```
//////////////////////////////////////////////////////////////////////////////
//
// 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;

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

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

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

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

   inout [43:0] daughtercard;

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

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

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

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

   // Video Output
   assign tv_out_ycrcb = 10'h0;
   assign tv_out_reset_b = 1'b0;
   assign tv_out_clock = 1'b0;
```

```
   assign tv_out_i2c_clock = 1'b0;
   assign tv_out_i2c_data = 1'b0;
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
//   assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b0;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b0;
//   assign tv_in_reset_b = 1'b0;
//   assign tv_in_clock = 1'b0;
       assign tv_in_clock = clock_27mhz;
//   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[4:1] = 4'b1111;

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

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

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

   // Logic Analyzer

        wire [11:0] rgbdata;
        wire [9:0] pixel_count_cam;
        wire [9:0] line_count_cam;

   assign analyzer1_data = 16'h0;
   assign analyzer1_clock = 1'b1;
//   assign analyzer2_data = 16'h0;
//   assign analyzer2_clock = 1'b1;
//        assign analyzer2_data = {up, up_clean, down, down_clean, right, right_clean, 10'h0};
//        assign analyzer2_clock = clock_27mhz;
   assign analyzer3_data = 16'h0;
   assign analyzer3_clock = 1'b1;
//   assign analyzer4_data = 16'h0;
//   assign analyzer4_clock = 1'b1;
//        assign analyzer4_data = 16'h0;
//   assign analyzer4_clock = 1'b1;//clock_27mhz;
```

```
   ////////////////////////////////////////////////////////////////////////
   //
   // Project Components
   //
   ////////////////////////////////////////////////////////////////////////

        // Initialize ADV7185
        adv7185init adcinit(.reset(1'b0),
                                                    .clock_27mhz(clock_27mhz),
                                                    .source(1'b0),
                                                    .tv_in_reset_b(tv_in_reset_b),
                                                    .tv_in_i2c_clock(tv_in_i2c_clock),
                                                    .tv_in_i2c_data(tv_in_i2c_data));

   //
   // 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 6
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

        // 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;

        wire [23:0] vga_out_RGB;

        //statements like these helped find suitable color ranges for the red, green and blue of the laser pointer

        assign vga_out_blue = (switch[0]) ? 8'h00:vga_out_RGB[7:0];
        assign vga_out_red = (switch[0]) ? ((switch[3]) ? ((vga_out_RGB[15:8] >= {switch[7:4],4'hf}) ? 8'hff: 8'h00):
                                                          ((vga_out_RGB[7:0] >= {switch[7:4],4'hf}) ? 8'hff: 8'h00)
):vga_out_RGB[23:16];
        assign vga_out_green = (switch[0]) ? 8'h00: vga_out_RGB[15:8];


        //this section is where the alphanumeric display controls are implemented
        wire reset, up_clean, down_clean, right_clean;

        //debouncers for some of the user inputs
        debounce debreset(1'b0, clock_27mhz, ~button0, reset);
        debounce debup(1'b0, clock_27mhz, ~button_up, up_clean);
        debounce debdown(1'b0, clock_27mhz, ~button_down, down_clean);
        debounce debright(1'b0, clock_27mhz, ~button_right, right_clean);
        debounce debleft(1'b0, clock_27mhz, ~button_left, left_clean);
```

```
        wire [639:0] dots;
        wire [2:0] state;
        wire [23:0] roof, floor;
        wire [7:0] a,b,c,d,e,f,g,h,i,j;

        userdisp udp(reset, clock_27mhz, switch[3:1], state, roof, floor, a,b,c,d,e,f,g,h,i,j);

        alpha_display af(.clock(clock_27mhz), .ascii(1'b1), .bits(a), .dots(dots[639:600]));
        alpha_display bf(.clock(clock_27mhz), .ascii(1'b1), .bits(b), .dots(dots[599:560]));
        alpha_display cf(.clock(clock_27mhz), .ascii(1'b1), .bits(c), .dots(dots[559:520]));
        alpha_display df(.clock(clock_27mhz), .ascii(1'b1), .bits(d), .dots(dots[519:480]));
        alpha_display ef(.clock(clock_27mhz), .ascii(1'b1), .bits(e), .dots(dots[479:440]));
        alpha_display ff(.clock(clock_27mhz), .ascii(1'b1), .bits(f), .dots(dots[439:400]));
        alpha_display gf(.clock(clock_27mhz), .ascii(1'b1), .bits(g), .dots(dots[399:360]));
        alpha_display hf(.clock(clock_27mhz), .ascii(1'b1), .bits(h), .dots(dots[359:320]));
        alpha_display jf(.clock(clock_27mhz), .ascii(1'b1), .bits(8'h0), .dots(dots[319:280]));
        alpha_display kf(.clock(clock_27mhz), .ascii(1'b0), .bits(i), .dots(dots[279:240]));
        alpha_display lf(.clock(clock_27mhz), .ascii(1'b0), .bits(j), .dots(dots[239:200]));
        alpha_display mf(.clock(clock_27mhz), .ascii(1'b1), .bits(8'h0), .dots(dots[199:160]));
        alpha_display nf(.clock(clock_27mhz), .ascii(1'b1), .bits(8'h0), .dots(dots[159:120]));
        alpha_display of(.clock(clock_27mhz), .ascii(1'b1), .bits(8'h0), .dots(dots[119:80]));
        alpha_display pf(.clock(clock_27mhz), .ascii(1'b1), .bits(8'h0), .dots(dots[79:40]));
        alpha_display qf(.clock(clock_27mhz), .ascii(1'b1), .bits(8'h0), .dots(dots[39:0]));

        alphanumeric_displays adisp(clock_27mhz, reset, disp_test,
                            disp_blank, disp_clock, disp_rs, disp_ce_b,
                            disp_reset_b, disp_data_out, dots);


        wire up, down, right;

        leveltopulse upster(reset, pixel_clock, up_clean, up);
        leveltopulse downster(reset, pixel_clock, down_clean, down);
        leveltopulse rightster(reset, pixel_clock, right_clean, right);


        //here is where the display module which consists of the image processing is instatiated
        assign led[7:5] = state;

        wire [9:0] x;
        wire [8:0] y;
        wire s1,s2,s3;

        debounce sd(1'b0, clock_27mhz, switch[1], s1);
        debounce td(1'b0, clock_27mhz, switch[2], s2);
        debounce ud(1'b0, clock_27mhz, switch[3], s3);

        display test_camera(.reset(reset),
```

```
                                                    .clock_tv(tv_in_line_clock1),
                                                    .pixel_clock(pixel_clock),
                                                    .tv_in_ycrcb(tv_in_ycrcb),
                                                    .vga_out_blank_b(vga_out_blank_b),
                                                    .vga_out_sync_b(vga_out_sync_b),
                                                    .vga_out_hsync(vga_out_hsync),
                                                    .vga_out_vsync(vga_out_vsync),
                                                    .rgb_out(vga_out_RGB),

                                                    .rgbdata(rgbdata),
                                                    .pixel_count_cam(pixel_count_cam),
                                                    .line_count_cam(line_count_cam),
                                                    .im_en(button2),

                                                    .mode(mode),
                                                    .func({s3,s2,s1}),
                                                    .found(led[0]),
                                                    .right(right),
                                                    .digit(left_clean),
                                                    .up(up),
                                                    .down(down),
                                                    .state(state),
                                                    .roof(roof),
                                                    .floor(floor),
                                                    .choose(switch[5:4]),
                                                    .rend(switch[6]),
                                                    .x(x),
                                                    .y(y));


        //from here on is the connection to the PS/2 part
        debounce db3(1'b0, clock_27mhz, button3, b3);
        debounce db2(1'b0, clock_27mhz, button2, b2);

        wire ps2clkout, ps2dataout;
        assign user4[4] = ~ps2clkout;
        assign user4[2] = ~ps2dataout;
        wire update;

        wire [10:0] write_data, read_data;
        wire write_done, read_done, write_data_enable;

        wire write_enable, read_enable;
        wire [3:0] state_fsm;
        wire interrupt;
        wire [3:0] state_ctl;
        wire fpar;
        wire [7:0] first;
```

```
        wire [8:0] dx, dy;
        wire [8:0] x_old, y_old;
        wire state_md;

        //the mousedata module packages data from the image processing so that it can be sent to PS/2
        mousedata mdat(reset, pixel_clock, update, x[8:0], y, x_old, y_old, mouse_data_enable, dx, dy, state_md);

        //logic analyzer signals for debugging/testing
        assign analyzer2_data = {dx[8:0], mouse_data_enable, update, 5'h0};
        assign analyzer2_clock = pixel_clock;
        assign analyzer4_data = {dy[8:0], state_md, 6'h0};
        assign analyzer4_clock = pixel_clock;


        //This bit of code was used to fabricate a mouse_data_enable signal for testing the operation of the ps2 with the
        //labkit's buttons.
        /*
        reg mouse_data_enable = 1'b0;
        reg [20:0] count = 21'd0;
        always @ (posedge clock_27mhz)
        begin
                if (count == 21'd1350000) // 50 ms
                begin
                        mouse_data_enable <= ~(left_clean & right_clean & up_clean & down_clean & b3 & b2);
                        count <= 21'd0;
                end
                else
                begin
                        mouse_data_enable <= 1'b0;
                        count <= count + 1;
                end
        end
        */

        ps2system ps2(.power(user4[0]),
                                        .reset(~button0),
                                        .clock(pixel_clock),
                                        .mouse_data_enable(mouse_data_enable),        //these signals were used to test
the ps2 with labkit buttons
                                        .dx(dx),
//(left_clean) ? -4 : ((right_clean) ? 4 : 0))
                                        .dy(dy),                                                                        //
(down_clean) ? -4 : ((up_clean) ? 4 : 0))
                                        .left(~b3),
                                        .right(~b2),
                                        .ps2clkin(user4[3]),
                                        .ps2datain(user4[1]),
                                        .ps2clkout(ps2clkout),
                                        .ps2dataout(ps2dataout),
```

```
                                        .update(update),

                                        .write_data(write_data),
                                        .read_data(read_data),
                                        .write_done(write_done),
                                        .read_done(read_done),
                                        .write_data_enable(write_data_enable),

                                        .write_enable(write_enable),
                                        .state_fsm(state_fsm),
                                        .interrupt(interrupt),
                                        .read_enable(read_enable),
                                        .state(state_ctl),
                                        .fpar(fpar),
                                        .first(first));
endmodule

/************************************************/
// William Putnam's code begins here
/************************************************/


//the display module originally was written by Xing and its original purpose
//was to read video data out of the ram and use the ycrcb2rgb module to convert
//it to VGA and then output this VGA data along with the relevant VGA signals.
//This module turned out to be a relevant place for Billy to place instatiate
//his image processing and control modules, so now it serves as a workhorse module
//containing all of the image processing modules and most of the control of the entire
//system.

module display(reset, clock_tv, pixel_clock, tv_in_ycrcb, vga_out_blank_b, vga_out_sync_b,
                                        vga_out_hsync, vga_out_vsync, rgb_out,
                                        rgbdata, pixel_count_cam, line_count_cam, im_en, mode, func, found,
                                        right, digit, up, down, state, roof, floor, choose, rend, x, y);         //added

        //billy's additions
        input im_en; //overall image processing enable
        input [2:0] func;
        input right, digit, up, down;
        output [2:0] state;
        output [23:0] roof, floor;
        output [1:0] mode;
        output found; //this signal ended up not being used
        input [1:0] choose;
        input rend;
        output [9:0] x;
        output [8:0] y;
        //end of additions
```

```
        input reset;
        input clock_tv;
        input pixel_clock;
        input [19:0] tv_in_ycrcb;
        output vga_out_blank_b;
        output vga_out_sync_b;
        output vga_out_hsync;
        output vga_out_vsync;
        output [23:0] rgb_out;

        output [11:0] rgbdata;
        output [9:0] pixel_count_cam;
        output [9:0] line_count_cam;

        wire reset_sync;
        wire enable;
        wire [11:0] rgbdata, rgb;
        wire [9:0] pixel_count_cam, pixel_count;
        wire [9:0] line_count_cam, line_count;
        wire [16:0] addr_read, addr_write;
        wire hblank, vblank;

        //this was the original display module
        debounce reset_db(1'b0, clock_tv, reset, reset_sync);

        ycrcb2rgb y2r(reset_sync, clock_tv, tv_in_ycrcb, enable, rgbdata, pixel_count_cam, line_count_cam);
        address addrin(pixel_count_cam, line_count_cam, addr_write);  ///test
        //      testram test(pixel_clock, enable, rgbdata, addr_write);

        ram frame(.clka(clock_tv), .addra(addr_write), .dina(rgbdata), .wea(enable),
                                .clkb(pixel_clock), .addrb(addr_read), .doutb(rgb));

        //this is now enabled by the first frame
        vga_gen vga(reset_sync, pixel_clock, vga_out_hsync, vga_out_vsync, hblank, vblank, pixel_count, line_count);


        address addrout(pixel_count, line_count, addr_read);     /// test

        assign vga_out_blank_b = hblank & vblank;
        assign vga_out_sync_b = 1'b1;
        //end of original module (the assignement of the rgb data has been moved)


        wire out_en;
        wire pfound, pfound1; //these signals ended up not being used
        wire [9:0] x_cen1, x_cen2;
        wire [8:0] y_cen1, y_cen2;
        wire [18:0] top_l, tmpr, tmpl, botr_in, topl_in;
        wire corner_en;
```

```
        wire find_en;
        wire init_en;

        //here is the instantiation of the image processing modules
        otherfind ofinder(reset, pixel_clock, (find_en && (choose[0] || choose[1])), {rgb[11:8],4'b0,rgb[7:4],4'b0,rgb[3:0
],4'b0}, addr_read, bot_r, top_l, x_cen1, y_cen1, pfound1, floor, roof);
        pointerfind finder(reset, pixel_clock, (find_en && (!choose[0] || choose[1])), {rgb[11:8],4'b0,rgb[7:4],4'b0,rgb[3
:0],4'b0}, addr_read, bot_r, top_l, x_cen2, y_cen2, pfound, floor, roof); //first_frame);
        cornerfind corners(reset, pixel_clock, corner_en, {rgb[11:8],4'b0,rgb[7:4],4'b0,rgb[3:0],4'b0}, addr_read, bot_r,
top_l, found);
        ImageControl ic(reset, pixel_clock, im_en, func, mode, find_en, corner_en, init_en);
        initialize init(reset, pixel_clock, init_en, right, digit, up, down, state, floor, roof);

        wire [9:0] x_ren;
        wire [8:0] y_ren;
        wire [9:0] x_cen;
        wire [8:0] y_cen;
        wire [10:0] xint;
        wire [9:0] yint;
        wire [9:0] otherx;
        wire [8:0] othery;

        //here we average the results of the two pointer finding algorithms to provide
        //in a sense a third option for an algorithm which is a combination of the two
        assign xint = x_cen1+x_cen2;
        assign yint = y_cen1+y_cen2;
        assign otherx = xint[10:1];
        assign othery = yint[9:1];

        //here we select which pointing finding algorithm to use
        assign x_cen = choose[1] ? otherx :(choose[0] ? x_cen1: x_cen2);
        assign y_cen = choose[1] ? othery :(choose[0] ? y_cen1: y_cen2);

        render ren(reset, pixel_clock, top_l[18:9], top_l[8:0], bot_r[18:9], bot_r[8:0], found, x_cen, y_cen, x_ren, y_ren
);

        //select to use rendering or not (we would only not use rendering for debugging)
        assign x = rend ? x_ren: x_cen;
        assign y = rend ? y_ren: y_cen;

        //here the rgb data is assigned and a red square is drawn with its top left corner being
        //at the center of the laser pointer also two green squares are drawn at the top left
        //and bottom right corners of the screen
        assign rgb_out[23:0] = ((pixel_count >= x) && (pixel_count <= (x+8)) &&
                                                                (line_count >= y) && (line_count <= (y+8)))? 24'hf
f0000:

                                                                (( ((pixel_count >= bot_r[18:9]) && (pixel_count <
= (bot_r[18:9]+4)) &&
```

```
                                                    (line_count >= bot_r[8:0]) && (line_count <= (bot
_r[8:0]+4))) ||
                                                    ((pixel_count >= top_l[18:9]) && (pixel_count <=
(top_l[18:9]+4)) &&
                                                    (line_count >= top_l[8:0]) && (line_count <= (top
_l[8:0]+4))) )
                                                      ? 24'h00ff00: {rgb[11:8],4'b0,rgb[7:4],4'b0,rgb
[3:0],4'b0});
endmodule


//This is the corner finding algorithm.  This module will take in the video data
//and output the coordinates of the top left and bottom right corners of the screen

module cornerfind(reset, clk, enable, video, address, bot_r, top_l, found);

                input reset, clk, enable;
                input [23:0] video;
                input [16:0] address;

                output [18:0] bot_r;
                output [18:0] top_l;
                output found; //ended up not using

          parameter R_FLOOR = 8'h4f; //here are the color range paramters used for
                parameter R_ROOF = 8'hff;  //the finding the white of the screen
                parameter G_FLOOR = 8'h2c;
                parameter G_ROOF = 8'hff;
                parameter B_FLOOR = 8'h7f;
                parameter B_ROOF = 8'hff;

                parameter ADD_MAX = 17'b10011111111101111; //{640/2-1,480/2-1} converted to binary
                parameter BOUND = 0;
                parameter LIM = 7'd32;

                reg [18:0] bot_r;
                reg [18:0] top_l;
                reg [18:0] botr_old;
                reg [18:0] topl_old;
                reg [18:0] tmp_botr;
                reg [18:0] tmp_topl;
                reg [6:0] rcount;
                reg [6:0] lcount;
                reg found;

                //It is important to note that corners do not actually represent a single pixel with those
                //coordinates; they instead represent the greatest or least x coordinate among all the  white
                //pixels matched with the greatest or least y coordinate of all the white pixels
```

```
                //we begin by setting the corners to be the largest and smallest values they can be
                always @ (posedge clk or posedge reset)
                        if(reset) begin bot_r <= 0; top_l <= ADD_MAX; botr_old <= 0; topl_old <= ADD_MAX; found <= 0;
                                                                        tmp_botr <= 0; tmp_topl <= ADD_MAX; found <= 0; rc
ount <= 0; lcount <= 0; end
                        else begin
                                if(enable) begin
                                        //if the bottom right and top left corner have been stable for a certain period of
 time
                                        //specified by LIM then keep them at those spots indefinitely
                                        if((rcount == LIM) && (lcount == LIM)) begin bot_r <= bot_r; top_l <= top_l;
                                                tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; found <= 1; rcount <= LIM; lco
unt <= LIM; end


                                        //when we reach a full frame take the registered values of the corners and pass th
em to the actual
                                        //corner output also if a corner is in the same spot (same spot specified by BOUND
) then increment
                                        //a count
                                        else if(address[16:0] == ADD_MAX) begin
                                                        if((tmp_botr[18:9] >= (bot_r[18:9]-BOUND)) && (tmp_botr[18:9] <= (
bot_r[18:9]+BOUND)) &&
                                                                (tmp_botr[8:0] >= (bot_r[8:0]-BOUND)) && (tmp_botr[8:0] <=
 (bot_r[8:0]+BOUND))) begin
                                                                rcount <= (rcount+1); bot_r <= bot_r; top_l <= top
_l; found <= 0;
                                                                tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; end
                                                        if((tmp_topl[18:9] >= (top_l[18:9]-BOUND)) && (tmp_topl[18:9] <= (
top_l[18:9]+BOUND)) &&
                                                                (tmp_topl[8:0] >= (top_l[8:0]-BOUND)) && (tmp_topl[8:0] <=
 (top_l[8:0]+BOUND))) begin
                                                                lcount <= (lcount+1); bot_r <= bot_r; top_l <= top
_l; found <= 0;
                                                                tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; end


                                                        else begin
                                                                bot_r <= tmp_botr; top_l <= tmp_topl; botr_old <= bot_r; t
opl_old <= top_l;
                                                                tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; found <= 0; rc
ount <= 0; lcount <= 0; end
                                                        end


                                        //if a pixel falls in the appropriate color range and is on the screen then check
to see if its
                                        //coordinates are greater or less than those of the current corners if they are th
en update
                                        //the corner values accordingly
                                        else if((address[16:8] < ADD_MAX[16:8]) && (address[7:0] < ADD_MAX[7:0]) &&
                                                        (video[23:16] >= R_FLOOR) && (video[23:16] <= R_ROOF) &&
```

```
                                                                (video[15:8] >= G_FLOOR) && (video[15:8] <= G_ROOF) &&
                                                                (video[7:0] >= B_FLOOR) && (video[7:0] <= B_ROOF)) begin
                                                                if(({address[16:8],1'b1} > bot_r[18:9])) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_bo
tr <= {address[16:8],1'b1,tmp_botr[8:0]};
                                                                        tmp_topl <= tmp_topl; rcount <= rcount; lcount <=
lcount; end
                                                                if(({address[7:0],1'b1} > bot_r[8:0])) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_bo
tr <= {tmp_botr[18:9],address[7:0],1'b1};
                                                                        tmp_topl <= tmp_topl; rcount <= rcount; lcount <=
lcount; end
                                                                if(({address[16:8],1'b1} < top_l[18:9])) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_to
pl <= {address[16:8],1'b1,tmp_topl[8:0]};
                                                                        tmp_botr <= tmp_botr; rcount <= rcount; lcount <=
lcount; end
                                                                if(({address[7:0],1'b1} < top_l[8:0])) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_to
pl <= {tmp_topl[18:9],address[7:0],1'b1};
                                                                        tmp_botr <= tmp_botr; rcount <= rcount; lcount <=
lcount; end
                                                end
                                        else
                                                begin bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_botr <= tmp_botr;
                                                                tmp_topl <= tmp_topl; rcount <= rcount; lcount <= lcount;
end
                                        end
                                else begin bot_r <= 0; top_l <= ADD_MAX; found <= 0; tmp_botr <= tmp_botr;
                                                                tmp_topl <= tmp_topl; rcount <= 0; lcount <= 0; end
                        end


endmodule


///Here is a large portion of code that I was using earlier for the corner finding.  It implemented slightly different
///ideas than the above code and did not perform as well.
/*
if({address[16:8],1'b1} > bot_r[18:9]) begin
                                                                        //if((count == 0) || ({address[16:8],1'b1} < (bot_
r[18:9]+BOUND))) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_bo
tr <= {address[16:8],1'b1,tmp_botr[8:0]}; tmp_topl <= tmp_topl; end
                                                                        //else begin bot_r <= bot_r; top_l <= top_l; found
 <= 0; tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; end end
                                                                if({address[7:0],1'b1} > bot_r[8:0]) begin
                                                                        //if((count == 0) || ({address[7:0],1'b1} < (bot_r
[8:0]+BOUND))) begin
```

```
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_bo
tr <= {tmp_botr[18:9],address[7:0],1'b1}; tmp_topl <= tmp_topl; end
                                                                        //else begin bot_r <= bot_r; top_l <= top_l; found
 <= 0; tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; end end
                                                                    if({address[16:8],1'b1} < top_l[18:9]) begin
                                                                        //if((count == 0) || ({address[16:8],1'b1} > (top_
l[18:9]-BOUND))) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_to
pl <= {address[16:8],1'b1,tmp_topl[8:0]}; tmp_botr <= tmp_botr; end
                                                                        //else begin bot_r <= bot_r; top_l <= top_l; found
 <= 0; tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; end end
                                                                    if({address[7:0],1'b1} < top_l[8:0]) begin
                                                                        //if((count == 0) || ({address[7:0],1'b1} > (top_l
[8:0]-BOUND))) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_to
pl <= {tmp_topl[18:9],address[7:0],1'b1}; tmp_botr <= tmp_botr; end
                                                                */

/*      if(({address[16:8],1'b1} > bot_r[18:9]) && ({address[16:8],1'b1} < (bot_r[18:9]+BOUND))) begin
                                                                        //if((count == 0) || ({address[16:8],1'b1} < (bot_
r[18:9]+BOUND))) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_bo
tr <= {address[16:8],1'b1,tmp_botr[8:0]}; tmp_topl <= tmp_topl; count <= count+1; end
                                                                        //else begin bot_r <= bot_r; top_l <= top_l; found
 <= 0; tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; end end
                                                                    if(({address[7:0],1'b1} > bot_r[8:0]) && ({address[7:0],1'
b1} < (bot_r[8:0]+BOUND))) begin
                                                                        //if((count == 0) || ({address[7:0],1'b1} < (bot_r
[8:0]+BOUND))) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_bo
tr <= {tmp_botr[18:9],address[7:0],1'b1}; tmp_topl <= tmp_topl; count <= count+1; end
                                                                        //else begin bot_r <= bot_r; top_l <= top_l; found
 <= 0; tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; end end
                                                                    if(({address[16:8],1'b1} < top_l[18:9]) && ({address[16:8]
,1'b1} > (top_l[18:9]-BOUND))) begin
                                                                        //if((count == 0) || ({address[16:8],1'b1} > (top_
l[18:9]-BOUND))) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_to
pl <= {address[16:8],1'b1,tmp_topl[8:0]}; tmp_botr <= tmp_botr; count <= count+1; end
                                                                        //else begin bot_r <= bot_r; top_l <= top_l; found
 <= 0; tmp_botr <= tmp_botr; tmp_topl <= tmp_topl; end end
                                                                    if(({address[7:0],1'b1} < top_l[8:0]) && ({address[16:8],1
'b1} > (top_l[18:9]-BOUND))) begin
                                                                        //if((count == 0) || ({address[7:0],1'b1} > (top_l
[8:0]-BOUND))) begin
                                                                        bot_r <= bot_r; top_l <= top_l; found <= 0; tmp_to
pl <= {tmp_topl[18:9],address[7:0],1'b1}; tmp_botr <= tmp_botr; count <= count+1; end
                                                                */
```

```
///ImageControl is a very simple FSM.  It has only three states that pass the
///appropriate enable signals to the various image processing modules.

module ImageControl(reset, clk, enable, func, mode, find_en, corner_en, init_en);

                input reset, clk, enable;
                input [1:0] func;

                output [1:0] mode; //0 idle; 1 find; 2 adjust;
                output find_en, corner_en, init_en;

                parameter idle = 0; //in idle corner finding is enabled and nothing else
                parameter find = 1; //in find pointer finding is enabled and corner finding is enabled (the corner
                                                                //have probably already been found however)
                parameter init = 2; //in init the user is given the ability to adjust various image processing
                                                                //parameters, the initialization FSM is enabled

                reg [1:0] mode;
                reg [1:0] state, next;
                reg find_en;
                reg corner_en;
                reg init_en;

                always @ (posedge clk or posedge reset)
                if(reset) state <= idle;
                else state <= next;

                        always @ ( * )
                case(state)

                                                idle: if(!enable) begin next <= idle; mode <= 0; find_en <= 0; corner_en <
= 1; init_en <= 0; end

                                                        else if(func == 1) begin next <= find; mode <= 0; find_en
<= 0; corner_en <= 1; init_en <= 0; end

                                                        else if(func == 2) begin next <= init; mode <= 0; find_en
<= 0; corner_en <= 1; init_en <= 0; end

                                                        else begin next <= idle; mode <= 0; find_en <= 0; corner_e
n <= 1; init_en <= 0; end


                                                find: if(!enable) begin next <= idle; mode <= 1; find_en <= 0; corner_en <
= 0; init_en <= 0; end

                                                        else if(func == 1) begin next <= find; mode <= 1; find_en
<= 1; corner_en <= 1; init_en <= 0; end

                                                        else if(func == 2) begin next <= init; mode <= 1; find_en
<= 0; corner_en <= 1; init_en <= 0; end

                                                        else begin next <= idle; mode <= 1; find_en <= 0; corner_e
```

```
n <= 1; init_en <= 0; end

                                                    init: if(!enable) begin next <= idle; mode <= 2; find_en <= 0; corner_en <
= 0; init_en <= 0; end
                                                                    else if(func == 1) begin next <= find; mode <= 2; find_en
<= 0; corner_en <= 1; init_en <= 0; end
                                                                    else if(func == 2) begin next <= init; mode <= 2; find_en
<= 0; corner_en <= 1; init_en <= 1; end //initialization
                                                                    else begin next <= idle; mode <= 2; find_en <= 0; corner_e
n <= 1; init_en <= 0; end
                                        endcase

endmodule


///The initialize FSM allows the user to change various image processing parameters
///by interacting with the labkit.  The user can increment parameters with up, decrement
///them with down, and change parameters with right.  Left (digit) allows the user to
///switch which digit of the two digit hex parameter he or she wishes to increment or
///decrement

module initialize(reset, clk, enable, right, digit, up, down, state, floor, roof);

                input reset, clk, enable;
                input right, digit, up, down;

                output [2:0] state;
                output [23:0] floor;
                output [23:0] roof;

                parameter idle = 0;
                parameter redfloor = 1;
                parameter redroof = 2;
                parameter greenfloor = 3;
                parameter greenroof = 4;
                parameter bluefloor = 5;
                parameter blueroof = 6;

                reg [2:0] state, next;
                reg [23:0] floor; //= 24'hbf4f4f; these are the starting paramters for roof and floor
                reg [23:0] roof; //= 24'hff8f8f;
                reg [23:0] f_reg;
                reg [23:0] r_reg;

                always @ (posedge clk or posedge reset)
                if(reset) state <= idle;
                                        else begin state <= next; floor <= f_reg; roof <= r_reg; end

                        ///each state will check if the user is incrementing or decrementing and if the user is trying
```

```
                    ///to switch parameters

                    always @ ( * )
              case(state)
                                          idle:                   if(!enable)begin next <= idle; f_reg <= 24'hbf4f4f
; r_reg <= 24'he88f8f; end
                                                                          else if(right) begin next <= redfloor; f_r
eg <= 24'hbf4f4f; r_reg <= 24'hff8f8f; end
                                                                          else begin next <= idle; f_reg <= 24'hbf4f
4f; r_reg <= 24'hff8f8f; end

                                          redfloor:       if(!enable) begin next <= redfloor; f_reg <= floor; r_reg
<= roof; end
                                                                          else if(right) begin next <= redroof; f_re
g <= floor; r_reg <= roof; end
                                                                          else if(up) begin       next <= redfloor;
                                                                                  if(!digit) begin f_reg <= (floor+2
4'h010000); r_reg <= roof; end
                                                                                  else begin f_reg <= (floor+24'h100
000); r_reg <= roof; end end
                                                                          else if(down) begin next <= redfloor;
                                                                                  if(!digit) begin f_reg <= (floor-2
4'h010000); r_reg <= roof; end
                                                                                  else begin f_reg <= (floor-24'h100
000); r_reg <= roof; end end
                                                                          else begin next <= redfloor; f_reg <= floo
r; r_reg <= roof; end

                                        redroof:          if(!enable) begin next <= redfloor; f_reg <= floor; r_reg
<= roof; end
                                                                          else if(right) begin next <= greenfloor; f
_reg <= floor; r_reg <= roof; end
                                                                          else if(up) begin next <= redroof;
                                                                                  if(!digit) begin r_reg <= (roof+24
'h010000); f_reg <= floor; end
                                                                                  else begin r_reg <= (roof+24'h1000
00); f_reg <= floor; end end
                                                                          else if(down) begin next <= redroof;
                                                                                  if(!digit) begin r_reg <= (roof-24
'h010000); f_reg <= floor; end
                                                                                  else begin r_reg <= (roof-24'h1000
00); f_reg <= floor; end end
                                                                          else begin next <= redroof; f_reg <= floor
; r_reg <= roof; end

                                          greenfloor:     if(!enable) begin next <= redfloor; f_reg <= floor; r_reg
<= roof; end
                                                                          else if(right) begin next <= greenroof; f_
reg <= floor; r_reg <= roof; end
```

```
                                                    else if(up) begin next <= greenfloor;
                                                           if(!digit) begin r_reg <= roof; f_
reg <= (floor+24'h000100); end
                                                           else begin r_reg <= roof; f_reg <=
 (floor+24'h001000); end end
                                                    else if(down) begin next <= greenfloor;
                                                           if(!digit) begin r_reg <= roof; f_
reg <= (floor-24'h000100); end
                                                           else begin r_reg <= roof; f_reg <=
 (floor-24'h001000); end end
                                                    else begin next <= greenfloor; f_reg <= fl
oor; r_reg <= roof; end

                                    greenroof:      if(!enable) begin next <= redfloor; f_reg <= floor; r_reg
<= roof; end
                                                    else if(right) begin next <= bluefloor; f_
reg <= floor; r_reg <= roof; end
                                                    else if(up) begin next <= greenroof;
                                                           if(!digit) begin r_reg <= (roof+24
'h000100); f_reg <= floor; end
                                                           else begin r_reg <= (roof+24'h0010
00); f_reg <= floor; end end
                                                    else if(down) begin next <= greenroof;
                                                           if(!digit) begin r_reg <= (roof-24
'h000100); f_reg <= floor; end
                                                           else begin r_reg <= (roof-24'h0010
00); f_reg <= floor; end end
                                                    else begin next <= greenroof; f_reg <= flo
or; r_reg <= roof; end

                                    bluefloor:      if(!enable) begin next <= redfloor; f_reg <= floor; r_reg
<= roof; end
                                                    else if(right) begin next <= blueroof; f_r
eg <= floor; r_reg <= roof; end
                                                    else if(up) begin next <= bluefloor;
                                                           if(!digit) begin r_reg <= roof; f_
reg <= (floor+24'h000001); end
                                                           else begin r_reg <= roof; f_reg <=
 (floor+24'h000010); end end
                                                    else if(down) begin next = bluefloor;
                                                           if(!digit) begin r_reg <= roof; f_
reg <= (floor-24'h000001); end
                                                           else begin r_reg <= roof; f_reg <,
= (floor-24'h000010); end end
                                                    else begin next <= bluefloor; f_reg <= flo
or; r_reg = roof; end

                                    blueroof:       if(!enable) begin next <= redfloor; f_reg <= floor; r_reg
<= roof; end
```

```
                                                                    else if(right) begin next <= redfloor; f_r
eg <= floor; r_reg <= roof; end
                                                                    else if(up) begin next <= blueroof;
                                                                            if(!digit) begin r_reg <= (roof+24
'h000001); f_reg <= floor; end
                                                                            else begin r_reg <= (roof+24'h0000
10); f_reg <= floor; end end
                                                                    else if(down) begin next <= blueroof;
                                                                            if(!digit) begin r_reg <= (roof-24
'h000001); f_reg <= floor; end
                                                                            else begin r_reg <= (roof-24'h0000
10); f_reg <= floor; end end
                                                                    else begin next <= blueroof; f_reg <= floo
r; r_reg <= roof; end
                                                endcase

endmodule


///This module is one of the two pointer finding algorithms.  This algorithm will look
///for four pixels in a small color range and then average there coordinates.

module otherfind(reset, clk, enable, video, address, bot_r, top_l, x_cen, y_cen, found, floor, roof);

                input reset, clk, enable;
                input [23:0] video;
                input [16:0] address;
                input [18:0] bot_r;
                input [18:0] top_l;
                input [23:0] floor;
                input [23:0] roof;

                output found; //ended up not being used
                output [9:0] x_cen; //x is {x_ad, 0}
                output [8:0] y_cen;

                parameter BOX = 16;
                parameter PIX = 4; //grab four pixels and average note the bit shift
                                                                //average depends on this value so it should be a
                                                                //power of two

                reg found;
                reg [2:0] count; //depends on pix
                reg [10:0] xad_sum;
                reg [9:0] yad_sum;
                reg [9:0] x_cen;
                reg [8:0] y_cen;

                reg [8:0] xad_old;
```

```
                reg [7:0] yad_old;

                reg [8:0] xcen_old;

                reg [7:0] ycen_old;

                always @ (posedge clk or posedge reset)
                        if(reset) begin x_cen <= 0; y_cen <= 0; xad_sum <= 0; yad_sum <= 0; found <= 0;
                                                                count <= 0; xcen_old <= 0; ycen_old <= 0; xad_old
<= 0; yad_old <= 0; end
                        else begin
                                if(enable) begin
                                        ///if we have reached the desired number of pixels average the sums and pass it ou
t as
                                        ///the center coordinate
                                        if(count == PIX) begin
                                                x_cen <= {xad_sum[10:2],2'b0}; y_cen <= {yad_sum[9:1],2'b0}; xad_sum <= 0;
 yad_sum <= 0;
                                                found <= 1; count <= 0; xcen_old <= x_cen; ycen_old <= y_cen; xad_old <= 0
; yad_old <= 0;
                                                end
                                        ///if the cuurent pixel is in the appropriate color range and it is on the screen
                                        ///that we are projecting onto then update the running sums
                                        else if((video[23:16] >= floor[23:16]) && (video[23:16] <= roof[23:16]) &&
                                                (video[15:8] >= floor[15:8]) && (video[15:8] <= roof[15:8]) &&
                                                (video[7:0] >= floor[7:0]) && (video[7:0] <= roof[7:0]) &&
                                                ({address[16:8],1'b1} >= top_l[18:9]) && ({address[7:0],1'b1} >= top_l[8:0
]) &&
                                                ({address[16:8],1'b1} <= bot_r[18:9]) && ({address[7:0],1'b1} <= bot_r[8:0
]))
                                                begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= xad_sum+address[16:8]; ya
d_sum <= yad_sum+address[7:0];
                                                        found <= 0; count <= count+1; xcen_old <= xcen_old; ycen_o
ld <= ycen_old;
                                                        xad_old <= address[16:8]; yad_old <= address[7:0]; end
                                        else
                                                begin x_cen <= x_cen; y_cen <= y_cen; xcen_old <= xcen_old; ycen_old <= yc
en_old;
                                                        xad_sum <= xad_sum; yad_sum <= yad_sum; found <= 0; count
<= count;
                                                        xad_old <= xad_old; yad_old <= yad_old; end
                                        end
                                else begin x_cen <= 0; y_cen <= 0; xcen_old <= 0; ycen_old <= 0;
                                                        xad_sum <= 0; yad_sum <= 0; found <= 0; count <= 0;
                                                        xad_old <= 0; yad_old <= 0; end
                        end

endmodule
```

```
///This is one of the pointer finding algorithms.  It basically looks for two red pixels that are very
///close to each other.  It then averages their coordinates and calls this the center of the laser pointer.

module pointerfind(reset, clk, enable, video, address, bot_r, top_l, x_cen, y_cen, found, floor, roof);

                input reset, clk, enable;
                input [23:0] video;
                input [16:0] address;
                input [18:0] bot_r;
                input [18:0] top_l;
                input [23:0] floor;
                input [23:0] roof;

                output found; //ended up not being used
                output [9:0] x_cen; //x is {x_ad, 0}
                output [8:0] y_cen;

                parameter BOX = 2; //how close the two pixels must be to each other
                parameter STILL = 3; //if the center of the pointer is this close to where it was on the previous
                                                                        //assignment then do not move it--this limits lase
r pointer jumpiness
                parameter PIX = 11'd350; //count through this many pixels looking for a red pixel in close
                                                                                //proximity and then restart the process
                parameter VEL = 16;

                reg found;
                reg [2:0] count; //depends on pix
                reg [10:0] inc;
                reg [10:0] xad_sum;
                reg [9:0] yad_sum;
                reg [9:0] x_cen, x_old;
                reg [8:0] y_cen, y_old;


                always @ (posedge clk or posedge reset)
                        if(reset) begin x_cen <= 0; y_cen <= 0; xad_sum <= 0; yad_sum <= 0; inc <= 0;
                                                                        count <= 0; x_old <= 0; y_old <= 0; found <= 0; en
d
                        else begin
                                if(enable) begin
                                //if we find a second red pixel that is close then check to see if the average coordinate
                                //is within a certain range of the previous average--if it is do not move the pointer loca
tion
                                //this limits jumpiness
                                        if(count == 2) begin
                                                if(({xad_sum[9:1],1'b1} <= (x_cen+STILL)) && ({xad_sum[9:1],1'b1} >= (x_ce
n-STILL)) &&
                                                        ({yad_sum[8:1],1'b1} <= (y_cen+STILL)) && ({yad_sum[8:1],1'b1} >=
```

```
(y_cen-STILL)))
                                                        begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= 0; yad_sum <= 0;
                                                                count <= 0; inc <= 0; x_old <= x_cen; y_old <= y_c
en; found <= 1; end
                                                //if it is not that close then simply update the pointer location
                                                else begin x_cen <= {xad_sum[9:1],1'b1}; y_cen <= {yad_sum[8:1],1'b1}; xad
_sum <= 0; yad_sum <= 0;
                                                                count <= 0; inc <= 0; x_old <= x_cen; y_old <= y
_cen; found <= 1; end
                                                end
                                        //If we have one red pixel look for another one that is in on the projected screen and clo
se
                                        //to our original pixel.  For every pixel that is not red increment a count.  When the cou
nt
                                        //reaches a certain value we discard the original pixel as noise and start the process ove
r
                                        else if(count == 1) begin
                                                if((video[23:16] >= floor[23:16]) && (video[23:16] <= roof[23:16]) &&
                                                        (video[15:8] >= floor[15:8]) && (video[15:8] <= roof[15:8]) &&
                                                        (video[7:0] >= floor[7:0]) && (video[7:0] <= roof[7:0]) &&
                                                        ({address[16:8],1'b1} >= top_l[18:9]) && ({address[7:0],1'b1} >= t
op_l[8:0]) &&
                                                        ({address[16:8],1'b1} <= bot_r[18:9]) && ({address[7:0],1'b1} <= b
ot_r[8:0]) &&
                                                        (address[16:8] <= (xad_sum[10:0]+BOX)) && (address[16:8] >= (xad_s
um[10:0]-BOX)) &&
                                                        (address[7:0] <= (yad_sum[9:0]+BOX)) && (address[7:0] >= (yad_sum[
9:0]-BOX)) && (inc <= PIX))
                                                        begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= (xad_sum+address[
16:8]); yad_sum <= (yad_sum+address[7:0]);
                                                                count <= 2; inc <= 0; found <= 0; end
                                                else if(inc > PIX) begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= 0;
                                                                yad_sum <= 0; count <= 0; inc <= 0; found <= 0;
end
                                                else begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= xad_sum;
                                                                yad_sum <= yad_sum; count <= 1; inc <= inc+1; fo
und <= 0; end
                                                end
                                         //here we have no red pixels yet so we simply look for a red pixel that is on the screen
                                        else if(count == 0) begin
                                                if((video[23:16] >= floor[23:16]) && (video[23:16] <= roof[23:16]) &&
                                                        (video[15:8] >= floor[15:8]) && (video[15:8] <= roof[15:8]) &&
                                                        (video[7:0] >= floor[7:0]) && (video[7:0] <= roof[7:0]) &&
                                                        ({address[16:8],1'b1} >= top_l[18:9]) && ({address[7:0],1'b1} >= t
op_l[8:0]) &&
                                                        ({address[16:8],1'b1} <= bot_r[18:9]) && ({address[7:0],1'b1} <= b
ot_r[8:0]))
                                                        begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= address[16:8]; ya
d_sum <= address[7:0];
```

```
                                                        count <= 1; inc <= 0; found <= 0; end
                                                else begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= 0;
                                                                yad_sum <= 0; count <= 0; inc <= 0; found <= 0;
end
                                                end
                                        else
                                                begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= 0;
                                                        yad_sum <= 0; count <= 0; inc <= 0; found <= 0; end
                                        end
                                else
                                        begin x_cen <= x_cen; y_cen <= y_cen; xad_sum <= 0;
                                                yad_sum <= 0; count <= 0; inc <= 0; found <= 0; end
                        end

endmodule


///This module takes absolute pixel coordinates from the image processing and converts
///them into a change in x change in y format that the PS/2 interface can use

module mousedata(reset, clk, update, x, y, x_old, y_old, data_en, dx, dy, state);
//the x_old, y_old, and state signals were used for debugging

                input reset, clk, update;
                input [8:0] x;
                input [8:0] y;

                output data_en;
                output [8:0] dx, dy;

                //debugging
                output [8:0] x_old, y_old;
                output state;

                reg [8:0] dx, dy;
                reg data_en;
                reg [8:0] x_old;
                reg [8:0] y_old;
                reg state;

                wire pulse;

                //we need this module to find the negedge of any reset because the PS/2 must have
                //zeros written to it in order for it to begin functioning properly (the PS/2 requests
                //updates and it will not request updates unless it first has something written to it
                //so on reset we write zeros to it and it then goes into it's normal routine of requesting
                //updates)
                leveltopulse levp(1'b0, clk, !reset, pulse);
```

```
                always @ (posedge clk or posedge reset)
                        if(reset) begin dx <= 0; dy <= 0; data_en <= 0; x_old <= 0; y_old <= 0; state <= 1'b0; end
                        else if(pulse) begin dx <= 0; dy <= 0; data_en <= 1; x_old <= x_old; y_old <= y_old; state <= 1'b0
; end
                        else if(update) begin dx <= (x-x_old); dy <= (y_old-y); data_en <= 1; x_old <= x; y_old <= y; stat
e <= 1'b1; end
                        else begin dx <= dx; dy <= dy; data_en <= 0; x_old <= x_old; y_old <= y_old; state <= 1'b0; end

endmodule


///this module takes the very long pulses given by someone pressing a button on the labkit and
///converts them to single clock cycle pulses

module leveltopulse(reset, clk, level, pulse);

                input reset, clk, level;
                output pulse;

                reg pulse;
                reg oldlev;

                always @ (posedge clk or posedge reset) //negedge
                        if(reset) begin pulse <= 0; oldlev <= 0; end
                        else if(level && !oldlev) begin pulse <= 1; oldlev <= level; end
                        else begin pulse <= 0; oldlev <= level; end

endmodule


/************************************************/
// Xinpeng Huang's code begins here
/************************************************/

// ycrcb2rgb takes in ycrcb data from the ADV7185, decodes the signals to figure out
// pixel and line coordinates, and converts color data into RGB to be stored into block ram.
module ycrcb2rgb(reset, clock, ycrcb_in, enable_out, rgbdata_out, pixel_count_cam, line_count_cam);
        input reset;
        input clock;
        input [19:0] ycrcb_in;
        output enable_out; // when new rgbdata_out is valid
        output [11:0] rgbdata_out; // 4 for r, 4 for g, 4 for b
        output [9:0] pixel_count_cam;
        output [9:0] line_count_cam;

        // states
        parameter BLANK = 0;
        parameter SAV1 = 1;
        parameter SAV2 = 2;
```

```
        parameter SAV3 = 3;
        parameter Y = 4;
        parameter CR = 5;
        parameter CB = 6;
        parameter EAV1 = 7;
        parameter EAV2 = 8;
        parameter EAV3 = 9;
        parameter RESET_LINE = 10;

        wire [9:0] ycrcb;
        assign ycrcb = ycrcb_in[19:10];
        wire [7:0] r, g, b;

        reg [3:0] state = BLANK;
        reg [3:0] next;
        reg [9:0] pixel_count_cam = 10'd0;
        reg [9:0] line_count_cam = 10'd0; // care only about the odd lines
        reg next_cr = 1'b0; // determine if the next chrominance value is red or blue
        reg [9:0] y1, y2, cr, cb;
        reg enable_out = 1'b0;

        always @ (posedge clock)
        begin
                if (reset)
                begin
                        state <= BLANK;
                        pixel_count_cam <= 10'd0;
                        line_count_cam <= 10'd0;
                        enable_out <= 1'b0;
                end
                else
                        state <= next;

                if (next == Y)
                begin
                        enable_out <= 1'b0;
                        if (state == SAV3)
                        begin
                                pixel_count_cam <= 10'h3FF;
                                next_cr <= 1'b0;
                                y1 <= ycrcb;
                                y2 <= 10'h0;
                                cr <= 10'h0;
                                cb <= 10'h0;
                                if (line_count_cam == 10'd0)
                                        line_count_cam <= 10'h3EB; // -21
                                else
                                        line_count_cam <= line_count_cam + 2;
                        end
```

```
                        else
                        begin
                                next_cr <= ~next_cr;
                                y2 <= ycrcb;
                        end
                end
                else if (next == CR)
                begin
                        cr <= ycrcb;
                                y1 <= y2;
                                pixel_count_cam <= pixel_count_cam + 1;
                        enable_out <= 1'b1;
                end
                else if (next == CB)
                begin
                        cb <= ycrcb;
                        if (y2 != 10'h0)
                        begin
                                y1 <= y2;
                                pixel_count_cam <= pixel_count_cam + 1;
                        end
                        enable_out <= 1'b1;
                end
                else if (next == RESET_LINE)
                        line_count_cam <= 10'd0;

        end

        always @ (ycrcb or state or next or next_cr)
        begin
                case (state)
                        BLANK: next = (ycrcb == 10'h3FF) ? SAV1 : BLANK;
                        SAV1: next = (ycrcb == 10'h0) ? SAV2 : ((ycrcb == 10'h3FF) ? SAV1 : BLANK);
                        SAV2: next = (ycrcb == 10'h0) ? SAV3 : ((ycrcb == 10'h3FF) ? SAV1 : BLANK);
                        SAV3:   next = (ycrcb == 10'h200) ? Y : ((ycrcb == 10'h2D8) ? RESET_LINE : BLANK);
                        Y: next = (ycrcb == 10'h3FF) ? EAV1 : (next_cr ? CR : CB);
                        CR: next = (ycrcb == 10'h3FF) ? EAV1 : Y;
                        CB: next = (ycrcb == 10'h3FF) ? EAV1 : Y;
                        EAV1: next = (ycrcb == 10'h0) ? EAV2 : ((ycrcb == 10'h3FF) ? EAV1 : Y);
                        EAV2: next = (ycrcb == 10'h0) ? EAV3 : ((ycrcb == 10'h3FF) ? EAV1 : Y);
                        EAV3: next = BLANK;
                        RESET_LINE: next = BLANK;
                        default: next = state;
                endcase
        end
/*
        // RGB color values
        assign r = y1 - 10'd64 + 3 * (cr - 10'd512) / 2;
        assign g = y1 - 10'd64 - 3 * (cr - 10'd512) / 4 - 3 * (cb - 512) / 8;
```

```
        assign b = y1 - 10'd64 + 2 * (cb - 512);

        assign rgbdata_out = {r[9:6], g[9:6], b[9:6]};
*/
        // YCrCb to RGB conversion, from Xilinx
        wire [20:0] R_int, G_int, B_int, X_int;
        reg [9:0] const1 = 10'b0100101010; //1.164 = 01.00101010
        reg [9:0] const2 = 10'b0110011000; //1.596 = 01.10011000
        reg [9:0] const3 = 10'b0011010000; //0.813 = 00.11010000
        reg [9:0] const4 = 10'b0001100100; //0.392 = 00.01100100
        reg [9:0] const5 = 10'b1000000100; //2.017 = 10.00000100

        assign X_int = (const1 * (y1 - 'd64)) ;
   assign R_int = X_int + (const2 * (cr - 'd512));
   assign G_int = X_int - (const3 * (cr - 'd512)) - (const4 * (cb - 'd512));
   assign B_int = X_int + (const5 * (cb - 'd512));

        // limit output to 0 - 4095, <0 equals o and >4095 equals 4095
        assign r = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
        assign g = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
        assign b = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;

        assign rgbdata_out = {r[7:4], g[7:4], b[7:4]};

endmodule

// render takes in upper left and lower right projection coordinates,
// the current location of the laser pointer, and maps that into pixel coordinates
// of the actual computer display.

module render(reset, clock, ulxin, ulyin, lrxin, lryin, en_corner, xin, yin, xout, yout);
        input reset;
        input clock;
        input [9:0] ulxin;
        input [8:0] ulyin;
        input [9:0] lrxin;
        input [8:0] lryin;
        input en_corner;
        input [9:0] xin;
        input [8:0] yin;
        output [9:0] xout;
        output [8:0] yout;

        parameter XMAX = 640; // VGA resolution
        parameter YMAX = 480;

        reg [9:0] ulx, lrx, xout;
        reg [8:0] uly, lry, yout;
        wire [19:0] xdividend, ydividend;
```

```
        wire [9:0] xdivisor, ydivisor;
        wire [19:0] xout_tmp, yout_tmp;
        wire [9:0] xremd, yremd;
        wire xrfd, yrfd;

        always @ (posedge clock)
        begin
                if (reset)
                begin
                        ulx <= 10'd0;
                        uly <= 9'd0;
                        lrx <= XMAX - 1;
                        lry <= YMAX - 1;
                end
                else if (en_corner)
                begin
                        ulx <= ulxin;
                        uly <= ulyin;
                        lrx <= lrxin;
                        lry <= lryin;
                end
                xout <= xout_tmp;
                yout <= yout_tmp;
        end

        // linear transformation: divide
        assign xdividend = (XMAX - 1) * (((xin < ulx) ? ulx : ((xin > lrx) ? lrx : xin)) - ulx);
        assign ydividend = (YMAX - 1) * (((yin < uly) ? uly : ((yin > lry) ? lry : yin)) - uly);
        assign xdivisor = lrx - ulx;
        assign ydivisor = lry - uly;

        divider xdiv(.clk(clock), .dividend(xdividend), .divisor(xdivisor), .quot(xout_tmp), .remd(xremd), .rfd(xrfd));
        divider ydiv(.clk(clock), .dividend(ydividend), .divisor(ydivisor), .quot(yout_tmp), .remd(yremd), .rfd(yrfd));

endmodule


// ps2count takes a start signal and counts to COUNT_TO, and outputs a done signal when done.
// used by clkgen, interrupt, and ps2fsm to count durations of time.
module ps2count(reset, clock, interrupt, count_to, count_start, count_reset, count_done, count);
        input reset;
        input clock;
        input interrupt;
        input [20:0] count_to;
        input count_start;
        input count_reset;
        output count_done;

        output [20:0] count;
```

```
        parameter IDLE = 0;
        parameter COUNT = 1;

        reg [20:0] count;
        reg count_done;
        reg state = IDLE;
        reg next;

        always @ (posedge clock)
        begin
                if (reset || count_reset || interrupt)
                begin
                        state <= IDLE;
                        count <= 21'd0;
                end
                else
                        state <= next;

                if (next == IDLE)
                        count <= 21'd0;
                else
                        count <= count + 1;

                if (count == count_to - 1)
                        count_done <= 1'b1;
                else
                        count_done <= 1'b0;
        end

        always @ (count_start or count or state or next)
        begin
                case (state)
                        IDLE: next = count_start ? COUNT : IDLE;
                        COUNT: next = (count >= count_to) ? IDLE : COUNT;
                        default: next = IDLE;
                endcase
        end

endmodule


// ps2clkgen takes a clk_start signal from the ps2fsm, and begins generating clock cycles.
// the clock frequency should be between 10-16.7 kHz.  We choose 13.5 kHz for our implementation.
// clkgen counts for 11 cycles, because all read and written data is 11 bits long.
module ps2clkgen(reset, clock, interrupt, clk_start, count_done, count_start, ps2clkout);
        input reset;
        input clock;
        input interrupt;
```

```
        input clk_start;
        input count_done;
        output count_start;
        output ps2clkout;

        parameter IDLE = 0;
        parameter START_HIGH = 1; // start counting high
        parameter COUNT_HIGH = 2; // continue counting high
        parameter START_LOW = 3;  // start counting low
        parameter COUNT_LOW = 4;  // continue counting low

        reg count_start;
        reg ps2clkout;
        reg [3:0] count; // Generate 11 clock cycles
        reg [2:0] state = IDLE;
        reg [2:0] next;

        always @ (posedge clock)
        begin
                if (reset || interrupt)
                begin
                        state <= IDLE;
                        count <= 4'd0;
                end
                else
                        state <= next;

                if (next == IDLE)
                begin
                        count <= 4'd0;
                        ps2clkout <= 1'b1;
                end
                else if (next == START_HIGH || next == COUNT_HIGH)
                        ps2clkout <= 1'b1;
                else if (next == START_LOW || next == COUNT_LOW)
                        ps2clkout <= 1'b0;

                count_start <= 1'b0;
                if (next == START_HIGH || next == START_LOW)
                        count_start <= 1'b1;

                if (next == START_LOW)
                        count <= count + 1;
        end

        always @ (clk_start or count or count_done or state or next)
        begin
                case (state)
                        IDLE: next = clk_start ? START_HIGH : IDLE;
```

```
                        START_HIGH: next = COUNT_HIGH;
                        COUNT_HIGH: next = count_done ? START_LOW : COUNT_HIGH;
                        START_LOW: next = COUNT_LOW;
                        COUNT_LOW: next = count_done ? (count < 4'd11 ? START_HIGH : IDLE) : COUNT_LOW;
                        default: next = IDLE;
                endcase
        end

endmodule


// ps2interrupt takes the clock line from the host and counts whenever the clock line is low.
// if the clock line is low for 100 microseconds, send interrupt signal to rest of protocol
// to prepare to read data.
module ps2interrupt(reset, clock, ps2clkin, count_done, count_start, count_reset, interrupt);
        input reset;
        input clock;
        input ps2clkin;
        input count_done;
        output count_start;
        output count_reset;
        output interrupt;

        reg count_start;
        reg count_reset;
        reg interrupt;

        always @ (posedge clock)
        begin
                if (reset || ps2clkin)
                begin
                        count_start <= 1'b0;
                        count_reset <= 1'b1;
                        interrupt <= 1'b0;
                end
                else
                begin
                        count_start <= 1'b1;
                        count_reset <= 1'b0;
                        if (count_done)
                                interrupt <= 1'b1;
                        else
                                interrupt <= interrupt;
                end
        end

endmodule
```

```
// ps2serialize takes a start signal from the ps2fsm and a packet of data from the controller to be written.
//      it waits for clkgen to begin generating clock cycles, and dumps bits one by one, least significant bit first,
// onto the data line on the rising edge of ps2clk.  Basically a shift register.         Outputs done back to
// ps2fsm to signal completion.
module ps2serialize(reset, clock, interrupt, ps2clkout, start, packet, data, done);
        input reset;
        input clock;
        input interrupt;
        input ps2clkout;
        input start;
        input [10:0] packet;
        output data;
        output done;

        parameter IDLE = 0;
        parameter HIGH_START = 1;
        parameter HIGH = 2;
        parameter LOW_START = 3;
        parameter LOW = 4;

        reg [10:0] packet_data;
        reg data;
        reg done;
        reg [2:0] state = IDLE;
        reg [2:0] next;

        always @ (posedge clock)
        begin
                if (reset || interrupt)
                         state <= IDLE;
                else
                        state <= next;

                done <= 1'b0;
                if (next == IDLE)
                begin
                        packet_data <= packet;
                        data <= 1'b1;
                end
                else if (next == HIGH_START)
                begin
                        packet_data <= (packet_data >> 1);
                        data <= packet_data[0];
                        if (packet_data == 11'h1)
                                done <= 1'b1;
                end
        end

        always @ (start or ps2clkout or packet_data or state or next)
```

```
        begin
                case (state)
                        // Assumes period of ps2clkout > 2 * clock (reasonable in practice = thousands)
                        IDLE: next = start ? HIGH_START : IDLE;
                        HIGH_START: next = HIGH;
                        HIGH: next = ps2clkout ? HIGH : LOW_START;
                        LOW_START: next = LOW;
                        LOW: next = (packet_data == 11'h0) ? IDLE : (ps2clkout ? HIGH_START : LOW);
                        default: next = IDLE;
                endcase
        end

endmodule


// ps2deserialize does the opposite of serialize.  After getting a start signal from the ps2fsm,
// it waits for clock signals and reads data from the data line from the host on every rising edge.
// Basically another shift register.  Outputs done back to ps2fsm to signal completion.
module ps2deserialize(reset, clock, interrupt, ps2clkout, start, data, packet, done);
        input reset;
        input clock;
        input interrupt;
        input ps2clkout;
        input start;
        input data;
        output [10:0] packet;
        output done;

        parameter IDLE = 0;
        parameter HIGH_START = 1;
        parameter HIGH = 2;
        parameter LOW_START = 3;
        parameter LOW = 4;

        reg [10:0] packet, packet_data;
        reg done;
        reg [3:0] count;
        reg [2:0] state = IDLE;
        reg [2:0] next;

        always @ (posedge clock)
        begin
                if (reset || interrupt)
                        state <= IDLE;
                else
                        state <= next;

                done <= 1'b0;
                if (next == IDLE)
```

```
                begin
                        packet_data <= 11'b0;
                        count <= 4'd0;
                end
                else if (next == HIGH_START)
                begin
                        packet_data <= packet_data + (data << count);
                        if (count == 4'd10)
                        begin
                                done <= 1'b1;
                                packet <= packet_data + (data << count);
                        end
                end
                else if (next == LOW_START)
                                count <= count + 1;
        end

        always @ (start or ps2clkout or count or state or next)
        begin
                case (state)
                        // Assumes period of ps2clkout > 2 * clock (reasonable in practice = thousands)
                        IDLE: next = start ? HIGH_START : IDLE;
                        HIGH_START: next = HIGH;
                        HIGH: next = ps2clkout ? HIGH : LOW_START;
                        LOW_START: next = LOW;
                        LOW: next = (count == 4'd11) ? IDLE : (ps2clkout ? HIGH_START : LOW);
                        default: next = IDLE;
                endcase
        end

endmodule


// ps2fsm controls the read/write/acknowledge states of the protocol.
// start in idle.  if no interrupt, count to 150 microseconds of clock line high before entering write state.
// if interrupted, go into read_ready state, and wait for host to pull data line low.
// tells serialize and deserialize to activate at the appropriate read/write states.
// at the end of a read, outputs ack to pull data line low for one last clock cycle -- signals receipt of data.
module ps2fsm(power, reset, clock, interrupt, ps2clkin, ps2datain, ps2clkout,
                                write_data_enable, count_done, write_done, read_done,
                                      clk_start, count_start, write_enable, read_enable, ack);

        input power;
        input reset;
        input clock;
        input interrupt;
        input ps2clkin;
        input ps2datain;
        input ps2clkout;
```

```
        input write_data_enable;
        input count_done;
        input write_done;
        input read_done;
        output clk_start;
        output count_start;
        output write_enable;
        output read_enable;
        output ack;

        parameter OFF = 0;
        parameter IDLE = 1;
        parameter COUNT_HIGH_START = 2;
        parameter COUNT_HIGH = 3;
        parameter WRITE = 4;
        parameter READ_READY = 5;
        parameter READ = 6;
        parameter ACK1 = 7;
        parameter ACK2 = 8;

        reg count_start = 1'b0;
        reg clk_start = 1'b0;
        reg write_enable = 1'b0;
        reg read_enable = 1'b0;
        reg ack = 1'b0;
        reg [3:0] prev; // store previous value on interrupt in case host is not sending data
        reg [3:0] state = OFF;
        reg [3:0] next;

        always @ (posedge clock)
        begin
                if (!power)
                        state <= OFF;
                else if (state == OFF || reset)
                        state <= IDLE;
                else if (interrupt)
                begin
                        if (state != READ_READY)
                        begin
                                prev <= state;
                                if (state == COUNT_HIGH || state == WRITE)
                                        prev <= COUNT_HIGH_START;
                        end
                        state <= READ_READY;
                end
                else
                        state <= next;

                if (next == COUNT_HIGH_START)
```

```
                        count_start <= 1'b1;
                else
                        count_start <= 1'b0;

                if (next == WRITE)
                begin
                        write_enable <= 1'b1;
                        read_enable <= 1'b0;
                        clk_start <= 1'b1;
                end
                else if (next == READ)
                begin
                        write_enable <= 1'b0;
                        read_enable <= 1'b1;
                        clk_start <= 1'b1;
                end
                else
                begin
                        write_enable <= 1'b0;
                        read_enable <= 1'b0;
                        clk_start <= 1'b0;
                end

                if (next == ACK1 || next == ACK2)
                        ack <= 1'b1;
                else
                        ack <= 1'b0;
        end

        always @ (ps2clkin or ps2datain or ps2clkout
                                or write_data_enable or count_done or write_done or read_done or state or next)
        begin
                case (state)
                        IDLE: next = (ps2clkout & write_data_enable) ? COUNT_HIGH_START : IDLE;
                        COUNT_HIGH_START: next = COUNT_HIGH;
                        COUNT_HIGH: next = count_done ? WRITE : COUNT_HIGH;
                        WRITE: next = write_done ? IDLE : WRITE;
                        READ_READY: next = (ps2clkin & ~ps2datain) ? READ : ((ps2clkin & ps2datain) ? prev : READ_READY);
                        READ: next = read_done ? ACK1 : READ;
                        ACK1: next = ~ps2clkout ? ACK2 : ACK1;
                        ACK2: next = ps2clkout ? IDLE : ACK2;
                        default: next = IDLE;
                endcase
        end

endmodule


// ps2protocol puts together the data reading and writing to ps2.  can read and write a generic message.
```

```
module ps2protocol(power, reset, clock, write_data_enable, write_data, read_data, write_done, read_done,
                                                ps2clkin, ps2datain, ps2clkout, ps2dataout,
                                                write_enable, read_enable, state_fsm, interrupt);

        input power;
        input reset;
        input clock;
        input write_data_enable;
        input [10:0] write_data;
        output [10:0] read_data;
        output write_done;
        output read_done;

        input ps2clkin;
        input ps2datain;
        output ps2clkout;
        output ps2dataout;

        // outputs for debugging
        output write_enable;
        output read_enable;
        output [3:0] state_fsm;
        output interrupt;

        // Change these count values when synthesizing
//      parameter CLK_COUNT = 6;
//      parameter INT_COUNT = 16;
//      parameter HIGH_COUNT = 10;

//      parameter CLK_COUNT = 1000; // 13.5 kHz ps2clk at 27 MHz
//      parameter INT_COUNT = 2700; // 100 us interrupt at 27 MHz
//      parameter HIGH_COUNT = 4050; // 150 us high at 27 MHz

        parameter CLK_COUNT = 1167; // 13.5 kHz ps2clk at 31.5 MHz
        parameter INT_COUNT = 3150; // 100 us interrupt at 31.5 MHz
        parameter HIGH_COUNT = 4725; // 150 us high at 27 MHz

        wire interrupt, clk_start;
        wire write_enable, read_enable;
        wire write_start, read_start;
        wire count_clk_start, count_int_start, count_high_start;
        wire count_clk_done, count_int_done, count_high_done;
        wire count_int_reset;
        wire dataout, ack;
        wire [3:0] state_fsm, prev;

        ps2clkgen clkgen(reset, clock, interrupt, clk_start, count_clk_done, count_clk_start, ps2clkout);
        ps2interrupt interr(reset, clock, ps2clkin, count_int_done, count_int_start, count_int_reset, interrupt);
```

```
        ps2count clk_count(reset, clock, interrupt, CLK_COUNT, count_clk_start, interrupt, count_clk_done);
        ps2count int_count(reset, clock, interrupt, INT_COUNT, count_int_start, count_int_reset, count_int_done);
        ps2count high_count(reset, clock, interrupt, HIGH_COUNT, count_high_start, interrupt, count_high_done);

        ps2fsm fsm(power, reset, clock, interrupt, ps2clkin, ps2datain, ps2clkout,
                                write_data_enable, count_high_done, write_done, read_done,
                                clk_start, count_high_start, write_enable, read_enable, ack);
        ps2serialize ser(reset, clock, interrupt, ps2clkout, write_enable, write_data, dataout, write_done);
        ps2deserialize deser(reset, clock, interrupt, ps2clkout, read_enable, ps2datain, read_data, read_done,
                packet_data);
        assign ps2dataout = ack ? 1'b0 : dataout;       // mux: ack is high pulls data line down

endmodule


// ps2control is the data control unit.  It figures out which signals to send at what time, and how
// to reply to signals sent from the host.
module ps2control(power, reset, clock, mouse_data_enable, dx, dy, left, right,
                                                write_done, read_done, read_data, write_data, write_data_enable, update, s
tate, fpar, first);
        input power; // from computer
        input reset;
        input clock;
        input mouse_data_enable; // from mousedata
        input [8:0] dx;
        input [8:0] dy;
        input left; // left mouse click
        input right; // right mouse click
        input write_done;
        input read_done;
        input [10:0] read_data;
        output [10:0] write_data;
        output write_data_enable;
        output update; // update mouse position, to mousedata

        // outputs for debugging
        output [3:0] state;
        output fpar;
        output [7:0] first;

        parameter OFF = 0;
        parameter IDLE = 1;
        parameter FIRST = 2;
        parameter XMOV = 3;
        parameter YMOV = 4;
        parameter UPDATE = 5;
        parameter SELFTEST = 6;
        parameter MOUSEID = 7;
        parameter ACK = 8;
```

```
        reg data_enable = 1'b0;
        reg [10:0] write_data;
        reg write_data_enable = 1'b0;
        reg update = 1'b0;
        reg [3:0] state = OFF;
        reg [3:0] next;

        wire [7:0] first;
        wire fpar, xpar, ypar;

        assign first = {1'b0, 1'b0, dy[8], dx[8], 1'b1, 1'b0, right, left};
        assign fpar = dy[8] ^ dx[8] ^ right ^ left;
        assign xpar = 1'b1 ^ dx[0] ^ dx[1] ^ dx[2] ^ dx[3] ^ dx[4] ^ dx[5] ^ dx[6] ^ dx[7];
        assign ypar = 1'b1 ^ dy[0] ^ dy[1] ^ dy[2] ^ dy[3] ^ dy[4] ^ dy[5] ^ dy[6] ^ dy[7];

        always @ (posedge clock)
        begin
                if (!power)
                begin
                        state <= OFF;
                        data_enable <= 1'b0;
                end
                else if (state == OFF)
                        state <= SELFTEST;
                else if (reset || state == UPDATE)
                        state <= IDLE;
                else if (read_done)
                        state <= ACK;
                else if (state == IDLE && mouse_data_enable && data_enable)
                        state <= FIRST;
                else if (write_done)
                        state <= next;

                if (read_data == {1'b1, 1'b1, 8'hF5, 1'b0}) // Disable Data Reporting
                        data_enable <= 1'b0;
                else if (read_data == {1'b1, 1'b0, 8'hF4, 1'b0}) // Enable Data Reporting
                        data_enable <= 1'b1;

                update <= 1'b0;
                if (write_done && next == UPDATE)
                        update <= 1'b1;

                write_data_enable <= 1'b0;
                if (!write_done && !update && state >= FIRST)
                        write_data_enable <= 1'b1;

        end
```

```
        always @ (mouse_data_enable or write_done or read_done or state or next)
        begin
                case (state)
                        FIRST: next = XMOV;
                        XMOV: next = YMOV;
                        YMOV: next = UPDATE;
                        SELFTEST: next = MOUSEID;
                        MOUSEID: next = IDLE;
                        ACK: next = ((read_data == {1'b1, 1'b1, 8'hFF, 1'b0}) ? SELFTEST : // Reset
                                                        ((read_data == {1'b1, 1'b0, 8'hF2, 1'b0}) ? MOUSEID : // Get Devic
e ID
                                                        IDLE));
                        default: next = IDLE;
                endcase

                if (state == FIRST)
                        write_data = {1'b1, fpar, first, 1'b0}; // first bit
                else if (state == XMOV)
                        write_data = {1'b1, xpar, dx[7:0], 1'b0};       // x movement
                else if (state == YMOV)
                        write_data = {1'b1, ypar, dy[7:0], 1'b0};       // y movement
                else if (state == SELFTEST)
                        write_data = {1'b1, 1'b1, 8'hAA, 1'b0}; // selftest
                else if (state == MOUSEID)
                        write_data = {1'b1, 1'b1, 8'h00, 1'b0}; // mouseid
                else if (state == ACK)
                        write_data = {1'b1, 1'b1, 8'hFA, 1'b0}; // acknowledge
                else
                        write_data = 11'b0;
        end

endmodule


// ps2system puts ps2control and ps2protocol together into a single ps2 entity.
// user inputs dx, dy, left, right, and mouse_data_enable, and receives an update signal
// when this data has been successfully sent to the computer.
module ps2system(power, reset, clock, mouse_data_enable, dx, dy, left, right,
                                        ps2clkin, ps2datain, ps2clkout, ps2dataout, update,
                                        write_data, read_data, write_done, read_done, write_data_enable,
                                        write_enable, read_enable, state_fsm, interrupt, state, fpar, first);
        input power;
        input reset;
        input clock;
        input mouse_data_enable;
        input [8:0] dx;
        input [8:0] dy;
        input left;
        input right;
```

```
        input ps2clkin;
        input ps2datain;
        output ps2clkout;
        output ps2dataout;
        output update;

        // outputs for debugging
        output [10:0] write_data, read_data;
        output write_done, read_done, write_data_enable;

        output write_enable;
        output read_enable;
        output [3:0] state_fsm;
        output interrupt;

        output [3:0] state;
        output fpar;
        output [7:0] first;

        wire [10:0] write_data, read_data;
        wire write_done, read_done;

        // Feed in a delay at startup
//      parameter DELAY = 1350000; // 50 ms delay       at 27 MHz
        parameter DELAY = 1575000; // 50 ms delay at 31.5 MHz
//      parameter DELAY = 50;

        reg count_power_start = 1'b0;
        reg enable = 1'b0;
        wire count_power_done;

        always @ (posedge clock)
        begin
                if (power)
                        count_power_start <= 1'b1;
                else
                begin
                        count_power_start <= 1'b0;
                        enable <= 1'b0;
                end
                if (count_power_done)
                        enable <= 1'b1;
        end

        ps2count power_count(reset, clock, interrupt, DELAY, count_power_start, power, count_power_done);

        ps2protocol protocol(enable, reset, clock, write_data_enable, write_data, read_data, write_done, read_done,
                                                                ps2clkin, ps2datain, ps2clkout, ps2dataout,
                                                                write_enable, read_enable, state_fsm, interrupt);
```

```
        ps2control control(enable, reset, clock, mouse_data_enable, dx, dy, left, right,
                                                        write_done, read_done, read_data, write_data, write_data_enable,
update, state, fpar, first);

endmodule



////////the following are a few pieces of code not written by us but borrowed for the project



///Isaac Rosmarin's ascii to alphanumeric display code

'timescale 1ns / 1ps
///
///
///This module was not written by us.  We obtained it from Isaac Rosmarin, and
///then modified the way a few of the characters looked.
///
///

module alpha_display ( clock, ascii, bits, dots );
//
   input                                clock;
   input                                ascii;
   input        [7:0]   bits;
//
   output       [39:0]  dots;
//
   reg          [39:0]  dots;
//

   always @ ( posedge clock )
        begin
                        if ( ascii )
                                begin

                                        case ( bits )
                                       8'd000: dots <= {8'b00000000, 8'b00000000, 8'b00000000, 8'b00000000, 8'b00000000};
// ' *'

                                                // punctuation
                                       8'd042: dots <= {8'b00101010, 8'b00011100, 8'b00001000, 8'b00011100, 8'b00101010};
// ' *'
                                       8'd043: dots <= {8'b00001000, 8'b00001000, 8'b00111110, 8'b00001000, 8'b00001000};
// ' +'
                                       8'd044: dots <= {8'b00000000, 8'b01000000, 8'b00110000, 8'b00110000, 8'b00000000};
// ' ,'
```

```
                                    8'd045: dots <= {8'b00001000, 8'b00001000, 8'b00001000, 8'b00001000, 8'b00001000};
// ' -'
                                    8'd046: dots <= {8'b00000000, 8'b00000000, 8'b01100000, 8'b01100000, 8'b00000000};
// ' .'
                                    8'd047: dots <= {8'b00100000, 8'b00010000, 8'b00001000, 8'b00000100, 8'b00000010};
// ' /'

                                    // NUMBERS
                                    8'd048: dots <= {8'b00111110, 8'b01010001, 8'b01001001, 8'b01000101, 8'b00111110};
// ' 0'
                                    8'd049: dots <= {8'b00000000, 8'b01000010, 8'b01111111, 8'b01000000, 8'b00000000};
// ' 1'
                                    8'd050: dots <= {8'b01100010, 8'b01010001, 8'b01001001, 8'b01001001, 8'b01000110};
// ' 2'
                                    8'd051: dots <= {8'b00100010, 8'b01000001, 8'b01001001, 8'b01001001, 8'b00110110};
// ' 3'
                                    8'd052: dots <= {8'b00011000, 8'b00010100, 8'b00010010, 8'b01111111, 8'b00010000};
// ' 4'
                                    8'd053: dots <= {8'b00100111, 8'b01000101, 8'b01000101, 8'b01000101, 8'b00111001};
// ' 5'
                                    8'd054: dots <= {8'b00111100, 8'b01001010, 8'b01001001, 8'b01001001, 8'b00110000};
// ' 6'
                                    8'd055: dots <= {8'b00000001, 8'b01110001, 8'b00001001, 8'b00000101, 8'b00000011};
// ' 7'
                                    8'd056: dots <= {8'b00110110, 8'b01001001, 8'b01001001, 8'b01001001, 8'b00110110};
// ' 8'
                                    8'd057: dots <= {8'b00000110, 8'b01001001, 8'b01001001, 8'b00101001, 8'b00011110};
// ' 9'

                                    // MORE punctuation

                                    8'd058: dots <= {8'b00000000, 8'b00000000, 8'b00110110, 8'b00110110, 8'b00000000};
// ' :'

                                    // UPPERCASE LETTERS
                                    8'd065: dots <= {8'b01111110, 8'b00001001, 8'b00001001, 8'b00001001, 8'b01111110};
// ' A'
                                    8'd066: dots <= {8'b01111111, 8'b01001001, 8'b01001001, 8'b01001001, 8'b00110110};
// ' B'
                                    8'd067: dots <= {8'b00111110, 8'b01000001, 8'b01000001, 8'b01000001, 8'b00100010};
// ' C'
                                    8'd068: dots <= {8'b01111111, 8'b01000001, 8'b01000001, 8'b01000001, 8'b00111110};
// ' D'
                                    8'd069: dots <= {8'b01111111, 8'b01001001, 8'b01001001, 8'b01001001, 8'b01000001};
// ' E'

                                    8'd070: dots <= {8'b01111111, 8'b00001001, 8'b00001001, 8'b00001001, 8'b00000001};
```

```
// ' F'
                                    8'd071: dots <= {8'b00111110, 8'b01000001, 8'b01000001, 8'b01010001, 8'b00110010};
// ' G'
                                    8'd072: dots <= {8'b01111111, 8'b00001000, 8'b00001000, 8'b00001000, 8'b01111111};
// ' H'
                                    8'd073: dots <= {8'b01000001, 8'b01000001, 8'b01111111, 8'b01000001, 8'b01000001};
// ' I'
                                    8'd074: dots <= {8'b00110000, 8'b01000000, 8'b01000000, 8'b01000000, 8'b01111111};
// ' J'

                                    8'd075: dots <= {8'b01111111, 8'b00001000, 8'b00001000, 8'b00010100, 8'b01100011};
// ' K'
                                    8'd076: dots <= {8'b01111111, 8'b01000000, 8'b01000000, 8'b01000000, 8'b01100000};
// ' L'
                                    8'd077: dots <= {8'b01111111, 8'b00000010, 8'b00001100, 8'b00000010, 8'b11111111};
// ' M'
                                    8'd078: dots <= {8'b01111111, 8'b00000110, 8'b00001000, 8'b00110000, 8'b01111111};
// ' N'
                                    8'd079: dots <= {8'b00111110, 8'b01000001, 8'b01000001, 8'b01000001, 8'b00111110};
// ' O'

                                    8'd080: dots <= {8'b01111111, 8'b00001001, 8'b00001001, 8'b00001001, 8'b00000110};
// ' P'
                                    8'd081: dots <= {8'b00111110, 8'b01000001, 8'b01010001, 8'b01100001, 8'b01111110};
// ' Q'
                                    8'd082: dots <= {8'b01111111, 8'b00001001, 8'b00011001, 8'b00101001, 8'b01000110};
// ' R'
                                    8'd083: dots <= {8'b00100110, 8'b01001001, 8'b01001001, 8'b01001001, 8'b00110010};
// ' S'
                                    8'd084: dots <= {8'b00000011, 8'b00000001, 8'b01111111, 8'b00000001, 8'b00000011};
// ' T'

                                    8'd085: dots <= {8'b00111111, 8'b01000000, 8'b01000000, 8'b01000000, 8'b00111111};
// ' U'
                                    8'd086: dots <= {8'b00001111, 8'b00011000, 8'b01100000, 8'b00011000, 8'b00001111};
// ' V'
                                    8'd087: dots <= {8'b01111111, 8'b00110000, 8'b00001000, 8'b00110000, 8'b01111111};
// ' W'
                                    8'd088: dots <= {8'b01100011, 8'b00010100, 8'b00001000, 8'b00010100, 8'b01100011};
// ' X'
                                    8'd089: dots <= {8'b00000011, 8'b00000100, 8'b01111000, 8'b00000100, 8'b00000011};
// ' Y'

                                    8'd090: dots <= {8'b01100011, 8'b01000101, 8'b01001001, 8'b01010001, 8'b01100011};
// ' Z'


                                    //LOWERCASE LETTERS
                                    8'd097: dots <= {8'b00100000, 8'b01010100, 8'b01010100, 8'b01010100, 8'b01111000};
```

```
// ' a'                                    8'd098: dots <= {8'b01111110, 8'b01010000, 8'b01010000, 8'b01010000, 8'b00100000};
// ' b'                                    8'd099: dots <= {8'b00110000, 8'b01001000, 8'b01001000, 8'b01001000, 8'b00000000};
// ' c'                                    8'd100: dots <= {8'b00100000, 8'b01010000, 8'b01010000, 8'b01010000, 8'b01111110};
// ' d'                                    8'd101: dots <= {8'b00111000, 8'b01010100, 8'b01010100, 8'b01010100, 8'b00001000};
// ' e'
                                           8'd102: dots <= {8'b00010000, 8'b01111100, 8'b00010010, 8'b00010010, 8'b00000100};
// ' f'  //changing f                      8'd103: dots <= {8'b00001000, 8'b01010100, 8'b01010100, 8'b01010100, 8'b00111100};
// ' g'                                    8'd104: dots <= {8'b01111110, 8'b00010000, 8'b00010000, 8'b00010000, 8'b01100000};
// ' h'                                    8'd105: dots <= {8'b00000000, 8'b01000000, 8'b01111010, 8'b01000000, 8'b00000000};
// ' i'                                    8'd106: dots <= {8'b00100000, 8'b01000000, 8'b01000000, 8'b00111010, 8'b00000000};
// ' j'
                                           8'd107: dots <= {8'b01111110, 8'b00010000, 8'b00101100, 8'b01000000, 8'b00000000};
// ' k'                                    8'd108: dots <= {8'b00000000, 8'b00000000, 8'b01111110, 8'b00000000, 8'b00000000};
// ' l'                                    8'd109: dots <= {8'b01110000, 8'b00011000, 8'b00110000, 8'b00011000, 8'b01110000};
// ' m'                                    8'd110: dots <= {8'b01111000, 8'b00001000, 8'b00001000, 8'b00001000, 8'b01110000};
// ' n'                                    8'd111: dots <= {8'b00110000, 8'b01001000, 8'b01001000, 8'b01001000, 8'b00110000};
// ' o'
                                           8'd112: dots <= {8'b01111100, 8'b00010100, 8'b00010100, 8'b00010100, 8'b00001000};
// ' p'                                    8'd113: dots <= {8'b00001000, 8'b00010100, 8'b00010100, 8'b00010100, 8'b01111100};
// ' q'                                    8'd114: dots <= {8'b01111000, 8'b00001000, 8'b00001000, 8'b00001000, 8'b00010000};
// ' r'                                    8'd115: dots <= {8'b00001000, 8'b01010100, 8'b01010100, 8'b01010100, 8'b00100000};
// ' s'                                    8'd116: dots <= {8'b00001000, 8'b00001000, 8'b01111110, 8'b00001000, 8'b00001000};
// ' t'
                                           8'd117: dots <= {8'b00111000, 8'b01000000, 8'b01000000, 8'b01000000, 8'b00111000};
// ' u'                                    8'd118: dots <= {8'b00001000, 8'b00110000, 8'b01000000, 8'b00110000, 8'b00001000};
// ' v'                                    8'd119: dots <= {8'b00111000, 8'b01000000, 8'b01110000, 8'b01000000, 8'b00111000};
// ' w'
```

```
                                        8'd120: dots <= {8'b01000100, 8'b00101000, 8'b00010000, 8'b00101000, 8'b01000100};
// ' x'
                                        8'd121: dots <= {8'b00011100, 8'b01010000, 8'b01010000, 8'b01010000, 8'b00111100};
// ' y'

                                        8'd122: dots <= {8'b01000100, 8'b01100100, 8'b01010100, 8'b01001100, 8'b01000100};
// ' z'
               default:
                                dots <= {8'b01010101, 8'b01010101, 8'b01010101, 8'b01010101, 8'b01010101};
               endcase // case( bits )
                 end

        else
          begin
             case ( bits [3:0] )
               8'd15: dots <= {8'b01111111, 8'b00001001, 8'b00001001, 8'b00001001, 8'b00000001};        // ' F'
               8'd14: dots <= {8'b01111111, 8'b01001001, 8'b01001001, 8'b01001001, 8'b01000001};        // ' E'
               8'd13: dots <= {8'b01111111, 8'b01000001, 8'b01000001, 8'b01000001, 8'b00111110};        // ' D'
               8'd12: dots <= {8'b00111110, 8'b01000001, 8'b01000001, 8'b01000001, 8'b00100010};        // ' C'
               8'd11: dots <= {8'b01111111, 8'b01001001, 8'b01001001, 8'b01001001, 8'b00110110};        // ' B'
               8'd10: dots <= {8'b01111110, 8'b00001001, 8'b00001001, 8'b00001001, 8'b01111110};        // ' A'

               8'd09: dots <= {8'b00000110, 8'b01001001, 8'b01001001, 8'b00101001, 8'b00011110}; // ' 9'
               8'd08: dots <= {8'b00110110, 8'b01001001, 8'b01001001, 8'b01001001, 8'b00110110}; // ' 8'
               8'd07: dots <= {8'b00000001, 8'b01110001, 8'b00001001, 8'b00000101, 8'b00000011}; // ' 7'
               8'd06: dots <= {8'b00111100, 8'b01001010, 8'b01001001, 8'b01001001, 8'b00110000}; // ' 6'
               8'd05: dots <= {8'b00100111, 8'b01000101, 8'b01000101, 8'b01000101, 8'b00111001}; // ' 5'
               8'd04: dots <= {8'b00011000, 8'b00010100, 8'b00010010, 8'b01111111, 8'b00010000}; // ' 4'
               8'd03: dots <= {8'b00100010, 8'b01000001, 8'b01001001, 8'b01001001, 8'b00110110}; // ' 3'
               8'd02: dots <= {8'b01100010, 8'b01010001, 8'b01001001, 8'b01001001, 8'b01000110}; // ' 2'
               8'd01: dots <= {8'b00000000, 8'b01000010, 8'b01111111, 8'b01000000, 8'b00000000}; // ' 1'
               8'd00: dots <= {8'b00111110, 8'b01010001, 8'b01001001, 8'b01000101, 8'b00111110}; // ' 0'
               default:
                                dots <= {8'b00101010, 8'b00101010, 8'b00101010, 8'b00101010, 8'b00101010};

             endcase // case( bits)
          end

     end


endmodule


///lab3 alphanumeric display code modified by Chris Buenrostro

///
///
///This code was not written by us.  It was used in lab3 and later modified by
```

```
///Chris Buenrostro.
///
///

module alphanumeric_displays
(
        global_clock, manual_reset, disp_test,
        disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b, disp_data_out,
        dots
);

   input                                        global_clock;
        input                                   manual_reset;
   output                                disp_blank;
   output                                disp_clock;
   output                                disp_data_out;          // serial data to displays
   output                                disp_rs;                // register select
   output                                disp_ce_b;              // chip enable
   output                                disp_reset_b;           // display reset
   input        [639:0]         dots;
        input                                   disp_test;                              // check functionality of display

   assign                                disp_blank = 1'b0;      // 0=not blanked

   reg                                   disp_clock;
        reg                                     disp_data_out;
        reg                                     disp_rs;
        reg                                     disp_ce_b;
        reg                                     disp_reset_b;
   // Internal signals
   reg          [5:0]   count;
   reg          [7:0]   state;
   reg          [9:0]   dot_index;
   reg          [639:0]         ldots;
        ////////////////////////////////
        //
        //      There are four Control Words
        //
        ////////////////////////////////
        //      parameter       control_reg_load_length = 8;    //first four Banks
        //      parameter       control_reg_load_length = 16;   //second four Banks
        //      parameter       control_reg_load_length = 24;   //third four Banks
        parameter       control_reg_load_length = 32;           //all four Banks
   reg          [control_reg_load_length - 1:0] control;

        parameter       RESET                                           =       0;
        parameter       END_RESET                               =       1;
        parameter       INIT_DOT_REG                    =       2;
        parameter       LATCH_INIT_DOT_DATA     =       3;
```

```
        parameter       SELECT_CONTROL_REG      =       4;
        parameter       SET_CONTROL_REG         =       5;
        parameter       LATCH_CONTROL_REG               =       6;
        parameter       SELECT_DOT_REG                  =       7;
        parameter       FILL_DOT_REG                    =       8;
        parameter       LATCH_DOT_DATA                  =       9;
        parameter       LOAD_NEW_DOT_DATA               =       10;

        parameter       control_reg_value               =  32'h7F7F7F7F;        // Controls LED brightness

    ////////////////////////////////////////////////////////////////////////
    //
    // Initial Reset Generation
        //
        //              *** SRL16 is a variable-width shift register
    //
    ////////////////////////////////////////////////////////////////////////

    wire reset;

    SRL16 reset_sr (.D(1'b0), .CLK(global_clock), .Q(reset),
                    .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
    defparam reset_sr.INIT = 16'hFFFF;

    ////////////////////////////////////////////////////////////////////////
    //
    // Sequential Logic Block
    //
    ////////////////////////////////////////////////////////////////////////

    always @(posedge global_clock)
                begin
                if (reset || manual_reset)
                                begin
                                        count                   <= 0;
                                        disp_clock              <= 0;
                                        disp_data_out   <= 0;
                                        disp_rs                         <= 0;
                                        disp_ce_b               <= 1;
                                        disp_reset_b    <= 0;
                                        dot_index               <= 0;
                                        state                   <= 0;
                                        control                         <= control_reg_value;
                                end
                        else if (count==26)
                begin
                                        count                   <= count+1;
                                        disp_clock              <= 1;
                end
```

```
                else if (count==53)
                                begin
                                        count                   <= 0;
                                        disp_clock              <= 0;

        ////////////////////////////////////////////////////////////////////////
        //
        // Display State Machine
        //
        ////////////////////////////////////////////////////////////////////////

                                casex (state)

                                        RESET:
                                                begin
                                                        disp_data_out           <= 1'b0;
                                                        disp_rs                                 <= 1'b0; // dot register
                                                        disp_ce_b                       <= 1'b1;
                                                        disp_reset_b            <= 1'b0;
                                                        dot_index                       <= 0;
                                                        state                           <= END_RESET;
                                          end
                                       END_RESET:
                                          begin
                                                        disp_reset_b            <= 1'b1;
                                                        state                           <= INIT_DOT_REG;
                                          end
                                       INIT_DOT_REG:
                                          begin
                                                        disp_ce_b                       <= 1'b0;
                                                        disp_data_out           <= 1'b0;

                                                        if (dot_index == 0)
                                                        state                   <= LATCH_INIT_DOT_DATA;
                                                        else
                                                                dot_index               <= dot_index-1;
                                                end
                                     LATCH_INIT_DOT_DATA:
                                          begin
                                                        disp_ce_b                       <= 1'b1;
                                                        dot_index                       <= control_reg_load_length-1;
                                                        state                           <= SELECT_CONTROL_REG;
                                          end
                                            SELECT_CONTROL_REG:
                                                begin
                                                        disp_rs                                 <= 1'b1;
                                                        state                           <= SET_CONTROL_REG;
                                                end
                                       SET_CONTROL_REG:
```

```
                        begin
                                disp_ce_b <= 1'b0;
                                disp_data_out           <= control[dot_index];

                                if (dot_index == 0)
                                state                   <= LATCH_CONTROL_REG;
                                else
                                dot_index               <= dot_index-1;
                        end
                      LATCH_CONTROL_REG:
                        begin
                                disp_ce_b                       <= 1'b1;
                                dot_index                       <= 639;
                                state                           <= SELECT_DOT_REG;
                        end
                          SELECT_DOT_REG:
                                  begin
                                disp_rs                                 <= 1'b0;
                                state                                   <= FILL_DOT_REG;
                                  end
                      FILL_DOT_REG:
                        begin
                                disp_ce_b                       <= 1'b0;

                                if (disp_test)
                                        disp_data_out   <= 1'b1;
                                else
                                        disp_data_out   <= ldots[dot_index];

                                if (dot_index == 0)
                                state                   <= LATCH_DOT_DATA;
                                else
                                        dot_index               <= dot_index-1;
                                  end
                     LATCH_DOT_DATA:
                        begin
                                disp_ce_b                       <= 1'b1;
                                dot_index                       <= 639;
                                state                           <= LOAD_NEW_DOT_DATA;
                        end
                     LOAD_NEW_DOT_DATA:
                        begin
                                ldots                           <= dots;
                                state                           <= FILL_DOT_REG;
                        end
                          default:      state                   <= state;
                  endcase
          end
  else
```

```
               count <= count+1;
        end
endmodule



//debounce module

// 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;   // .1 sec
   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
```