? l_cmd_data[55-bit_count] : 1'b0;

   else if ((bit_count >= 56) && (bit_count <= 75))
```



```
     begin
        // Slot 3: Left channel
        ac97_sdata_out <= l_left_v ? l_left_data[19] : 1'b0;
        l_left_data <= { l_left_data[18:0], l_left_data[19] };
     end
   else if ((bit_count >= 76) && (bit_count <= 95))
     // Slot 4: Right channel
        ac97_sdata_out <= l_right_v ? l_right_data[95-bit_count] : 1'b0;
   else
     ac97_sdata_out <= 1'b0;

   bit_count <= bit_count+1;

end // always @ (posedge ac97_bit_clock)


     // Construct a sample bit by bit.  Note serial to parallel conversion.
always @(negedge ac97_bit_clock) begin
   if ((bit_count >= 57) && (bit_count <= 76))
     // Slot 3: Left channel
     left_in_data <= { left_in_data[18:0], ac97_sdata_in };
   else if ((bit_count >= 77) && (bit_count <= 96))
     // Slot 4: Right channel
     right_in_data <= { right_in_data[18:0], ac97_sdata_in };
end

endmodule

//////////////////////////////////////////////////////////////////////////////////


/*

        Finite state machine which continously loops though all of the commands

        for configuring AC97 the audio controller.  Note that volume and record

        source are hardwired to maximum volume and line_in respectively for

        brevity.  These could be configured with switches from the labkit.

*/
module ac97commands (clock, ready, command_address, command_data,
                     command_valid);

   input clock;
   input ready;
   output [7:0] command_address;
   output [15:0] command_data;
   output command_valid;

   reg [23:0] command;
   reg command_valid;

   reg old_ready;
   reg done;
   reg [3:0] state;

   initial begin
      command <= 4'h0;
      // synthesis attribute init of command is "0";
      command_valid <= 1'b0;
      // synthesis attribute init of command_valid is "0";
```

```
      done <= 1'b0;
      // synthesis attribute init of done is "0";
      old_ready <= 1'b0;
      // synthesis attribute init of old_ready is "0";
      state <= 16'h0000;
      // synthesis attribute init of state is "0000";
   end

   assign command_address = command[23:16];
   assign command_data = command[15:0];

   wire [4:0] vol;
   assign vol = 5'd0;   // maximum volume (lowest atinuation)

   always @(posedge clock) begin
      if (ready && (!old_ready))
        state <= state+1;

      case (state)
        4'h0: // Read ID
          begin
             command <= 24'h80_0000;
             command_valid <= 1'b1;
          end
        4'h1: // Read ID
          command <= 24'h80_0000;
        4'h2: // Master volume
          command <= { 8'h02, 3'b000, vol, 3'b000, vol };
        4'h3: // Aux volume
          command <= { 8'h04, 3'b000, vol, 3'b000, vol };
        4'h4: // Mono volume
          command <= 24'h06_8000;
        4'h5: // PCM volume
          command <= 24'h18_0808;
        4'h6: // Record source select
           command <= 24'h1A_0404; // line-in
        4'h7: // Record gain
          command <= 24'h1C_0000;
        4'h8: // Line in gain
          command <= 24'h10_8000;
        //4'h9: // Set jack sense pins
          //command <= 24'h72_3F00;
        4'hA: // Set beep volume
          command <= 24'h0A_0000;
        //4'hF: // Misc control bits
          //command <= 24'h76_8000;
        default:
          command <= 24'h80_0000;
      endcase // case(state)

      old_ready <= ready;

   end // always @ (posedge clock)

endmodule // ac97commands
```



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:47:30 05/07/06
// Design Name:
// Module Name:    binary4tobcd
// Project Name:
// Target Device:
// Tool versions:
// Description:  4 bit binary number converter to binary coded decimal
//                                  found online - see URL under "Additional Comments"
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
http://www.engr.udayton.edu/faculty/jloomis/ece314/notes/devices/binary_to_BCD/bin_t
o_BCD.html
//
//////////////////////////////////////////////////////////////////////////////////
module binary8tobcd(A,ONES,TENS,HUNDREDS);
input [7:0] A;
output [3:0] ONES, TENS;
output [1:0] HUNDREDS;
wire [3:0] c1,c2,c3,c4,c5,c6,c7;
wire [3:0] d1,d2,d3,d4,d5,d6,d7;


assign d1 = {1'b0,A[7:5]};
assign d2 = {c1[2:0],A[4]};
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:             Shirley Fung
//
// Create Date:    16:00:45 04/28/06
// Design Name:
// Module Name:    blocks
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module blocks(pixel_clk, reset_sync,
                                  p_cnt, l_cnt, pattern,
                                  vga_out_rgb);



input pixel_clk;
input reset_sync;
input [9:0] p_cnt, l_cnt;
input [15:0] pattern;
output [23:0] vga_out_rgb;

reg [23:0] vga_out_rgb;
```



```
assign d3 = {c2[2:0],A[3]};
assign d4 = {c3[2:0],A[2]};
assign d5 = {c4[2:0],A[1]};
assign d6 = {1'b0,c1[3],c2[3],c3[3]};
assign d7 = {c6[2:0],c4[3]};
add3 m1(d1,c1);
add3 m2(d2,c2);
add3 m3(d3,c3);
add3 m4(d4,c4);
add3 m5(d5,c5);
add3 m6(d6,c6);
add3 m7(d7,c7);
assign ONES = {c5[2:0],A[0]};
assign TENS = {c7[2:0],c5[3]};
assign HUNDREDS = {c6[3],c7[3]};

endmodule
```

```
parameter top_space = 32-1;
parameter vert_distance = 16;
parameter height = 8;

parameter left_offset = 125;
parameter horiz_distance = 15;
parameter width = 8;
parameter off1 = 24'h006699;
parameter on1 = 24'h33CCFF;
parameter off2 = 24'hFF6600;
parameter on2 = 24'hFFCC33;



always @ (posedge pixel_clk)

        begin
        if (reset_sync) // reset is high
                vga_out_rgb <= 24'd0;

        // specifies which lines to apply drawing
        else if ((l_cnt >= top_space && l_cnt <= top_space+height) ||
                                (l_cnt >= top_space+vert_distance && l_cnt <=
top_space+vert_distance+height) ||
                                (l_cnt >= top_space+vert_distance*2 && l_cnt <=
top_space+vert_distance*2+height) ||
                                (l_cnt >= top_space+vert_distance*3 && l_cnt <=
top_space+vert_distance*3+height) ||
                                (l_cnt >= top_space+vert_distance*4 && l_cnt <=
top_space+vert_distance*4+height) ||
                                (l_cnt >= top_space+vert_distance*5 && l_cnt <=
top_space+vert_distance*5+height) ||
                                (l_cnt >= top_space+vert_distance*6 && l_cnt <=
top_space+vert_distance*6+height) ||
```



```
                                (l_cnt >= top_space+vert_distance*7 && l_cnt <=
top_space+vert_distance*7+height) ||
                                (l_cnt >= top_space+vert_distance*8 && l_cnt <=
top_space+vert_distance*8+height) ||
                                (l_cnt >= top_space+vert_distance*9 && l_cnt <=
top_space+vert_distance*9+height) ||
                                (l_cnt >= top_space+vert_distance*10 && l_cnt <=
top_space+vert_distance*10+height) ||
                                (l_cnt >= top_space+vert_distance*11 && l_cnt <=
top_space+vert_distance*11+height) ||
                                (l_cnt >= top_space+vert_distance*12 && l_cnt <=
top_space+vert_distance*12+height) ||
                                (l_cnt >= top_space+vert_distance*13 && l_cnt <=
top_space+vert_distance*13+height) ||
                                (l_cnt >= top_space+vert_distance*14 && l_cnt <=
top_space+vert_distance*14+height) ||
                                (l_cnt >= top_space+vert_distance*15 && l_cnt <=
top_space+vert_distance*15+height) ||
                                (l_cnt >= top_space+vert_distance*16 && l_cnt <=
top_space+vert_distance*16+height) ||
                                (l_cnt >= top_space+vert_distance*18 && l_cnt <=
top_space+vert_distance*18+height))


                if (p_cnt >= left_offset && p_cnt <= left_offset+width)
                        // try drawing
                        // if on - white, if off - gray
                        vga_out_rgb <= (pattern[15])? on1 : off1;


                else if (p_cnt >= left_offset+horiz_distance && p_cnt <=
left_offset+horiz_distance+width)
                        vga_out_rgb <= (pattern[14])? on1 : off1;


                else if (p_cnt >= left_offset+horiz_distance*2 && p_cnt <=
left_offset+horiz_distance*2+width)
                        vga_out_rgb <= (pattern[13])? on1 : off1;


                else if (p_cnt >= left_offset+horiz_distance*3 && p_cnt <=
left_offset+horiz_distance*3+width)
                        vga_out_rgb <= (pattern[12])? on1 : off1;
```

```
                else if (p_cnt >= left_offset+horiz_distance*4 && p_cnt <=
left_offset+horiz_distance*4+width)
                        vga_out_rgb <= (pattern[11])? on2 : off2;


                else if (p_cnt >= left_offset+horiz_distance*5 && p_cnt <=
left_offset+horiz_distance*5+width)
                        vga_out_rgb <= (pattern[10])? on2 : off2;


                else if (p_cnt >= left_offset+horiz_distance*6 && p_cnt <=
left_offset+horiz_distance*6+width)
                        vga_out_rgb <= (pattern[9])? on2 : off2;


                else if (p_cnt >= left_offset+horiz_distance*7 && p_cnt <=
left_offset+horiz_distance*7+width)
                        vga_out_rgb <= (pattern[8])? on2 : off2;


                else if (p_cnt >= left_offset+horiz_distance*8 && p_cnt <=
left_offset+horiz_distance*8+width)
                        vga_out_rgb <= (pattern[7])? on1 : off1;


                else if (p_cnt >= left_offset+horiz_distance*9 && p_cnt <=
left_offset+horiz_distance*9+width)
                        vga_out_rgb <= (pattern[6])? on1 : off1;


                else if (p_cnt >= left_offset+horiz_distance*10 && p_cnt <=
left_offset+horiz_distance*10+width)
                        vga_out_rgb <= (pattern[5])? on1 : off1;


                else if (p_cnt >= left_offset+horiz_distance*11 && p_cnt <=
left_offset+horiz_distance*11+width)
                        vga_out_rgb <= (pattern[4])? on1 : off1;


                else if (p_cnt >= left_offset+horiz_distance*12 && p_cnt <=
left_offset+horiz_distance*12+width)
                        vga_out_rgb <= (pattern[3])? on2 : off2;
```



```
                else if (p_cnt >= left_offset+horiz_distance*13 && p_cnt <=
left_offset+horiz_distance*13+width)
                        vga_out_rgb <= (pattern[2])? on2 : off2;


                else if (p_cnt >= left_offset+horiz_distance*14 && p_cnt <=
left_offset+horiz_distance*14+width)
                        vga_out_rgb <= (pattern[1])? on2 : off2;


                else if (p_cnt >= left_offset+horiz_distance*15 && p_cnt <=
left_offset+horiz_distance*15+width)
                        vga_out_rgb <= (pattern[0])? on2 : off2;









                else
                        // spacing blocks horizontally
                        vga_out_rgb <= 24'd0;
        else
                // spacing blocks vertically
                vga_out_rgb <= 24'd0;


        end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    17:59:47 05/04/06
// Design Name:
// Module Name:    bpmenable
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module bpmenable(clk, reset, inc_in, dec_in, enable, count16, bpm,
                                        // control signals
                                        play, stop, state);


        input clk;
   input reset;
        input inc_in, dec_in, play, stop;
        output state;
   output enable;
        output [3:0] count16;
        output [7:0] bpm;
```



```
        wire inc, dec;
        debounce_toggle dt1(.clock(clk), .in(inc_in), .out(inc));
        debounce_toggle dt2(.clock(clk), .in(dec_in), .out(dec));

        wire [15:0] bpm_squared;
        calc_new_bpm stepper(.clk(clk), .a(bpm), .b(bpm), .q(bpm_squared));

        wire [29:0] step;
        wire [15:0] remainder;



        // stepping increment = 3.77625e8 / bpm^2
        parameter STEP_CONST = 30'd377625000;
        divider div(.dividend(STEP_CONST), .divisor(bpm_squared),
                                        .quot(step), .remd(remainder), .clk(clk));


//parameter UPTO = 2'd2;
parameter UPTOdefault = 30'd3146880;  // bpm 120
        parameter UPTOmin = 30'd1897610; // bpm 199
        parameter UPTOmax = 30'd12587500;       // bpm 30

        // FSM - PLAY AND STOP
        parameter IDLE = 1'd0;
        parameter PLAYING = 1'd1;

        // bpm values
        parameter bpmdefault = 8'd120;
        parameter bpmmin = 8'd30;
        parameter bpmmax = 8'd199;

reg [29:0] count;
```

```
     reg [29:0] upto;
reg enable;

     reg [3:0] count16;
     reg [7:0] bpm;
     reg state, next;

always @ (posedge clk)

             if (reset)
                     begin
                       upto <= UPTOdefault;
                       count <= 0;
                       enable <= 0;
                       count16 <= 0;
                       bpm <= bpmdefault;
                       state <= IDLE;
                     end

             else // NOT IN RESET
                     begin
                     state <= next;
                     if (state == PLAYING)
                             // start beat counting
                             if (count >= upto)
                                begin
                            count <= 0;
                                enable <= 1;
                                        count16 <= count16 + 1;
                      end
```



```
                                else if (inc)
                                        begin
                                                upto <= (upto <= UPTOmin) ? upto :
upto - step;
                                                bpm <= (bpm == bpmmax) ? bpm : bpm +
1;
                                        end

                                else if (dec)
                                        begin
                                                upto <= (upto >= UPTOmax) ? upto :
upto + step;
                                                bpm <= (bpm == bpmmin) ? bpm : bpm -
1;
                                        end

                                else
                             begin
                                                enable <= 0;
                                                count <= count + 1;
                                        end

                        else    // not playing
                     begin
                                                enable <= 0;
                                                count <= 0;
                                                count16 <= 0;
                                        end
                                end

        always @ (state or play or stop)

                case (state)
                        IDLE:
```

```
                next = (play) ? PLAYING : IDLE;
        PLAYING:
                next = (stop) ? IDLE : PLAYING;
    endcase


endmodule
```



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    18:26:11 05/07/06
// Design Name:
// Module Name:    column_selector
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module column_selector(count16,
                                                    pattern0, pattern1,
pattern2, pattern3,
                                                    pattern4, pattern5,
pattern6, pattern7,
                                                    pattern8, pattern9,
pattern10, pattern11,
                                                    pattern12, pattern13,
pattern14, pattern15,
                                                    column_data);


input [3:0] count16;
input [15:0] pattern0, pattern1, pattern2, pattern3,
             pattern4, pattern5, pattern6, pattern7,
```

```
                pattern8, pattern9, pattern10, pattern11,
                pattern12, pattern13, pattern14, pattern15;
output [15:0] column_data;

parameter p15 = 4'd15;
parameter p14 = 4'd14;
parameter p13 = 4'd13;
parameter p12 = 4'd12;

parameter p11 = 4'd11;
parameter p10 = 4'd10;
parameter p9 = 4'd9;
parameter p8 = 4'd8;

parameter p7 = 4'd7;
parameter p6 = 4'd6;
parameter p5 = 4'd5;
parameter p4 = 4'd4;

parameter p3 = 4'd3;
parameter p2 = 4'd2;
parameter p1 = 4'd1;
parameter p0 = 4'd0;

// given the count16 - output the 1's and 0's at the column
assign column_data = (count16 == p15) ? {pattern0[0], pattern1[0], pattern2[0],
                pattern3[0], pattern4[0], pattern5[0],
                pattern6[0], pattern7[0], pattern8[0],
                pattern9[0], pattern10[0], pattern11[0],
                pattern12[0], pattern13[0], pattern14[0],
                pattern15[0]} :
```



```
(count16 == p14) ? {pattern0[1], pattern1[1], pattern2[1],
        pattern3[1], pattern4[1], pattern5[1],
        pattern6[1], pattern7[1], pattern8[1],
        pattern9[1], pattern10[1], pattern11[1],
        pattern12[1], pattern13[1], pattern14[1],
        pattern15[1]} :
(count16 == p13) ? {pattern0[2], pattern1[2], pattern2[2],
        pattern3[2], pattern4[2], pattern5[2],
        pattern6[2], pattern7[2], pattern8[2],
        pattern9[2], pattern10[2], pattern11[2],
        pattern12[2], pattern13[2], pattern14[2],
        pattern15[2]} :
(count16 == p12) ? {pattern0[3], pattern1[3], pattern2[3],
        pattern3[3], pattern4[3], pattern5[3],
        pattern6[3], pattern7[3], pattern8[3],
        pattern9[3], pattern10[3], pattern11[3],
        pattern12[3], pattern13[3], pattern14[3],
        pattern15[3]} :
(count16 == p11) ? {pattern0[4], pattern1[4], pattern2[4],
        pattern3[4], pattern4[4], pattern5[4],
        pattern6[4], pattern7[4], pattern8[4],
        pattern9[4], pattern10[4], pattern11[4],
        pattern12[4], pattern13[4], pattern14[4],
        pattern15[4]} :
(count16 == p10) ? {pattern0[5], pattern1[5], pattern2[5],
        pattern3[5], pattern4[5], pattern5[5],
        pattern6[5], pattern7[5], pattern8[5],
        pattern9[5], pattern10[5], pattern11[5],
        pattern12[5], pattern13[5], pattern14[5],
        pattern15[5]} :
(count16 == p9) ? {pattern0[6], pattern1[6], pattern2[6],
        pattern3[6], pattern4[6], pattern5[6],
```

```
        pattern6[6], pattern7[6], pattern8[6],
        pattern9[6], pattern10[6], pattern11[6],
        pattern12[6], pattern13[6], pattern14[6],
        pattern15[6]} :
(count16 == p8) ? {pattern0[7], pattern1[7], pattern2[7],
        pattern3[7], pattern4[7], pattern5[7],
        pattern6[7], pattern7[7], pattern8[7],
        pattern9[7], pattern10[7], pattern11[7],
        pattern12[7], pattern13[7], pattern14[7],
        pattern15[7]} :
(count16 == p7) ? {pattern0[8], pattern1[8], pattern2[8],
        pattern3[8], pattern4[8], pattern5[8],
        pattern6[8], pattern7[8], pattern8[8],
        pattern9[8], pattern10[8], pattern11[8],
        pattern12[8], pattern13[8], pattern14[8],
        pattern15[8]} :
(count16 == p6) ? {pattern0[9], pattern1[9], pattern2[9],
        pattern3[9], pattern4[9], pattern5[9],
        pattern6[9], pattern7[9], pattern8[9],
        pattern9[9], pattern10[9], pattern11[9],
        pattern12[9], pattern13[9], pattern14[9],
        pattern15[9]} :
(count16 == p5) ? {pattern0[10], pattern1[10], pattern2[10],
        pattern3[10], pattern4[10], pattern5[10],
        pattern6[10], pattern7[10], pattern8[10],
        pattern9[10], pattern10[10], pattern11[10],
        pattern12[10], pattern13[10], pattern14[10],
        pattern15[10]} :
(count16 == p4) ? {pattern0[11], pattern1[11], pattern2[11],
        pattern3[11], pattern4[11], pattern5[11],
        pattern6[11], pattern7[11], pattern8[11],
```



```
        pattern9[11], pattern10[11], pattern11[11],
        pattern12[11], pattern13[11], pattern14[11],
        pattern15[11]} :
(count16 == p3) ? {pattern0[12], pattern1[12], pattern2[12],
        pattern3[12], pattern4[12], pattern5[12],
        pattern6[12], pattern7[12], pattern8[12],
        pattern9[12], pattern10[12], pattern11[12],
        pattern12[12], pattern13[12], pattern14[12],
        pattern15[12]} :
(count16 == p2) ? {pattern0[13], pattern1[13], pattern2[13],
        pattern3[13], pattern4[13], pattern5[13],
        pattern6[13], pattern7[13], pattern8[13],
        pattern9[13], pattern10[13], pattern11[13],
        pattern12[13], pattern13[13], pattern14[13],
        pattern15[13]} :
(count16 == p1) ?
        {pattern0[14], pattern1[14], pattern2[14],
        pattern3[14], pattern4[14], pattern5[14],
        pattern6[14], pattern7[14], pattern8[14],
        pattern9[14], pattern10[14], pattern11[14],
        pattern12[14], pattern13[14], pattern14[14],
        pattern15[14]} :
// (count16 == 4'd0) ?
        {pattern0[15], pattern1[15], pattern2[15],
        pattern3[15], pattern4[15], pattern5[15],
        pattern6[15], pattern7[15], pattern8[15],
        pattern9[15], pattern10[15], pattern11[15],
        pattern12[15], pattern13[15], pattern14[15],
        pattern15[15]};


endmodule
```

```
// engineer: Hana Adaniya
`timescale 1ns / 1ps

module countersnare(clk, start, cnt, enable);
input clk, enable, start;
output [11:0] cnt;
reg [11:0] cnt;
reg state, next, cnt_enable;

parameter MAXCOUNT = 12'd3998;
parameter COUNTING = 0;
parameter RESTING = 1;

always @ (posedge clk)
        if (start)
                begin
                state <= COUNTING;
                cnt <= 12'b0;
                end
        else
        begin
                state <= next;
                cnt <= cnt + cnt_enable;
        end

always @ (state or cnt or enable or start) begin
        cnt_enable = 0;
        case (state)
                COUNTING:
                        if (cnt == MAXCOUNT)
                                begin
                                next = RESTING;
```

```
                cnt_enable = 0;
                end
        else
                begin
                        next = COUNTING;
                        cnt_enable = enable;
                end
RESTING:
        begin
                cnt_enable = 0;
                next = (start) ? COUNTING : RESTING;
    end
        endcase
end //always


endmodule
```



```
// Switch Debounce 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:    Shirley Fung
//
// Create Date:    18:47:08 05/04/06
// Design Name:
// Module Name:    debounce_toggle
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module debounce_toggle(clock, in, out);

input clock, in;
output out;
// IN and OUT must be clocked
// so that the output is on for only one clock cycle

reg prev_in;

always @ (posedge clock)

        prev_in <= in;
```



```
assign out = in & ~prev_in;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    14:30:51 04/23/06
// Design Name:
// Module Name:    debounce16
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module debounce16(reset, clock, noisy, clean);

// Switch Debounce Module - adapted from debounce
// use your system clock for the clock input
// to produce a synchronous, debounced output
   parameter DELAY = 270000;   // .01 sec with a 27Mhz clock
   input reset, clock;

        input [15:0] noisy;
   output [15:0] clean;

   reg [18:0] count;
   reg [15:0] new, clean;

   always @(posedge clock)
     if (reset)
       begin
          count <= 16'd0;
          new <= noisy;
          clean <= noisy;
       end
     else if (noisy != new)
       begin
```



```
          new <= noisy;
          count <= 16'd0;
       end
     else if (count == DELAY)
       clean <= new;
     else
       count <= count+1;


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Hana Adaniya
//
// Create Date:    14:25:11 04/23/06
// Design Name:
// Module Name:    display
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module display(pixel_clock, reset_sync,
                                    pattern_state, channel_id,
                                    count16,        bpm,
                                    column_data,
                                    state,
                                    vga_out_blank_b,
                                    vga_out_hblank,
                                    vga_out_hsync,
                                    vga_out_vblank,
                                    vga_out_vsync,
                                    vga_out_rgb,
                                    channel_sync,
```



```
                                    data_valid, // fft data
                                    fft_real, fft_img,
                                    fft_index_out);



input pixel_clock, reset_sync;
input [15:0] pattern_state;  // output from parameter module

input state;
input [15:0] count16;
input [15:0] column_data;
input [7:0] bpm;
input [3:0] channel_sync;       //input from channel selection switches

input data_valid;
input [16:0] fft_real, fft_img;
input [7:0] fft_index_out;

output [3:0] channel_id;  // query

// video signals - controlled by vgacount
output vga_out_blank_b, vga_out_hblank, vga_out_hsync,
                vga_out_vblank, vga_out_vsync;

output [23:0] vga_out_rgb;

wire [9:0] p_cnt, l_cnt;

vgacount counter(.pixel_31mhz(pixel_clock), .reset(reset_sync),
                                        .hblank(vga_out_hblank),
.hsync(vga_out_hsync),
```

```
                                        display
                                            // outputs for vga
                                            .vblank(vga_out_vblank),
.vsync(vga_out_vsync),
                                            // outputs for vga
                                            .p_cnt(p_cnt), .l_cnt(l_cnt));

                                            // outputs to FSM


wire [7:0] vga_out_red, vga_out_green, vga_out_blue;


displaynames instruments(.clk(pixel_clock), .p_cnt(p_cnt), .l_cnt(l_cnt),

.channel_sync(channel_sync),

.vga_out_red(vga_out_red),

.vga_out_green(vga_out_green),

.vga_out_blue(vga_out_blue),
                                                                        .bpm(bpm));


assign vga_out_blank_b = (vga_out_hblank && vga_out_vblank);


reg [23:0] vga_out_rgb_black;
reg [3:0] channel_id;
reg [15:0] current_pattern;


parameter top_space = 32-1;
parameter screen_width = 0;
parameter vert_distance = 16;


wire [23:0] vga_out_fft;
fft_display fft_display(.clk(pixel_clock), .reset(reset_sync),
.data_valid(data_valid),
                                    .fft_real_in(fft_real), .fft_img_in(fft_img),
```



```
                                        display
.fft_index_out(fft_index_out),
                                    .p_cnt(p_cnt), .l_cnt(l_cnt),
.vga_out_rgb(vga_out_fft));



always @ (posedge pixel_clock)


        if (reset_sync) // reset is true
                begin
                channel_id <= 4'd0;
                vga_out_rgb_black <= 24'd0;
                current_pattern <= 16'd0;
                end



        else // not during reset
                begin


                // when not drawing screen
                // get states from parameters module
                if (l_cnt == top_space-1)
                        begin
                        channel_id <= 4'd0;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance-1)
                        begin
```

```
                                display
                        channel_id <= 4'd1;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*2-1)
                        begin
                        channel_id <= 4'd2;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*3-1)
                        begin
                        channel_id <= 4'd3;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*4-1)
                        begin
```



```
                                display
                        channel_id <= 4'd4;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*5-1)
                        begin
                        channel_id <= 4'd5;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*6-1)
                        begin
                        channel_id <= 4'd6;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*7-1)
                        begin
```

```
                                display
                        channel_id <= 4'd7;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*8-1)
                        begin
                        channel_id <= 4'd8;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*9-1)
                        begin
                        channel_id <= 4'd9;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*10-1)
                        begin
```



```
                                display
                        channel_id <= 4'd10;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*11-1)
                        begin
                        channel_id <= 4'd11;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*12-1)
                        begin
                        channel_id <= 4'd12;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*13-1)
                        begin
```

```
                                display
                        channel_id <= 4'd13;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*14-1)
                        begin
                        channel_id <= 4'd14;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*15-1)
                        begin
                        channel_id <= 4'd15;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? pattern_state :       // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*16-1)
                        begin
```



```
                                display
                        //channel_id <= 4'd15;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? count16 :     // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else if (l_cnt == top_space+vert_distance*18-1)
                        begin
                        //channel_id <= 4'd15;
                        vga_out_rgb_black <= 24'd0;
                        current_pattern <= (p_cnt == screen_width+1) ? 16'd0 :
        // once off screen  set count id
                                                                        (p_cnt ==
screen_width+2) ? column_data : // next clock cycle register pattern state

                                                        current_pattern;
                        end


                else
                        begin
                        channel_id <= 4'd0;
                        current_pattern <= current_pattern;
                        vga_out_rgb_black <= 24'd0;
                        end


                end

        wire [23:0] vga_out_rgb_block, vga_out_rgb_play;


        blocks blocks16(.pixel_clk(pixel_clock), .reset_sync(reset_sync),
```

```
                                    .p_cnt(p_cnt), .l_cnt(l_cnt),
.pattern(current_pattern),
                                    .vga_out_rgb(vga_out_rgb_block));


playstopdisplay playstopdisplay(.clk(pixel_clock), .reset(reset_sync),

.p_cnt(p_cnt), .l_cnt(l_cnt), .play_state(state), .RGB(vga_out_rgb_play));



wire [23:0] vga_out_instr;
assign vga_out_instr = {vga_out_red, vga_out_green, vga_out_blue};


assign vga_out_rgb = (vga_out_rgb_block | vga_out_rgb_black | vga_out_instr |
vga_out_fft | vga_out_rgb_play);


endmodule
```



```
// engineer: Hana Adaniya
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Final Project: Sample Name Display
//
// This module connects all the letters together to form words by instantiating
//      letterselect with letterdisplay together
//
//////////////////////////////////////////////////////////////////////////////////
module displaynames(clk, p_cnt, l_cnt, channel_sync,
                                                vga_out_red, vga_out_green,
vga_out_blue,
                                                bpm);
        input clk;
        input [9:0] p_cnt, l_cnt;
        input [3:0] channel_sync;
        input [7:0] bpm;
        output [7:0] vga_out_red, vga_out_green, vga_out_blue;


        wire [23:0] RGB;
        wire [5:0] letter;
        wire hl_enable;


        parameter white = 24'b1111_1111_1111_1111_1111_1111;
        parameter highlight = 24'hFFFF33;



        letterselect letterselect(.clk(clk), .p_cnt(p_cnt), .l_cnt(l_cnt),
.bpm(bpm),

.letter(letter), .channel_sync(channel_sync), .hl_enable(hl_enable));
```

```
/*      always @ (posedge clk) begin
                if (current_row === channel_sync) hl_enable <= 1;
                else hl_enable <= 0;
        end*/

        letterdisplay letterdisplay(.clk(clk), .p_cnt(p_cnt), .l_cnt(l_cnt),
.letter(letter), .color(white), .highlight(highlight), .hl_enable(hl_enable),
.RGB(RGB));



        assign vga_out_red = RGB[23:16];
        assign vga_out_green = RGB[15:8];
        assign vga_out_blue = RGB[7:0];

endmodule
```



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    19:46:39 05/07/06
// Design Name:
// Module Name:    enable_minors
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module enable_minors(clock, column_data, enable, enable_minor_fsms);

input clock, enable;
input [15:0] column_data;
output [15:0] enable_minor_fsms; // MSB - CHANNEL0, LSB - CHANNEL16;

// column_data and enable_minor_fsms must be clocked
// so that the output is on for only one clock cycle

// enable signal must be delayed by at least 1 clock cycle (perhaps a few)
// because column_data is delayed by one clock cycle when count16 changes
```

```
reg enable_delay1;

reg enable_delay2;

reg [15:0] enable_minor_fsms1, enable_minor_fsms2, enable_minor_fsms3;


always @ (posedge clock)

        begin

        enable_delay1 <= enable;

        enable_delay2 <= enable_delay1;

        enable_minor_fsms1 <= column_data & { 16{enable_delay2} };      // bitwise
AND

        enable_minor_fsms2 <= enable_minor_fsms1;

        enable_minor_fsms3 <= enable_minor_fsms2;

        end

assign enable_minor_fsms = enable_minor_fsms1 | enable_minor_fsms2 |
enable_minor_fsms3;

endmodule
```



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    18:10:43 05/13/06
// Design Name:
// Module Name:    fft_controller
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module fft_controller(clk, reset, audio_input, data_valid, fft_real, fft_img,
fft_index_out);

input clk, reset;
input [11:0] audio_input;


output [16:0] fft_real, fft_img;
output [7:0] fft_index_out;

output data_valid;


wire [7:0] fft_index_in_wire, fft_index_out_wire;
wire ready_for_data_wire, busy_wire, data_valid_wire, early_done_wire, done_wire;


reg start_fft;


always @ (posedge clk)

        start_fft <= ~reset;



fft_core fft(
        .xn_re(audio_input),            // audio signal to be transformed,
                                                                        // always
real
        .xn_im(12'd0),                          // no imaginary signal
        .start(start_fft),              // always high for continuous processing
        .fwd_inv(1'd1),                         // forward tranform = 1, true
        .fwd_inv_we(reset),             // write enable for fwd_inv (active high)
        .clk(clk),
        .xk_re(fft_real),
        .xk_im(fft_img),
        .xn_index(fft_index_in_wire),
        .xk_index(fft_index_out),
        .rfd(ready_for_data_wire),
        .busy(busy_wire),
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    19:14:44 05/13/06
// Design Name:
// Module Name:    fft_display
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module fft_display(clk, reset, data_valid, fft_real_in, fft_img_in, fft_index_out,
                                                p_cnt, l_cnt, vga_out_rgb);

input clk, reset;
input data_valid;
input [16:0] fft_real_in, fft_img_in;
input [7:0] fft_index_out;
input [9:0] p_cnt, l_cnt;
output [23:0] vga_out_rgb;

reg [23:0] vga_out_rgb;
//reg [33:0] fft_mag [15:0];
```



```
        .dv(data_valid),
        .edone(early_done_wire),
        .done(done_wire));



endmodule
```

```
// registers for storing values
reg [16:0] fft_real [255:0];
reg [16:0] fft_img [255:0];

// register for serial computation of squares
// inputs
reg [16:0] fft_real_reg, fft_img_reg;
// outputs
wire [33:0] fft_real_sq_reg, fft_img_sq_reg;
wire [33:0] fft_mag_reg;

wire [17:0] fft_mag_sqrt;
reg [7:0] fft_mag_sqrt_reg [255:0];
// inputs and outputs are all registered
square17 sq1(.clk(clk), .a(fft_real_reg), .b(fft_real_reg), .q(fft_real_sq_reg));
square17 sq2(.clk(clk), .a(fft_img_reg), .b(fft_img_reg), .q(fft_img_sq_reg));
add34 add(.CLK(clk), .A(fft_real_sq_reg), .B(fft_img_sq_reg), .Q(fft_mag_reg));
sqrt34 sqrt(.clk(clk), .x_in(fft_mag_reg), .x_out(fft_mag_sqrt));

parameter top_space = 40;
parameter bottom_space = 296;
parameter left_space = 370;
parameter height = 256;  // 2^8
parameter width = 240;
parameter sq_width = 10;

parameter on = 24'h9966FF;
parameter off = 24'h000000;

integer i;
```



```
//integer j;
//integer k;

always @ (posedge clk)
        if (reset) // reset is high
                begin
                vga_out_rgb <= 24'd0;
                for (i=0; i<24; i=i+1)
                        begin
                        fft_mag_sqrt_reg[i] <= 8'd0;
                        fft_real[i] <= 17'd0;
                        fft_img[i] <= 17'd0;
                        end
                end

        else if (data_valid && l_cnt == 479) // load fft output to the registers;

                begin
                        fft_real[fft_index_out] <= fft_real_in;
                        fft_img[fft_index_out] <= fft_img_in;
                end

//      else if (l_cnt >= 480 && l_cnt < 504)
                /*for (j = 0; j < 24; j = j + 1)

                if (l_cnt == 480+j)

                        begin
                        // spit out values for calculation
                        fft_real_reg <= fft_real[j];
                        fft_img_reg <= fft_img[j];
                        //fft_mag[0] <= fft_mag_reg;
```

```
                fft_mag_sqrt_reg[j] <= fft_mag_sqrt[7:0];
                end*/

else if (l_cnt == 480)
// gives enough cycles to make sure mag is valid
        begin
                // spit out values for calculation
                fft_real_reg <= fft_real[0];
                fft_img_reg <= fft_img[0];
                //fft_mag[0] <= fft_mag_reg;
                fft_mag_sqrt_reg[0] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 481)
        begin
                fft_real_reg <= fft_real[1];
                fft_img_reg <= fft_img[1];
                //fft_mag[1] <= fft_mag_reg;
                fft_mag_sqrt_reg[1] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 482)
        begin
                fft_real_reg <= fft_real[2];
                fft_img_reg <= fft_img[2];
                //fft_mag[2] <= fft_mag_reg;
                fft_mag_sqrt_reg[2] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 483)
        begin
```



```
                fft_real_reg <= fft_real[3];
                fft_img_reg <= fft_img[3];
                //fft_mag[3] <= fft_mag_reg;
                fft_mag_sqrt_reg[3] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 484)
        begin
                fft_real_reg <= fft_real[4];
                fft_img_reg <= fft_img[4];
                //fft_mag[4] <= fft_mag_reg;
                fft_mag_sqrt_reg[4] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 485)
        begin
                fft_real_reg <= fft_real[5];
                fft_img_reg <= fft_img[5];
                //fft_mag[5] <= fft_mag_reg;
                fft_mag_sqrt_reg[5] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 486)
        begin
                fft_real_reg <= fft_real[6];
                fft_img_reg <= fft_img[6];
                //fft_mag[6] <= fft_mag_reg;
                fft_mag_sqrt_reg[6] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 487)
        begin
```

```
                fft_real_reg <= fft_real[7];
                fft_img_reg <= fft_img[7];
                //fft_mag[7] <= fft_mag_reg;
                fft_mag_sqrt_reg[7] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 488)
        begin
                fft_real_reg <= fft_real[8];
                fft_img_reg <= fft_img[8];
                //fft_mag[8] <= fft_mag_reg;
                fft_mag_sqrt_reg[8] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 489)
        begin
                fft_real_reg <= fft_real[9];
                fft_img_reg <= fft_img[9];
                //fft_mag[9] <= fft_mag_reg;
                fft_mag_sqrt_reg[9] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 490)
        begin
                fft_real_reg <= fft_real[10];
                fft_img_reg <= fft_img[10];
                //fft_mag[10] <= fft_mag_reg;
                fft_mag_sqrt_reg[10] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 491)
```



```
        begin
                fft_real_reg <= fft_real[11];
                fft_img_reg <= fft_img[11];
                //fft_mag[11] <= fft_mag_reg;
                fft_mag_sqrt_reg[11] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 492)
        begin
                fft_real_reg <= fft_real[12];
                fft_img_reg <= fft_img[12];
                //fft_mag[12] <= fft_mag_reg;
                fft_mag_sqrt_reg[12] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 493)
        begin
                fft_real_reg <= fft_real[13];
                fft_img_reg <= fft_img[13];
                //fft_mag[13] <= fft_mag_reg;
                fft_mag_sqrt_reg[13] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 494)
        begin
                fft_real_reg <= fft_real[14];
                fft_img_reg <= fft_img[14];
                //fft_mag[14] <= fft_mag_reg;
                fft_mag_sqrt_reg[14] <= fft_mag_sqrt[7:0];
        end

else if (l_cnt == 495)
```

```
        begin
                fft_real_reg <= fft_real[15];
                fft_img_reg <= fft_img[15];
                //fft_mag[15] <= fft_mag_reg;
                fft_mag_sqrt_reg[15] <= fft_mag_sqrt[7:0];
        end
else if (l_cnt == 496)
        begin
                fft_real_reg <= fft_real[16];
                fft_img_reg <= fft_img[16];
                //fft_mag[16] <= fft_mag_reg;
                fft_mag_sqrt_reg[16] <= fft_mag_sqrt[7:0];
        end
else if (l_cnt == 497)
        begin
                fft_real_reg <= fft_real[17];
                fft_img_reg <= fft_img[17];
                //fft_mag[17] <= fft_mag_reg;
                fft_mag_sqrt_reg[17] <= fft_mag_sqrt[7:0];
        end
else if (l_cnt == 498)
        begin
                fft_real_reg <= fft_real[18];
                fft_img_reg <= fft_img[18];
                //fft_mag[18] <= fft_mag_reg;
                fft_mag_sqrt_reg[18] <= fft_mag_sqrt[7:0];
        end
else if (l_cnt == 499)
        begin
                fft_real_reg <= fft_real[19];
                fft_img_reg <= fft_img[19];
```



```
                //fft_mag[19] <= fft_mag_reg;
                fft_mag_sqrt_reg[19] <= fft_mag_sqrt[7:0];
        end
else if (l_cnt == 500)
        begin
                fft_real_reg <= fft_real[20];
                fft_img_reg <= fft_img[20];
                //fft_mag[20] <= fft_mag_reg;
                fft_mag_sqrt_reg[20] <= fft_mag_sqrt[7:0];
        end
else if (l_cnt == 501)
        begin
                fft_real_reg <= fft_real[21];
                fft_img_reg <= fft_img[21];
                //fft_mag[21] <= fft_mag_reg;
                fft_mag_sqrt_reg[21] <= fft_mag_sqrt[7:0];
        end
else if (l_cnt == 502)
        begin
                fft_real_reg <= fft_real[22];
                fft_img_reg <= fft_img[22];
                //fft_mag[22] <= fft_mag_reg;
                fft_mag_sqrt_reg[22] <= fft_mag_sqrt[7:0];
        end
else if (l_cnt == 503)
        begin
                fft_real_reg <= fft_real[23];
                fft_img_reg <= fft_img[23];
                //fft_mag[23] <= fft_mag_reg;
                fft_mag_sqrt_reg[23] <= fft_mag_sqrt[7:0];
        end
// logic for displaying color
```

```
        else if (l_cnt >= top_space && l_cnt <= bottom_space)

/*              for (k = 0; k < 24; k = k + 1)
                        if (p_cnt >= left_space + sq_width*k && p_cnt < left_space +
sq_width*(k+1))
                                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[k] >=
bottom_space)? on : off;*/
                if (p_cnt >= left_space && p_cnt < left_space + sq_width)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[0] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width && p_cnt < left_space +
sq_width*2)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[1] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*2 && p_cnt < left_space +
sq_width*3)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[2] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*3 && p_cnt < left_space +
sq_width*4)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[3] >= bottom_space)?
on : off;

                /////////////////
                else if (p_cnt >= left_space + sq_width*4 && p_cnt < left_space +
sq_width*5)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[4] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*5 && p_cnt < left_space +
sq_width*6)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[5] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*6 && p_cnt < left_space +
```



```
sq_width*7)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[6] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*7 && p_cnt < left_space +
sq_width*8)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[7] >= bottom_space)?
on : off;

                /////////////////
                else if (p_cnt >= left_space + sq_width*8 && p_cnt < left_space +
sq_width*9)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[8] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*9 && p_cnt < left_space +
sq_width*10)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[9] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*10 && p_cnt < left_space +
sq_width*11)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[10] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*11 && p_cnt < left_space +
sq_width*12)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[11] >= bottom_space)?
on : off;



                /////////////////
                else if (p_cnt >= left_space + sq_width*12 && p_cnt < left_space +
sq_width*13)
                        vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[12] >= bottom_space)?
on : off;

                else if (p_cnt >= left_space + sq_width*13 && p_cnt < left_space +
```

```
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[13] >= bottom_space)?
on : off;

        else if (p_cnt >= left_space + sq_width*14 && p_cnt < left_space +
sq_width*15)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[14] >= bottom_space)?
on : off;

        else if (p_cnt >= left_space + sq_width*15 && p_cnt < left_space +
sq_width*16)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[15] >= bottom_space)?
on : off;



        /////////////////
        else if (p_cnt >= left_space + sq_width*16 && p_cnt < left_space +
sq_width*17)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[16] >= bottom_space)?
on : off;

        else if (p_cnt >= left_space + sq_width*17 && p_cnt < left_space +
sq_width*18)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[17] >= bottom_space)?
on : off;

        else if (p_cnt >= left_space + sq_width*18 && p_cnt < left_space +
sq_width*19)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[18] >= bottom_space)?
on : off;

        else if (p_cnt >= left_space + sq_width*19 && p_cnt < left_space +
sq_width*20)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[19] >= bottom_space)?
on : off;

        /////////////////
        else if (p_cnt >= left_space + sq_width*20 && p_cnt < left_space +
sq_width*21)
```



```
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[20] >= bottom_space)?
on : off;

        else if (p_cnt >= left_space + sq_width*21 && p_cnt < left_space +
sq_width*22)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[21] >= bottom_space)?
on : off;

        else if (p_cnt >= left_space + sq_width*22 && p_cnt < left_space +
sq_width*23)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[22] >= bottom_space)?
on : off;

        else if (p_cnt >= left_space + sq_width*23 && p_cnt < left_space +
sq_width*24)
                vga_out_rgb <= (l_cnt+fft_mag_sqrt_reg[23] >= bottom_space)?
on : off;



        else
                vga_out_rgb <= 24'd0;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    21:21:34 03/12/06
// Design Name:
// Module Name:    hcount
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module hcount(pixel_clk, reset,
                                hblank, hsync, hcount);

        input pixel_clk;
        input reset;
        output hblank, hsync;
        output [9:0] hcount;

        reg hblank, hsync;        // outputs are registered, free of glitches
        reg [9:0] count;

parameter hblank_b=800;
```



```
parameter hblank_a=640;
parameter hsync_a=656;
parameter hsync_c=752;

//parameter hblank_b=3;
//parameter hblank_a=3;
//parameter hsync_a=3;
//parameter hsync_c=3;

always @ (posedge pixel_clk)
        if (reset)
                 count <= 0;
        else
                count <= (count == hblank_b -1) ? 0 : count+1;

always @ (posedge pixel_clk)
        if (reset) begin
                 hblank <= 1;
                 hsync <= 1;
        end
        else
                begin
                if (count < hblank_a -1)
                        hblank <= 1;
           else
                        hblank <= 0;

                if ((count < hsync_a -1 ) || (count > hsync_c))
                        hsync <= 1;
           else
                        hsync <= 0;
```

```
    end


assign hcount = count;


endmodule
```



```
// LABKIT FILE ADOPTED FROM LAB4
// ENGINEERS: HANA ADANIYA AND SHIRLEY FUNG

//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module for Lab 4 (Spring 2006)
//
//
// Created: March 13, 2006
// 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;

input  rs232_rxd, rs232_cts;
output rs232_txd, rs232_rts;

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
// assign led = 8'hFF;

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

     assign user3[27:4] = 24'hZ;
assign user4[27:4] = 24'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;

// Logic Analyzer
assign analyzer1_data = 16'h0;
assign analyzer1_clock = 1'b1;
assign analyzer2_data = 16'h0;
assign analyzer2_clock = 1'b1;
assign analyzer3_data = 16'h0;
assign analyzer3_clock = 1'b1;
assign analyzer4_data = 16'h0;
assign analyzer4_clock = 1'b1;



////////////////////////////////////////////////////////////////////////////
//
// Lab 4 Components
//
////////////////////////////////////////////////////////////////////////////

//
// Generate a 31.5MHz 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 29
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 27

        // synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

   //
   // 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;

   // The following assignments should be deleted and replaced with your own
   // code to implement the Pong game.
   //assign vga_out_red = 8'h0;
   //assign vga_out_green = 8'h0;
   //assign vga_out_blue = 8'h0;
   //assign vga_out_blank_b = 1'b1;
   //assign vga_out_hsync = 1'b0;
   //assign vga_out_vsync = 1'b0;


         wire reset_sync;

         debounce debounce_reset(.reset(1'b0), .clock(pixel_clock),

                                                                  .noisy(~button0),
.clean(reset_sync));


         wire reprogram_sync;

         debounce debounce_reprogram(.reset(1'b0), .clock(pixel_clock),


.noisy(~button_enter), .clean(reprogram_sync));


         wire inc_sync;

         debounce debounce_inc(.reset(1'b0), .clock(pixel_clock),


.noisy(~button_up), .clean(inc_sync));


         wire dec_sync;
```

```
        debounce debounce_dec(.reset(1'b0), .clock(pixel_clock),
.noisy(~button_down), .clean(dec_sync));


        wire [15:0] pattern_sync;
        // testing w/o all buttons

        wire [15:0] switches;
        // switches[7:4] - 4 bits to try out registers
        // switches[3:2] - 2 bits to try out channel_sync
        // switches[1:0] - 2 bits to try out channel_id;

        // debounce all 16 toggles
        assign switches = {~user3[31:28], ~user3[3:0], ~user4[31:28], ~user4[3:0]};
        debounce16 debounce_pattern(.reset(reset_sync), .clock(pixel_clock),
.noisy(switches), .clean(pattern_sync));

        // switches[7:4] - 4 bits to try out registers
        // switches[3:2] - 2 bits to try out channel_sync
        // switches[1:0] - 2 bits to try out channel_id;

        // initial testing w/o all 16 switches
        // wire [15:0] test_pattern_sync;
        // assign test_pattern_sync = {12'b0, pattern_sync[7:4]};

        wire [3:0] channel_sync;
        wire [3:0] channel_id;
        assign channel_sync = switch[7:4];
        //assign channel_id = switch[3:0];
```



```
        wire enable;
        wire [3:0] count16;
        wire [7:0] bpm;

        wire play_sync, stop_sync;
        debounce debounce_play(.reset(1'b0), .clock(pixel_clock),
                                                                .noisy(~button1),
.clean(play_sync));
        debounce debounce_stop(.reset(1'b0), .clock(pixel_clock),
                                                                .noisy(~button2),
.clean(stop_sync));

        wire state_play;
        bpmenable bpmcounter(.clk(pixel_clock), .reset(reset_sync),
                                                            .inc_in(inc_sync),
.dec_in(dec_sync),
                                                            .play(play_sync),
.stop(stop_sync),
                                                            .enable(enable),
.count16(count16),
                                                            .bpm(bpm),
.state(state_play));

        wire recall_sync;
        debounce debounce_recall(.reset(1'b0), .clock(pixel_clock),
.noisy(~button_left), .clean(recall_sync));

        wire remember_sync;
        debounce debounce_remember(.reset(1'b0), .clock(pixel_clock),
                                                                .noisy(~button3),
.clean(remember_sync));

        wire [15:0] pattern_state, column_data;
        parameters param(// inputs
                                                .pixel_clock(pixel_clock),
```

```
.reset_sync(reset_sync),
                                        .pattern_sync(pattern_sync),
.channel_sync(channel_sync),
                                        .reprogram_sync(reprogram_sync),
                                        // outputs
                                        .pattern_state(pattern_state),
                                        // query for pattern_state
                                        .channel_id(channel_id),
                                        // outputs
                                        .column_data_reg(column_data),
                                        // query for pattern_state
                                        .count16(count16),
                                        .major_state(state_play),
                                        .recall_mode(recall_sync),
.remember(remember_sync));

        assign led[7:0] = 8'b1111_1111;

        wire [15:0] start_minor_fsms;
        enable_minors start_minors(.clock(pixel_clock),
.column_data(column_data), .enable(enable),
.enable_minor_fsms(start_minor_fsms));

        wire [15:0] dec_count16;
        decoder4to16 decoder(.in4(count16), .out16(dec_count16));
                // audio looping with BROM

        // top outputs to display module
        wire data_valid;
        wire [16:0] fft_real, fft_img;
        wire [7:0] fft_index_out;
```



```
        top top(.clk(pixel_clock), .reset(reset_sync), .start(start_minor_fsms),
.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),
                        .data_valid(data_valid), .fft_real(fft_real),
.fft_img(fft_img),
                        .fft_index_out(fft_index_out));

        wire [23:0] vga_out_rgb;
        display display_vga(.pixel_clock(pixel_clock), .reset_sync(reset_sync),
                                        .pattern_state(pattern_state),
.channel_id(channel_id),
                                        //.count16(dec_count16),
.bpm(fft_real[7:0]), .column_data(column_data),
                                        .count16(dec_count16), .bpm(bpm),
.column_data(column_data),
                                        .state(state_play),
                                        .vga_out_blank_b(vga_out_blank_b),
                                        .vga_out_hblank(vga_out_hblank),
                                        .vga_out_hsync(vga_out_hsync),
                                        .vga_out_vblank(vga_out_vblank),
                                        .vga_out_vsync(vga_out_vsync),
                                        .vga_out_rgb(vga_out_rgb),
                                        .channel_sync(channel_sync),
                                        .data_valid(data_valid),
.fft_real(fft_real), .fft_img(fft_img),
                                        .fft_index_out(fft_index_out));

        assign vga_out_red = vga_out_rgb[23:16];
   assign vga_out_green = vga_out_rgb[15:8];
   assign vga_out_blue = vga_out_rgb[7:0];
```

```
endmodule
```



```
// ENGINEER: HANA ADANIYA
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//      Final Project: letter to RGB output decoder
//
// This module takes in the letter from the letterselect module
// and deciphers what pixels should be set to the inputted color
//////////////////////////////////////////////////////////////////////////////////
module letterdisplay(clk, p_cnt, l_cnt, letter, color, highlight, hl_enable, RGB);
        input clk;
        input [9:0] p_cnt, l_cnt;
        input [5:0] letter;
        input [23:0] color;
        input [23:0] highlight;
        input hl_enable;
        output [23:0] RGB;

reg [2:0] offset_x, offset_y;
reg [23:0] RGB;
always @ (posedge clk) begin
        offset_x        <=      p_cnt % 8;
        offset_y <=     l_cnt % 8;
        case (letter)
                6'b0000: RGB <= 24'b0; //24'b0100_1111_0100_1111_0011_1111;
        //should make spaces black not gray
                6'b0001:        if      (               // letter A


                                                ((offset_x == 0) & ~((offset_y == 0)
| (offset_y == 7)))     |
                                                ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 3)))     |
                                                ((offset_x == 2) & ((offset_y == 0)
| (offset_y == 3)))     |
                                                ((offset_x == 3) & ((offset_y == 0)
```

```
| (offset_y == 3)))     |
| (offset_y == 7)))
color;
```

```
                                        ((offset_x == 4) & ~((offset_y == 0)
                                        )
                                        RGB <= (hl_enable)? highlight :
                                    else RGB <= 24'b0;
                6'b0010: if (   //letter B
                                        ((offset_x == 0) & (offset_y != 7))
|
                                        ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 3) | (offset_y == 6))) |
                                        ((offset_x == 2) & ((offset_y == 0)
| (offset_y == 3) | (offset_y == 6))) |
                                        ((offset_x == 3) & ((offset_y == 0)
| (offset_y == 3) | (offset_y == 6))) |
                                        ((offset_x == 4) & ((offset_y == 1)
| (offset_y == 2) | (offset_y == 4) | (offset_y == 5)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b0011: if     (       //letter C
                                        ((offset_x == 0) & ~((offset_y == 0)
| (offset_y == 6) | (offset_y == 7))) |
                                        ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 6))) |
                                        ((offset_x == 2) & ((offset_y == 0)
| (offset_y == 6))) |
                                        ((offset_x == 3) & ((offset_y == 0)
| (offset_y == 6))) |
                                        ((offset_x == 4) & ((offset_y == 1)
| (offset_y == 5)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b0100: if (   //letter D
                                        ((offset_x == 0) & (offset_y != 7))
```



```
|
| (offset_y == 6))) |
| (offset_y == 6))) |
| (offset_y == 6))) |
| (offset_y == 6) | (offset_y == 7)))
color;
```

```
                                        ((offset_x == 1) & ((offset_y == 0)
                                        ((offset_x == 2) & ((offset_y == 0)
                                        ((offset_x == 3) & ((offset_y == 0)
                                        ((offset_x == 4) & ~((offset_y == 0)
                                        )
                                        RGB <= (hl_enable)? highlight :
                                    else RGB <= 24'b0;
                6'b0101: if (   //letter E
                                        ((offset_x == 0) & (offset_y != 7))
|
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3) | (offset_x == 4)) &
                                                ((offset_y == 0) | (offset_y
== 3) | (offset_y == 6)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b0110: if (   //letter F
                                        ((offset_x == 0) & (offset_y != 7))
|
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) &
                                                ((offset_y == 0) | (offset_y
== 3)))  |
                                        ((offset_x == 4) & (offset_y == 0))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b0111: if (   //letter G
                                        ((offset_x == 0) & ~((offset_y == 0)
| (offset_y == 6) | (offset_y == 7))) |
```

```
                                        ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 6))) |
                                        (((offset_x == 2) | (offset_x == 3))
& ((offset_y == 0) | (offset_y == 3) | (offset_y == 6))) |
                                        ((offset_x == 4) & ((offset_y == 1)
| (offset_y == 3) | (offset_y == 4) | (offset_y == 5)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b1000:        if (    //letter H
                                        (((offset_x == 0) | (offset_x == 4))
& (offset_y != 7)) |
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & (offset_y == 3))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b1001: if (   //letter I
                                        (((offset_x == 0) | (offset_x == 1)
| (offset_x == 3) | (offset_x == 4)) & ((offset_y == 0) | (offset_y == 6))) |
                                        ((offset_x == 2) & (offset_y != 7))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b1010: if (   // letter J
                                        ((offset_x == 0) & (offset_y == 5))
|
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & (offset_y == 6)) |
                                        ((offset_x == 4) & ~((offset_y == 6)
| (offset_y == 7)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
```



```
                6'b1011: if (   // letter K
                                        ((offset_x == 0) & (offset_y != 7))
|
                                        ((offset_x == 1) & (offset_y == 3))
|
                                        ((offset_x == 2) & ((offset_y == 2)
| (offset_y == 4))) |
                                        ((offset_x == 3) & ((offset_y == 1)
| (offset_y == 5))) |
                                        ((offset_x == 4) & ((offset_y == 0)
| (offset_y == 6)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b1100: if (   // letter L
                                        ((offset_x == 0) & (offset_y != 7))
|
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3) | (offset_x == 4)) & (offset_y == 6))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b1101: if (   // letter M
                                        (((offset_x == 0) | (offset_x == 4))
& (offset_y != 7)) |
                                        (((offset_x == 1) | (offset_x == 3))
& (offset_y == 1)) |
                                        ((offset_x == 2) & (offset_y == 2))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b1110: if (   // letter N
                                        (((offset_x == 0) | (offset_x == 4))
& (offset_y != 7)) |
                                        ((offset_x == 1) & ((offset_y == 1)
```

```
| (offset_y == 2))) |
                                                ((offset_x == 2) & ((offset_y == 2)
| (offset_y == 3) | (offset_y == 4))) |
                                                ((offset_x == 3) & ((offset_y == 4)
| (offset_y == 5)))
                                                )
                                                RGB <= (hl_enable)? highlight :
color;
                                        else RGB <= 24'b0;
                6'b1111:        if (    // letter O
                                                (((offset_x == 0) | (offset_x == 4))
& ~((offset_y == 0) | (offset_y == 6) | (offset_y == 7))) |
                                                (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & ((offset_y == 0) | (offset_y == 6)))
                                                )
                                                RGB <= (hl_enable)? highlight :
color;
                                        else RGB <= 24'b0;
                6'b10000:if     (       // letter P
                                                ((offset_x == 0) & (offset_y != 7))
|
                                                (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & ((offset_y == 0) | (offset_y == 3))) |
                                                ((offset_x == 4) & ((offset_y == 1)
| (offset_y == 2)))
                                                )
                                                RGB <= (hl_enable)? highlight :
color;
                                        else RGB <= 24'b0;
                6'b10001:if     (       // letter Q
                                                ((offset_x == 0) & ~((offset_y == 0)
| (offset_y == 6) | (offset_y == 7))) |
                                                ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 6))) |
                                                ((offset_x == 2) & ((offset_y == 0)
| (offset_y == 4) | (offset_y == 6))) |
                                                ((offset_x == 3) & ((offset_y == 0)
| (offset_y == 5) | (offset_y == 6))) |
                                                ((offset_x == 4) & ~((offset_y == 0)
```



```
| (offset_y == 7)))
                                                )
                                                RGB <= (hl_enable)? highlight :
color;
                                        else RGB <= 24'b0;
                6'b10010:if     (       //letter R
                                                ((offset_x == 0) & (offset_y != 7))
|
                                                ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 3))) |
                                                ((offset_x == 2) & ((offset_y == 0)
| (offset_y == 3) | (offset_y == 4))) |
                                                ((offset_x == 3) & ((offset_y == 0)
| (offset_y == 3) | (offset_y == 5))) |
                                                ((offset_x == 4) & ((offset_y == 1)
| (offset_y == 2) | (offset_y == 6)))
                                                )
                                                RGB <= (hl_enable)? highlight :
color;
                                        else RGB <= 24'b0;
                6'b10011:if (   //letter S
                                                ((offset_x == 0) & ((offset_y == 1)
| (offset_y == 2) | (offset_y == 5))) |
                                                (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & ((offset_y == 0) | (offset_y == 3) | (offset_y == 6))) |
                                                ((offset_x == 4) & ((offset_y == 1)
| (offset_y == 4) | (offset_y == 5)))
                                                )
                                                RGB <= (hl_enable)? highlight :
color;
                                        else RGB <= 24'b0;
                6'b10100:if     (       //letter T
                                                (((offset_x == 0) | (offset_x == 1)
| (offset_x == 3) | (offset_x == 4)) & (offset_y == 0)) |
                                                ((offset_x == 2) & (offset_y != 7))
                                                )
                                                RGB <= (hl_enable)? highlight :
color;
```

```
                else RGB <= 24'b0;
        6'b10101:if     (       //letter U
                        (((offset_x == 0) | (offset_x == 4))
& ~((offset_y == 6) | (offset_y == 7))) |
                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & (offset_y == 6))
                        )
                        RGB <= (hl_enable)? highlight :
color;
                else RGB <= 24'b0;
        6'b10110:if     (       //letter V
                        (((offset_x == 0) | (offset_x == 4))
& ~((offset_y == 5) | (offset_y == 6) | (offset_y == 7))) |
                        (((offset_x == 1) | (offset_x == 3))
& (offset_y == 5)) |
                        ((offset_x == 2) & (offset_y == 6))
                        )
                        RGB <= (hl_enable)? highlight :
color;
                else RGB <= 24'b0;
        6'b10111:if     (       //letter W
                        (((offset_x == 0) | (offset_x == 4))
& (offset_y != 7)) |
                        (((offset_x == 1) | (offset_x == 3))
& (offset_y == 5)) |
                        ((offset_x == 2) & (offset_y == 4))
                        )
                        RGB <= (hl_enable)? highlight :
color;
                else RGB <= 24'b0;
        6'b11000:if     (       //letter X
                        (((offset_x == 0) | (offset_x == 4))
& ((offset_y == 0) | (offset_y == 1) | (offset_y == 5) | (offset_y == 6))) |
                        (((offset_x == 1) | (offset_x == 3))
& ((offset_y == 2) | (offset_y == 4))) |
                        ((offset_x == 2) & (offset_y == 3))
                        )
```



```
                        RGB <= (hl_enable)? highlight :
color;
                else RGB <= 24'b0;
        6'b11001:if     (       //letter Y
                        (((offset_x == 0) | (offset_x == 4))
& ((offset_y == 0) | (offset_y == 1) | (offset_y == 2))) |
                        (((offset_x == 1) | (offset_x == 3))
& (offset_y == 3)) |
                        ((offset_x == 2) & ((offset_y == 3)
| (offset_y == 4) | (offset_y == 5) | (offset_y == 6)))
                        )
                        RGB <= (hl_enable)? highlight :
color;
                else RGB <= 24'b0;
        6'b11010:if     (       //letter Z
                        ((offset_x == 0) & ((offset_y == 0)
| (offset_y == 5) | (offset_y == 6))) |
                        ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 4) | (offset_y == 6))) |
                        ((offset_x == 2) & ((offset_y == 0)
| (offset_y == 3) | (offset_y == 6))) |
                        ((offset_x == 3) & ((offset_y == 0)
| (offset_y == 2) | (offset_y == 6))) |
                        ((offset_x == 4) & ((offset_y == 0)
| (offset_y == 1) | (offset_y == 6)))
                        )
                        RGB <= (hl_enable)? highlight :
color;
                else RGB <= 24'b0;
        6'b11011:if     (       //number 1
                        ((offset_x == 0)        & ((offset_y
== 2) | (offset_y == 6))) |
                        ((offset_x == 1)        & ((offset_y
== 1) | (offset_y == 6))) |
                        ((offset_x == 2)        & (offset_y
!= 7)) |
                        (((offset_x == 3) | (offset_x == 4))
& (offset_y == 6))
                        )
```

```
                                letterdisplay
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b11100:if     (       //number 2
                                        ((offset_x == 0) & ((offset_y == 1)
| (offset_y == 6))) |
                                        ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 5) | (offset_y == 6))) |
                                        ((offset_x == 2) & ((offset_y == 0)
| (offset_y == 4) | (offset_y == 6))) |
                                        ((offset_x == 3) & ((offset_y == 0)
| (offset_y == 3) | (offset_y == 6))) |
                                        ((offset_x == 4) & ((offset_y == 1)
| (offset_y == 2) | (offset_y == 6)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b11101:if     (       //number 3
                                        ((offset_x == 0) & ((offset_y == 0)
| (offset_y == 5))) |
                                        ((offset_x == 1) & ((offset_y == 0)
| (offset_y == 6))) |
                                        ((offset_x == 2) & ((offset_y == 0)
| (offset_y == 2) | (offset_y == 6))) |
                                        ((offset_x == 3) & ((offset_y == 0)
| (offset_y == 1) | (offset_y == 3) | (offset_y == 6))) |
                                        ((offset_x == 4) & ((offset_y == 0)
| (offset_y == 4) | (offset_y == 5)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b11110:if     (       //number 4
                                        ((offset_x == 0) & (offset_y == 3))
|
                                        ((offset_x == 1) & ((offset_y == 2)
| (offset_y == 3))) |
                                        ((offset_x == 2) & ((offset_y == 1)
| (offset_y == 3))) |
```



```
                                letterdisplay
                                        ((offset_x == 3) & (offset_y != 7))
|
                                        ((offset_x == 4) & (offset_y == 3))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b11111:if     (       //number 5
                                        ((offset_x == 0) & ((offset_y == 0)
| (offset_y == 1) | (offset_y == 2) | (offset_y == 5))) |
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & ((offset_y == 0) | (offset_y == 2) | (offset_y == 6))) |
                                        ((offset_x == 4) & ((offset_y == 0)
| (offset_y == 3) | (offset_y == 4) | (offset_y == 5)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b100000:if    (       //number 6
                                        ((offset_x == 0) & ~((offset_y == 0)
| (offset_y == 6) | (offset_y == 7))) |
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & ((offset_y == 0) | (offset_y == 3) | (offset_y == 6))) |
                                        ((offset_x == 4) & ((offset_y == 1)
| (offset_y == 4) | (offset_y == 5)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
                6'b100001:if (  //number 7
                                        (((offset_x == 0) | (offset_x == 1)
| (offset_x == 2) | (offset_x == 3)) & (offset_y == 0)) |
                                        ((offset_x == 4) & (offset_y != 7))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                    else RGB <= 24'b0;
```

```
                6'b100010:if    (       //number 8
                                        (((offset_x == 0) | (offset_x == 4))
& ((offset_y == 1) | (offset_y == 2) | (offset_y == 4) | (offset_y == 5))) |
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & ((offset_y == 0) | (offset_y == 3) | (offset_y == 6)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                else RGB <= 24'b0;
                6'b100011:if    (       //number 9
                                        ((offset_x == 0) & ((offset_y == 1)
| (offset_y == 2) | (offset_y == 5))) |
                                        (((offset_x == 1) | (offset_x == 2)
| (offset_x == 3)) & ((offset_y == 0) | (offset_y == 3) | (offset_y == 6))) |
                                        ((offset_x == 4) & ~((offset_y == 0)
| (offset_y == 6) | (offset_y == 7)))
                                        )
                                        RGB <= (hl_enable)? highlight :
color;
                                else RGB <= 24'b0;
                default: RGB <= 24'b0;
        endcase
        end     //end always @ (posedge clk)


endmodule
```



```
// ENGINEER: HANA ADANIYA
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Final Project: Letter to bitmap decoder
//
// This module converts a 5-bit input to a 40-dot (1 character) bitmap representing
// the letters 'A' through 'Z'.
//
//////////////////////////////////////////////////////////////////////////////////
module letters(clk, letter, dots);
        input clk;
        input [4:0] letter;
        output [39:0] dots;

        reg [39:0] dots;
        always @ (posedge clk) begin
                case (letter)
                        5'd00: dots <=
{40'b01111110_10010000_10010000_10010000_01111110};     // A
                        5'd01: dots <=
{40'b11111110_10010010_10010010_10010010_01101100};     // B
                        5'd02: dots <=
{40'b01111100_10000010_10000010_10000010_01000100};     // C
                        5'd03: dots <=
{40'b11111110_10000010_10000010_10000010_01111100};     // D
                        default: dots <=
{40'b11111111_11111111_11111111_11111111_11111111};     // default full square
                endcase
        end


endmodule
```

```
// ENGINEER: HANA ADANIYA
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//      Final Project: position to character correspondance
//
//      This module uses pixel and line count to determine the
//      position and corresponding character to display on screen
//
//////////////////////////////////////////////////////////////////////////////////
module letterselect(clk, p_cnt, l_cnt, letter, bpm, channel_sync, hl_enable);
        input clk;
        input [9:0] p_cnt, l_cnt;
        input [7:0] bpm;
        output [5:0] letter;
        input [3:0] channel_sync;
        output hl_enable;

parameter A = 6'b000001;
parameter B = 6'b000010;
parameter C = 6'b000011;
parameter D = 6'b000100;
parameter E = 6'b000101;
parameter F = 6'b000110;
parameter G = 6'b000111;
parameter H     = 6'b001000;
parameter I     = 6'b001001;
parameter J     = 6'b001010;
parameter K     = 6'b001011;
parameter L     = 6'b001100;
parameter M     = 6'b001101;
parameter N     = 6'b001110;
parameter O     = 6'b001111;
```



```
parameter P     = 6'b010000;
parameter Q     = 6'b010001;
parameter R     = 6'b010010;
parameter S     = 6'b010011;
parameter T     = 6'b010100;
parameter U     = 6'b010101;
parameter V     = 6'b010110;
parameter W     = 6'b010111;
parameter X     = 6'b011000;
parameter Y     = 6'b011001;
parameter Z     = 6'b011010;
parameter ONE = 6'b011011;
parameter TWO = 6'b011100;
parameter THREE = 6'b011101;
parameter FOUR = 6'b011110;
parameter FIVE = 6'b011111;
parameter SIX = 6'b100000;
parameter SEVEN = 6'b100001;
parameter EIGHT = 6'b100010;
parameter NINE = 6'b100011;

parameter TRIANGLE = 6'b100100;
parameter SQUARE = 6'b100101;

parameter SPACE = 6'b000000;

parameter START_ROW = 4;
parameter VERTICAL_SPACE = 2;
parameter START_COLUMN = 2;
parameter HORIZONTAL_SPACE = 1;
```

```
wire [3:0] ONES, TENS;
wire [1:0] HUNDREDS;
// module to convert the 8 bit binary number - bpm
// to binary coded decimal
binary8tobcd bpm_convert(.A(bpm),
                                                        .ONES(ONES),
                                                        .TENS(TENS),

.HUNDREDS(HUNDREDS));

        reg [5:0] letter;
        reg [9:0] position_row, position_col;
        reg [4:0] current_row;
        reg hl_enable;

        always @ (posedge clk) begin
                position_col = p_cnt / 8;
                position_row = l_cnt / 8;
                current_row[4] <= 1'b0;
                if (position_row == START_ROW)  begin
                        current_row[3:0] <= 4'b0000;
//ONE
                        if (position_col == START_COLUMN)       letter <= ONE;
                        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= S;
                        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= N;
                        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= A;
                        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= R;
                        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= E;
                        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= ONE;
                        else letter <= SPACE;   //space
```



```
                end
        else if (position_row == START_ROW + VERTICAL_SPACE)    begin
                current_row[3:0] <= 4'b0001;                    //TWO
                if (position_col == START_COLUMN)       letter <= TWO;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= S;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= N;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= A;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= R;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= E;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= TWO;
                else letter <= SPACE;   //space
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*2) begin
                current_row[3:0] <= 4'b0010;
//THREE
                if (position_col == START_COLUMN)       letter <= THREE;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= K;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= I;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= C;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= K;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= D;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= R;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*8)
letter <= U;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*9)
letter <= M;
                else letter <= SPACE;   //space
```

```
        end
    else if (position_row == START_ROW + VERTICAL_SPACE*3)  begin
        current_row[3:0] <= 4'b0011;
//FOUR
        if (position_col == START_COLUMN)       letter <= FOUR;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= T;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= O;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= M;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= T;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= O;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= M;
        else letter <= SPACE;   //space
        end
    else if (position_row == START_ROW + VERTICAL_SPACE*4) begin
        current_row[3:0] <= 4'b0100;
//FIVE
        if (position_col == START_COLUMN)       letter <= FIVE;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= H;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= I;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= G;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= H;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= H;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= A;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*8)
letter <= T;
        else letter <= SPACE;
        end
```



```
    else if (position_row == START_ROW + VERTICAL_SPACE*5) begin
        current_row[3:0] <= 4'b0101;
//SIX
        if (position_col == START_COLUMN)       letter <= SIX;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= O;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= P;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= E;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= N;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= H;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= A;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*8)
letter <= T;
        else letter <= SPACE;
        end
    else if (position_row == START_ROW + VERTICAL_SPACE*6) begin
        current_row[3:0] <= 4'b0110;
        if (position_col == START_COLUMN)       letter <= SEVEN;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= C;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= L;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= O;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= S;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= E;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= D;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*8)
letter <= H;
        else if (position_col == START_COLUMN + HORIZONTAL_SPACE*9)
letter <= A;
```

```
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*10)
letter <= T;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*7) begin
                current_row[3:0] <= 4'b0111;
                if (position_col == START_COLUMN)       letter <= EIGHT;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= R;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= I;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= M;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*8) begin
                current_row[3:0] <= 4'b1000;
                if (position_col == START_COLUMN)       letter <= NINE;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= C;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= Y;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= M;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= B;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= A;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= L;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*9) begin
                current_row[3:0] <= 4'b1001;
                if (position_col == START_COLUMN - HORIZONTAL_SPACE*1)
letter <= ONE;
                else if (position_col == START_COLUMN)  letter <= O;
//ZERO;
```



```
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= R;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= I;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= D;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= E;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*10) begin
                current_row[3:0] <= 4'b1010;
                if (position_col == START_COLUMN - HORIZONTAL_SPACE*1)
letter <= ONE;
                else if (position_col == START_COLUMN)  letter <= ONE;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= C;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= L;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= A;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= P;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*11) begin
                current_row[3:0] <= 4'b1011;
                if (position_col == START_COLUMN - HORIZONTAL_SPACE*1)
letter <= ONE;
                else if (position_col == START_COLUMN)  letter <= TWO;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= B;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= A;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= S;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= S;
```

```
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= ONE;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*12) begin
                current_row[3:0] <= 4'b1100;
                if (position_col == START_COLUMN - HORIZONTAL_SPACE*1)
letter <= ONE;
                else if (position_col == START_COLUMN)  letter <= THREE;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= E;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= F;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= F;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= E;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= C;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= T;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*8)
letter <= S;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*13) begin
                current_row[3:0] <= 4'b1101;
                if (position_col == START_COLUMN - HORIZONTAL_SPACE*1)
letter <= ONE;
                else if (position_col == START_COLUMN)  letter <= FOUR;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= C;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= H;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= O;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= O;
```



```
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= M;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*14) begin
                current_row[3:0] <= 4'b1110;
                if (position_col == START_COLUMN - HORIZONTAL_SPACE*1)
letter <= ONE;
                else if (position_col == START_COLUMN)  letter <= FIVE;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= T;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
letter <= A;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= M;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= B;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= O;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= U;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*8)
letter <= R;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*9)
letter <= I;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*10)
letter <= N;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*11)
letter <= E;
                else letter <= SPACE;
                end
        else if (position_row == START_ROW + VERTICAL_SPACE*15) begin
                current_row[3:0] <= 4'b1111;
                if (position_col == START_COLUMN - HORIZONTAL_SPACE*1)
letter <= ONE;
                else if (position_col == START_COLUMN)  letter <= SIX;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= S;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*3)
```

```
letter <= C;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)
letter <= R;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*5)
letter <= A;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*6)
letter <= T;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*7)
letter <= C;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*8)
letter <= H;
                else letter <= SPACE;
                end


        else if (position_row == START_ROW + VERTICAL_SPACE*16) begin
                current_row[4] <= 1'b1;
                if (position_col == START_COLUMN)       letter <= B;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*1)
letter <= P;
                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*2)
letter <= M;


                else if (position_col == START_COLUMN + HORIZONTAL_SPACE*4)

                        // HUNDREDS position
                        letter <= (HUNDREDS == 2'b01) ? ONE :
                                                (HUNDREDS == 2'b10) ? TWO :
// should never be true

                  SPACE;
                else if (position_col == 7)
                        // TENS position
                        letter <= (TENS == 4'd0) ? O :
                                                (TENS == 4'd1) ? ONE :
                                                (TENS == 4'd2) ? TWO :
                                                (TENS == 4'd3) ? THREE :
                                                (TENS == 4'd4) ? FOUR :
```



```
                                                (TENS == 4'd5) ? FIVE :
                                                (TENS == 4'd6) ? SIX :
                                                (TENS == 4'd7) ? SEVEN :
                                                (TENS == 4'd8) ? EIGHT :
                                                (TENS == 4'd9) ? NINE :
                SPACE;
                        else if (position_col == 8)
                                // ONES position
                                letter <= (ONES == 4'd0) ? O :
                                                (ONES == 4'd1) ? ONE :
                                                (ONES == 4'd2) ? TWO :
                                                (ONES == 4'd3) ? THREE :
                                                (ONES == 4'd4) ? FOUR :
                                                (ONES == 4'd5) ? FIVE :
                                                (ONES == 4'd6) ? SIX :
                                                (ONES == 4'd7) ? SEVEN :
                                                (ONES == 4'd8) ? EIGHT :
                                                (ONES == 4'd9) ? NINE :
                SPACE;
                        else letter <= SPACE;
                        end
                else letter <= SPACE;   // space

                if (current_row[3:0] == channel_sync && current_row[4] != 1'b1)
hl_enable <= 1;
                else hl_enable <= 0;

        end //end begin for always
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    SHIRLEY FUNG
//
// Create Date:    21:53:55 05/08/06
// Design Name:
// Module Name:    mixer
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module mixer(clock, dout0, dout1, dout2, dout3, dout4,
                                                dout5, dout6, dout7, dout8, dout9,
                                                dout10, dout11, dout12, dout13,
dout14, dout15,
                                                mixed_dout);



input clock;
input [7:0] dout0, dout1, dout2, dout3, dout4,
        dout5, dout6, dout7, dout8, dout9,
        dout10, dout11, dout12, dout13, dout14, dout15;
output [11:0] mixed_dout;
```



```
wire [8:0] add1, add2, add3, add4, add5, add6, add7, add8;
wire [9:0] add9, add10, add11, add12;
wire [10:0] add13, add14;
wire [11:0] add15;

// first top layer
adder1 a1(.A(dout0), .B(dout1), .Q(add1), .CLK(clock));
adder1 a2(.A(dout2), .B(dout3), .Q(add2), .CLK(clock));
adder1 a3(.A(dout4), .B(dout5), .Q(add3), .CLK(clock));
adder1 a4(.A(dout6), .B(dout7), .Q(add4), .CLK(clock));

adder1 a5(.A(dout8), .B(dout9), .Q(add5), .CLK(clock));
adder1 a6(.A(dout10), .B(dout11), .Q(add6), .CLK(clock));
adder1 a7(.A(dout12), .B(dout13), .Q(add7), .CLK(clock));
adder1 a8(.A(dout14), .B(dout15), .Q(add8), .CLK(clock));

// second layer
adder2 a9(.A(add1), .B(add2), .Q(add9), .CLK(clock));
adder2 a10(.A(add3), .B(add4), .Q(add10), .CLK(clock));

adder2 a11(.A(add5), .B(add6), .Q(add11), .CLK(clock));
adder2 a12(.A(add7), .B(add8), .Q(add12), .CLK(clock));

// third layer
adder3 a13(.A(add9), .B(add10), .Q(add13), .CLK(clock));
adder3 a14(.A(add11), .B(add12), .Q(add14), .CLK(clock));

// last layer
adder4 a15(.A(add13), .B(add14), .Q(add15), .CLK(clock));

assign mixed_dout = add15;
```

```
endmodule
```



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:             Shirley Fung
//
// Create Date:    13:24:22 04/23/06
// Design Name:    Parameters for the 16 channels (state registers)
// Module Name:    parameters
// Project Name:   Pattern Sequencer
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module parameters(pixel_clock, reset_sync,
                                          pattern_sync, channel_sync,
                                          reprogram_sync,
                                          pattern_state,  // for display
                                          channel_id,             // for
display
                                          count16,                        //
for counter
                                          column_data_reg,        // for
counter
                                          //start_minor_fsms,
                                          major_state,
                                          recall_mode, remember);
```

```
input   pixel_clock;     // from DCM
input reset_sync;                // from debounce
input [15:0] pattern_sync;      // 16-bit representing the state
input [3:0]     channel_sync;   // channel identifier
input reprogram_sync; // button
input recall_mode, remember; // swaps out recall_pattern grid

input [3:0] channel_id;                 // given channel_id
output [15:0] pattern_state;    // output the state with that channel
//output [15:0] start_minor_fsms; // MSB - CHANNEL0, LSB - CHANNEL16;

reg [15:0] current_pattern;
reg [15:0] pattern_state;

input major_state; // enable column_data_reg to be non zero.
input [3:0] count16;
output reg [15:0] column_data_reg; // top (MSB) to bottom (LSB)

// 16x16 state grid

reg [15:0] pattern [15:0];
reg [15:0] recall_pattern [15:0];
         // MSB - LEFT TOGGLE
         // LSB - RIGHT TOGGLE


integer i;

wire [15:0] column_data;
column_selector column(count16,
                                        pattern[0], pattern[1],
```



```
pattern[2], pattern[3],
                                                pattern[4], pattern[5],
pattern[6], pattern[7],
                                                pattern[8], pattern[9],
pattern[10], pattern[11],
                                                pattern[12], pattern[13],
pattern[14], pattern[15],
                                                column_data);


always @ (posedge pixel_clock)
        begin
        if (reset_sync) // reset is high, reset all registers
                begin
                        pattern[0] <= 16'd0;
                        pattern[1] <= 16'd0;
                        pattern[2] <= 16'd0;
                        pattern[3] <= 16'd0;

                        pattern[4] <= 16'd0;
                        pattern[5] <= 16'd0;
                        pattern[6] <= 16'd0;
                        pattern[7] <= 16'd0;

                        pattern[8] <= 16'd0;
                        pattern[9] <= 16'd0;
                        pattern[10] <= 16'd0;
                        pattern[11] <= 16'd0;

                        pattern[12] <= 16'd0;
                        pattern[13] <= 16'd0;
                        pattern[14] <= 16'd0;
                        pattern[15] <= 16'd0;
```

```
                recall_pattern[0] <= 16'd0;
                recall_pattern[1] <= 16'd0;
                recall_pattern[2] <= 16'd0;
                recall_pattern[3] <= 16'd0;

                recall_pattern[4] <= 16'd0;
                recall_pattern[5] <= 16'd0;
                recall_pattern[6] <= 16'd0;
                recall_pattern[7] <= 16'd0;

                recall_pattern[8] <= 16'd0;
                recall_pattern[9] <= 16'd0;
                recall_pattern[10] <= 16'd0;
                recall_pattern[11] <= 16'd0;

                recall_pattern[12] <= 16'd0;
                recall_pattern[13] <= 16'd0;
                recall_pattern[14] <= 16'd0;
                recall_pattern[15] <= 16'd0;

                current_pattern <= 16'd0;
                pattern_state <= 16'd0;
                column_data_reg <= 16'd0;

        end
else // reset isn't high, update values accordingly
        begin
                if (reprogram_sync) // reprogram button is pressed
                        // change the pattern in the state grid
                        begin
                        // current_pattern stays the same
                        current_pattern <= current_pattern;
```



```
                column_data_reg <= 16'd0;

                // pattern output is 0 (default)
                pattern_state <= 16'd0;

                // update accordingly
                pattern[channel_sync] <= pattern_sync;
                end

        else if (remember)
                begin
                recall_pattern[0] <= pattern[0];
                recall_pattern[1] <= pattern[1];
                recall_pattern[2] <= pattern[2];
                recall_pattern[3] <= pattern[3];
                recall_pattern[4] <= pattern[4];
                recall_pattern[5] <= pattern[5];
                recall_pattern[6] <= pattern[6];
                recall_pattern[7] <= pattern[7];
                recall_pattern[8] <= pattern[8];
                recall_pattern[9] <= pattern[9];
                recall_pattern[10] <= pattern[10];
                recall_pattern[11] <= pattern[11];
                recall_pattern[12] <= pattern[12];
                recall_pattern[13] <= pattern[13];
                recall_pattern[14] <= pattern[14];
                recall_pattern[15] <= pattern[15];
                end
        else if (recall_mode)
                begin
                pattern[0] <= recall_pattern[0];
```

```
                parameters
            pattern[1] <= recall_pattern[1];
            pattern[2] <= recall_pattern[2];
            pattern[3] <= recall_pattern[3];
            pattern[4] <= recall_pattern[4];
            pattern[5] <= recall_pattern[5];
            pattern[6] <= recall_pattern[6];
            pattern[7] <= recall_pattern[7];
            pattern[8] <= recall_pattern[8];
            pattern[9] <= recall_pattern[9];
            pattern[10] <= recall_pattern[10];
            pattern[11] <= recall_pattern[11];
            pattern[12] <= recall_pattern[12];
            pattern[13] <= recall_pattern[13];
            pattern[14] <= recall_pattern[14];
            pattern[15] <= recall_pattern[15];
            end
        else
            begin
            // reprogram is low
            // set current pattern only
            current_pattern <= pattern_sync;
            column_data_reg <= (major_state) ? column_data :
16'd0;

            // while not reprogramming
            // display module may query for pattern
            // by giving the channel id

            // if it's not the channel currently
            // being reprogrammed
            if      (channel_id != channel_sync)
                // output registered values
```



```
                parameters
                pattern_state <= pattern[channel_id];

            else // channel_id == channel_sync
                // output current pattern
                // so that the display automatically shows
                // the current input being programmed
                pattern_state <= current_pattern;

            end
        end
    end


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    HANA ADANIYA
//
// Create Date:    16:56:03 05/15/06
// Design Name:
// Module Name:    playstopdisplay
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module playstopdisplay(clk, reset, p_cnt, l_cnt, play_state, RGB);

        input clk, reset, play_state;
        input [9:0] p_cnt, l_cnt;
        output [23:0] RGB;

        reg [3:0] offset_x, offset_y;

        parameter on = 24'hFF0033;
        parameter off = 24'hCCCCCC;

        reg [23:0] RGB;

always @ (posedge clk) begin
        offset_x        <=      p_cnt % 16;
        offset_y <=     l_cnt % 16;
        if (reset)
                RGB <= 24'h000000;
        else if (p_cnt >= 480 & p_cnt < 496 & l_cnt >= 368 & l_cnt < 384)
//placement of play button
                if (
                        ((offset_x == 0) | (offset_x == 1) | (offset_x == 2) |
(offset_x == 3) | (offset_x == 4)) |
                        ((offset_x == 5) & ~((offset_y == 0) | (offset_y == 15))) |
                        ((offset_x == 6) & ~((offset_y == 0) | (offset_y == 1) |
(offset_y == 14) | (offset_y == 15))) |
                        ((offset_x == 7) & ~((offset_y == 0) | (offset_y == 1) |
(offset_y == 2) |

(offset_y == 13) | (offset_y == 14) | (offset_y == 15))) |
                        ((offset_x == 8) & ~((offset_y == 0) | (offset_y == 1) |
(offset_y == 2) | (offset_y == 3) |

(offset_y == 12) | (offset_y == 13) | (offset_y == 14) | (offset_y == 15))) |
                        ((offset_x == 9) & ((offset_y == 5) | (offset_y == 6) |
(offset_y == 7) | (offset_y == 8) | (offset_y == 9) | (offset_y == 10))) |
                        ((offset_x == 10) & ((offset_y == 6) | (offset_y == 7) |
(offset_y == 8) | (offset_y == 9))) |
                        ((offset_x == 11) & ((offset_y == 7) | (offset_y == 8)))
                        )
                        RGB <= (play_state)? on : off;
                else
                        RGB <= 24'h000000;
```



```
        else if (p_cnt >= 512 & p_cnt < 528 & l_cnt >= 368 & l_cnt < 384)         //
placement of stop button
                RGB <= (play_state)? off: on;
        else
                RGB <= 24'h000000;

end     //end always



endmodule
```

```
// ENGINEER: HANA ADANIYA
`timescale 1ns / 1ps

module snarefsm(clk, start, enable, sample);
input clk, start, enable;
output [7:0] sample;
wire [7:0] dout;
wire [11:0] addr;

countersnare counter(.clk(clk), .start(start), .cnt(addr), .enable(enable));

snare snarerom(.addr(addr), .clk(clk), .dout(dout));

assign sample = dout;

endmodule
```



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Hana Adaniya
//
// Create Date:    19:28:46 05/02/06
// Design Name:
// Module Name:    top
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module top(clk, reset, start, audio_reset_b, ac97_sdata_out,
           ac97_sdata_in, ac97_synch, ac97_bit_clock,
                          data_valid,
                          fft_real, fft_img,
                          fft_index_out);

input clk, reset;
input [15:0] start;
output audio_reset_b;
output ac97_sdata_out;
input ac97_sdata_in;
output ac97_synch;
```

```
input ac97_bit_clock;


output data_valid;
output [16:0] fft_real, fft_img;
output [7:0] fft_index_out;


wire [14:0] cnt;
wire cl_reset;          //clean/debounced reset signal
wire [11:0] mixer_out;
reg previous_synch;


reg [15:0] start_reg; // synchronizer


always @ (posedge ac97_bit_clock)
begin
        previous_synch <= ac97_synch;
        start_reg <= start;
end


wire enable;
assign enable = (!previous_synch && ac97_synch);


debounce debounce(.reset(1'b0), .clock(clk), .noisy(reset), .clean(cl_reset));


wire [7:0] sample0, sample1, sample2, sample3, sample4, sample5,
                                sample6, sample7, sample8, sample9, sample10,
sample11,
                                sample12, sample13, sample14, sample15;


snarefsm snarefsm(.clk(ac97_bit_clock), .start(start_reg[15]), .enable(enable),
.sample(sample0));
```



```
snare2fsm snare2fsm(.clk(ac97_bit_clock), .start(start_reg[14]), .enable(enable),
.sample(sample1));


kickfsm kickfsm(.clk(ac97_bit_clock), .start(start_reg[13]), .enable(enable),
.sample(sample2));


tomfsm tomfsm(.clk(ac97_bit_clock), .start(start_reg[12]), .enable(enable),
.sample(sample3));


highhatfsm highhatfsm(.clk(ac97_bit_clock), .start(start_reg[11]), .enable(enable),
.sample(sample4));


openhatfsm openhatfsm(.clk(ac97_bit_clock), .start(start_reg[10]), .enable(enable),
.sample(sample5));


closedhatfsm closedhatfsm(.clk(ac97_bit_clock), .start(start_reg[9]),
.enable(enable), .sample(sample6));


rimfsm rimfsm(.clk(ac97_bit_clock), .start(start_reg[8]), .enable(enable),
.sample(sample7));


cymbalfsm cymbalfsm(.clk(ac97_bit_clock), .start(start_reg[7]), .enable(enable),
.sample(sample8));


ridefsm ridefsm(.clk(ac97_bit_clock), .start(start_reg[6]), .enable(enable),
.sample(sample9));


clapfsm clapfsm(.clk(ac97_bit_clock), .start(start_reg[5]), .enable(enable),
.sample(sample10));


bassfsm bassfsm(.clk(ac97_bit_clock), .start(start_reg[4]), .enable(enable),
.sample(sample11));


vazbassfsm vazbassfsm(.clk(ac97_bit_clock), .start(start_reg[3]), .enable(enable),
.sample(sample12));


choomfsm choomfsm(.clk(ac97_bit_clock), .start(start_reg[2]), .enable(enable),
.sample(sample13));
```

```
tambourinefsm tambourine(.clk(ac97_bit_clock), .start(start_reg[1]),
.enable(enable), .sample(sample14));


scratchfsm scratch(.clk(ac97_bit_clock), .start(start_reg[0]), .enable(enable),
.sample(sample15));



wire [19:0] pcm_data;


//sinewave sinewave (.clock(clk), .ready(enable), .pcm_data(pcm_data));



mixer mix(.clock(ac97_bit_clock), .dout0(sample0), .dout1(sample1),
                                                .dout2(sample2), .dout3(sample3),
.dout4(sample4),
                                                .dout5(sample5), .dout6(sample6),
.dout7(sample7),
                                                .dout8(sample8), .dout9(sample9),
.dout10(sample10),
                                                .dout11(sample11),
.dout12(sample12), .dout13(sample13),
                                                .dout14(sample14),
.dout15(sample15),
                                                .mixed_dout(mixer_out));




wire [19:0] mixed_dout;
assign mixed_dout = { mixer_out , 8'd0 };


fft_controller fft_controller(.clk(clk),

.reset(reset),

.audio_input(mixer_out),
```



```
.data_valid(data_valid),

.fft_real(fft_real),

.fft_img(fft_img),

.fft_index_out(fft_index_out));




audio audio(.reset(cl_reset), .clock_27mhz(clk), .audio_reset_b(audio_reset_b),
                                    .ac97_sdata_out(ac97_sdata_out),
                                    .ac97_sdata_in(ac97_sdata_in),
//tie audio input to tom file in memory
                                    .sample(mixed_dout),
                            //tie audio input to tom file in memory
                .ac97_synch(ac97_synch), .ac97_bit_clock(ac97_bit_clock));




endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    22:05:22 03/12/06
// Design Name:
// Module Name:    vcount
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module vcount(pixel_clk, reset,
                                hcount,
                                vblank, vsync,
                                vcount);

        input pixel_clk;
        input reset;
        input [9:0] hcount;
        output vblank, vsync;
        output [9:0] vcount;

        reg vblank, vsync;        // outputs are registered, free of glitches
```



```
        reg [9:0] count;

parameter hcount_max=800;
parameter vblank_b=524;
parameter vblank_a=480;
parameter vsync_a=491;
parameter vsync_c=493;

//parameter hcount_max=3;
//parameter vblank_b=3;
//parameter vblank_a=3;
//parameter vsync_a=3;
//parameter vsync_c=3;

always @ (posedge pixel_clk)
        if (reset)
                 count <= 0;
        else
                count <= (count == vblank_b -1) ? 0 :
                                        (hcount == hcount_max-1) ? count+1: count;

always @ (posedge pixel_clk)
        if (reset) begin
                 vblank <= 1;
                 vsync <= 1;
        end
        else
                begin
                if (count < vblank_a -1)
                        vblank <= 1;
           else
```

```
                vblank <= 0;


        if ((count < vsync_a -1 ) || (count > vsync_c))
                vsync <= 1;
    else
                vsync <= 0;
        end


assign vcount = count;
endmodule
```



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:    Shirley Fung
//
// Create Date:    22:33:26 03/12/06
// Design Name:
// Module Name:    vgacount
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module vgacount(pixel_31mhz, reset,
                                        hblank, hsync,                  // outputs
for vga
                                        vblank, vsync,                  // outputs
for vga

                                        p_cnt, l_cnt);                  // outputs
to FSM

        input pixel_31mhz, reset;
        output hblank, hsync, vblank, vsync;

        output [9:0] p_cnt;

        output [9:0] l_cnt;

        wire [9:0] hcount_wire;

        wire [9:0] vcount_wire;

/*      hcountsml hcount(.pixel_clk(pixel_31mhz), .reset(reset),
                                                .hblank(hblank),
.hsync(hsync), .hcount(hcount_wire));

        vcountsml vcount(.pixel_clk(pixel_31mhz), .reset(reset),
                                                .hcount(hcount_wire),
.vcount(vcount_wire),
                                                .vblank(vblank), .vsync(vsync));*/




        hcount hcount(.pixel_clk(pixel_31mhz), .reset(reset),
                                        .hblank(hblank), .hsync(hsync),
.hcount(hcount_wire));

        vcount vcount(.pixel_clk(pixel_31mhz), .reset(reset),
                                        .hcount(hcount_wire), .vcount(vcount_wire),
                                        .vblank(vblank), .vsync(vsync));
```

```
        assign p_cnt = hcount_wire;
        assign l_cnt = vcount_wire;

endmodule
```