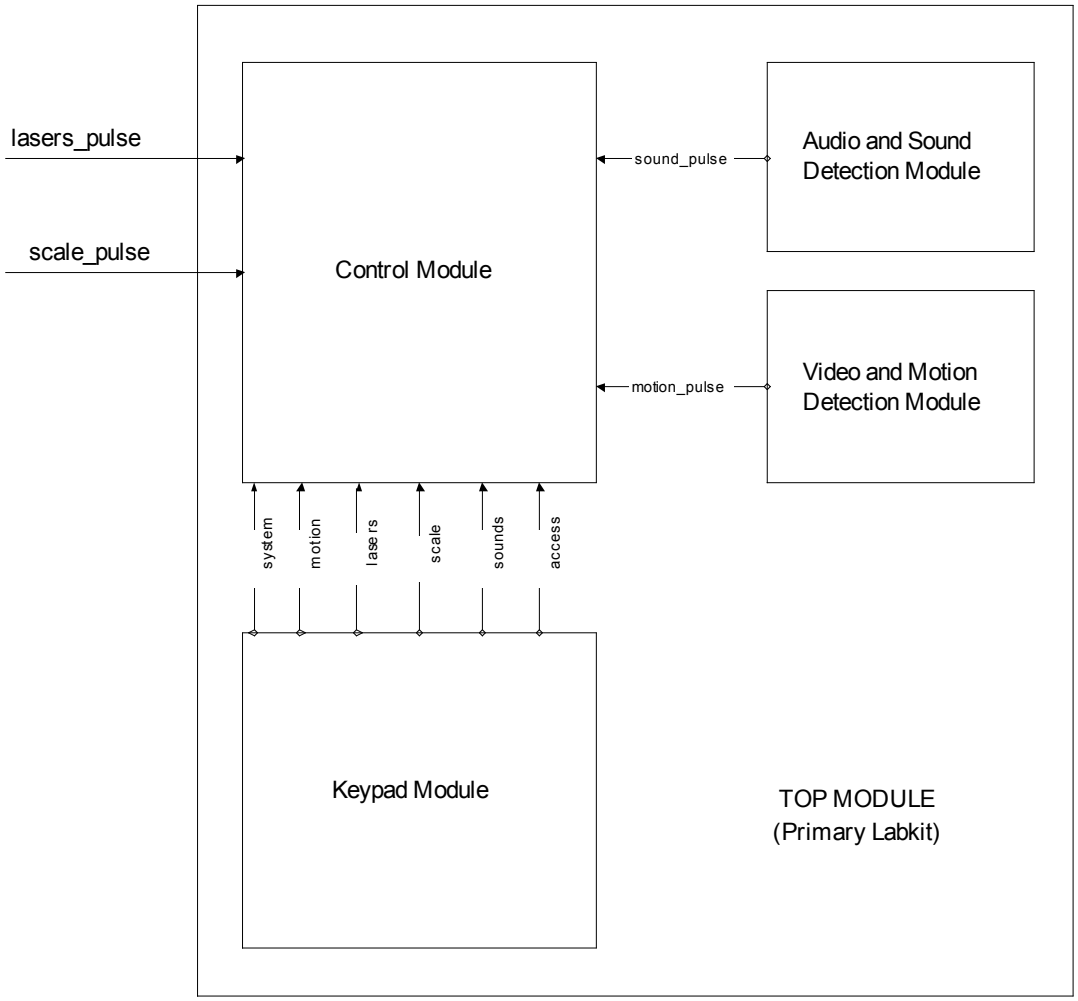

Figure 1. Top Module (Primary Labkit)

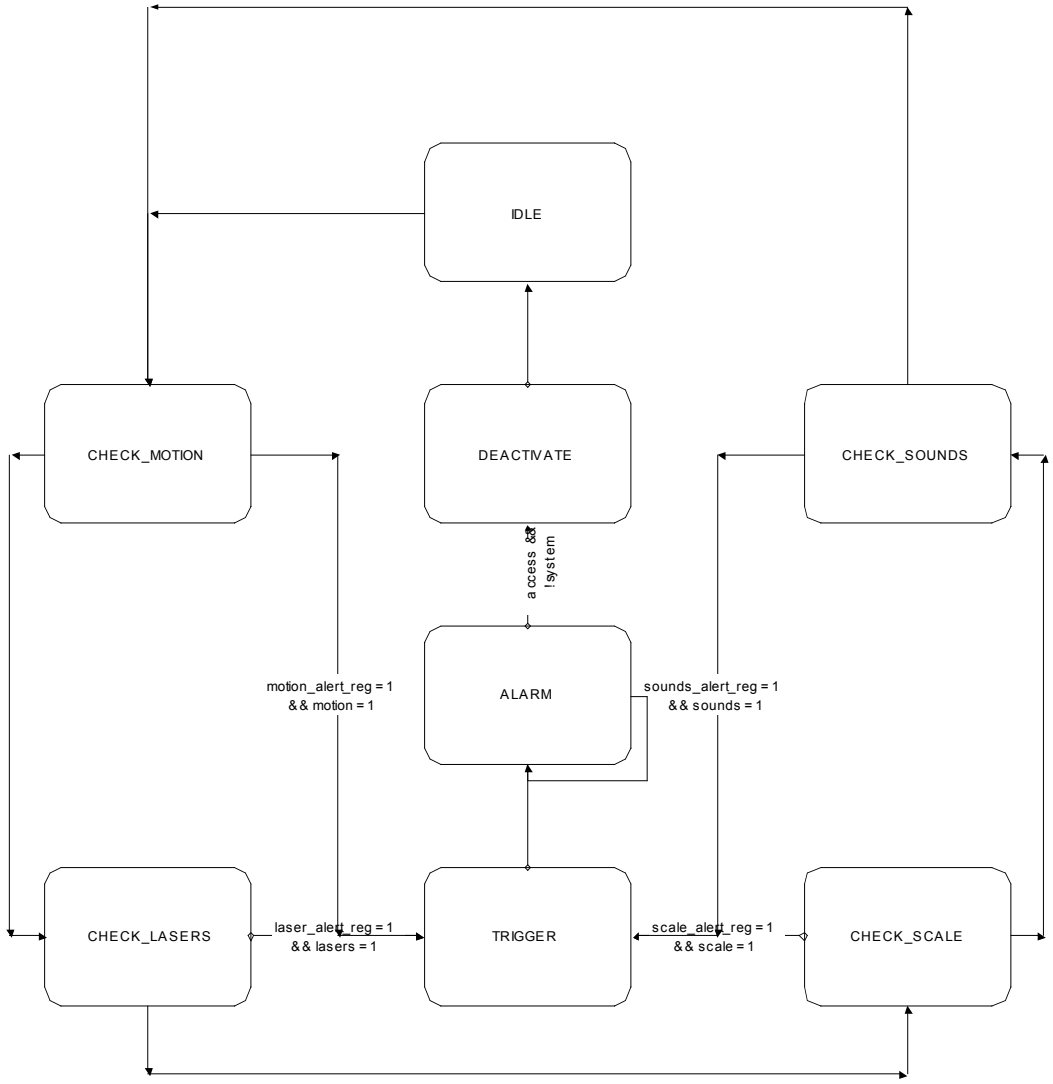

Figure 2. Major FSM

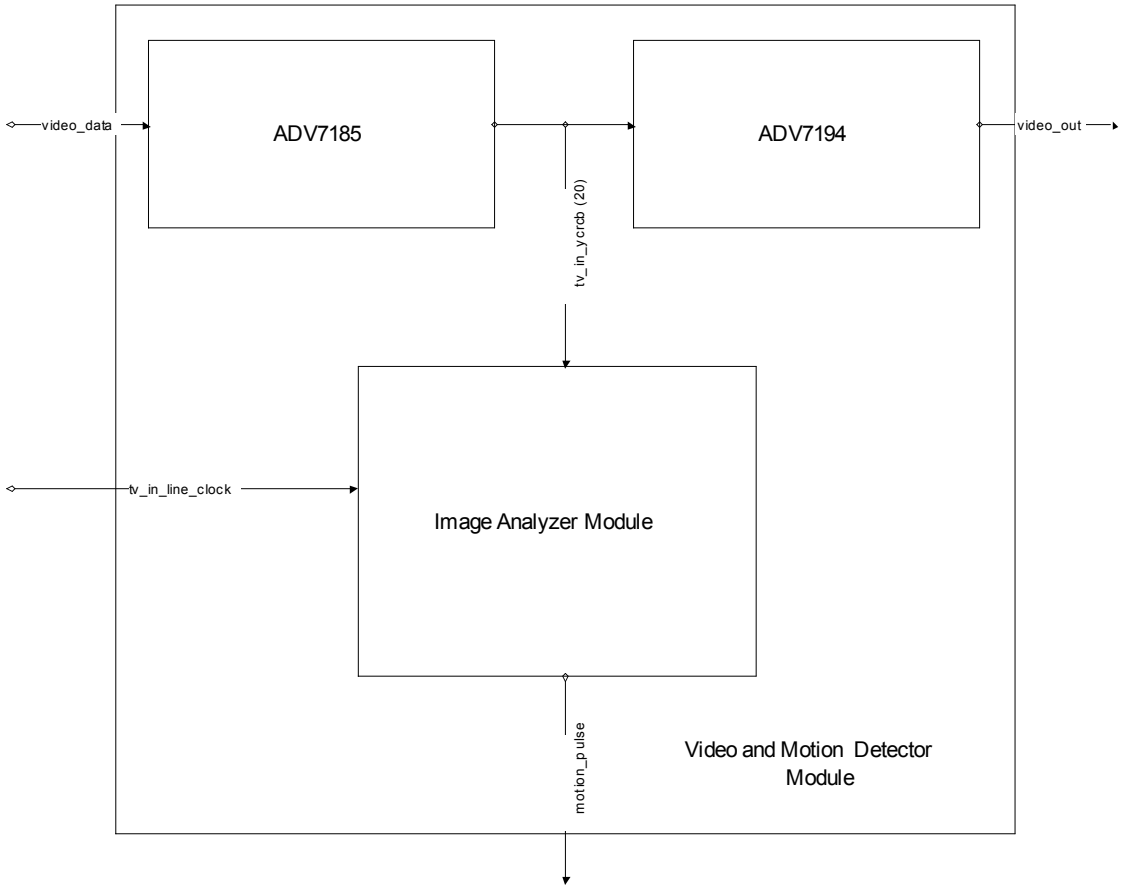

Figure 3. Video and Motion Detector Block

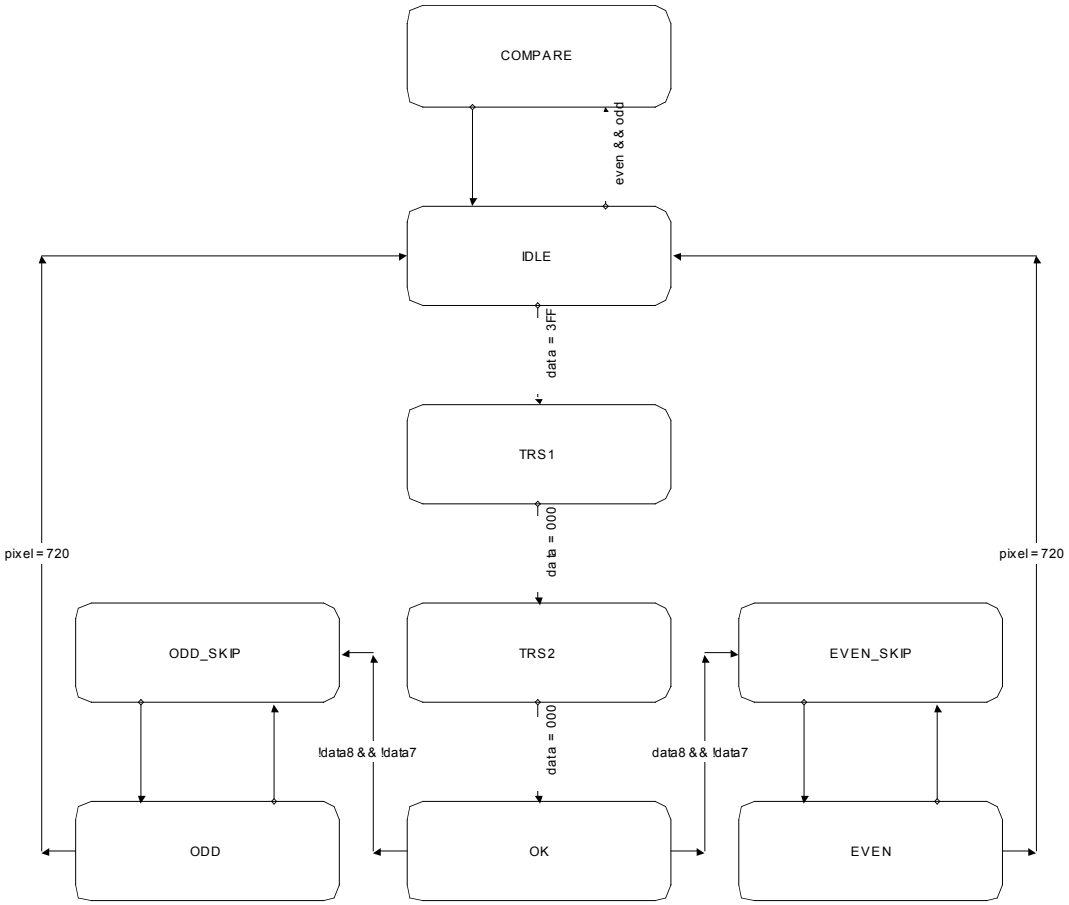

Figure 4. Image Analyzer FSM

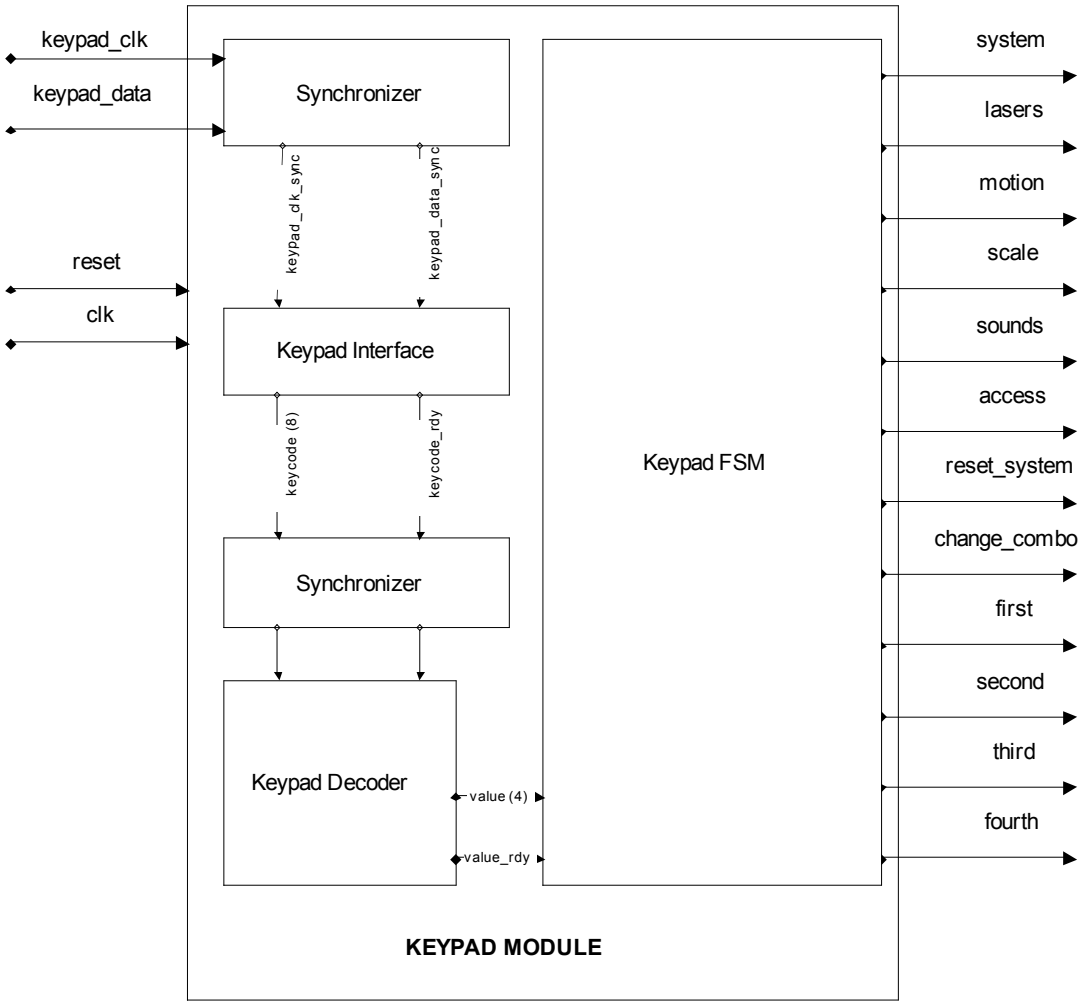

Figure 5. Keypad Block

Figure 6. Keypad FSM

Alarm Block

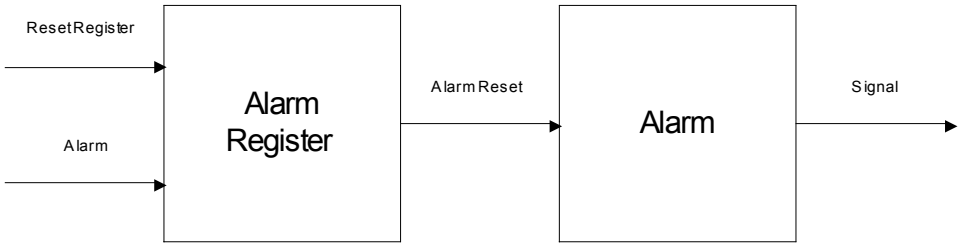

Figure 7. Alarm Block

Laser Net Block & State Transition Diagram

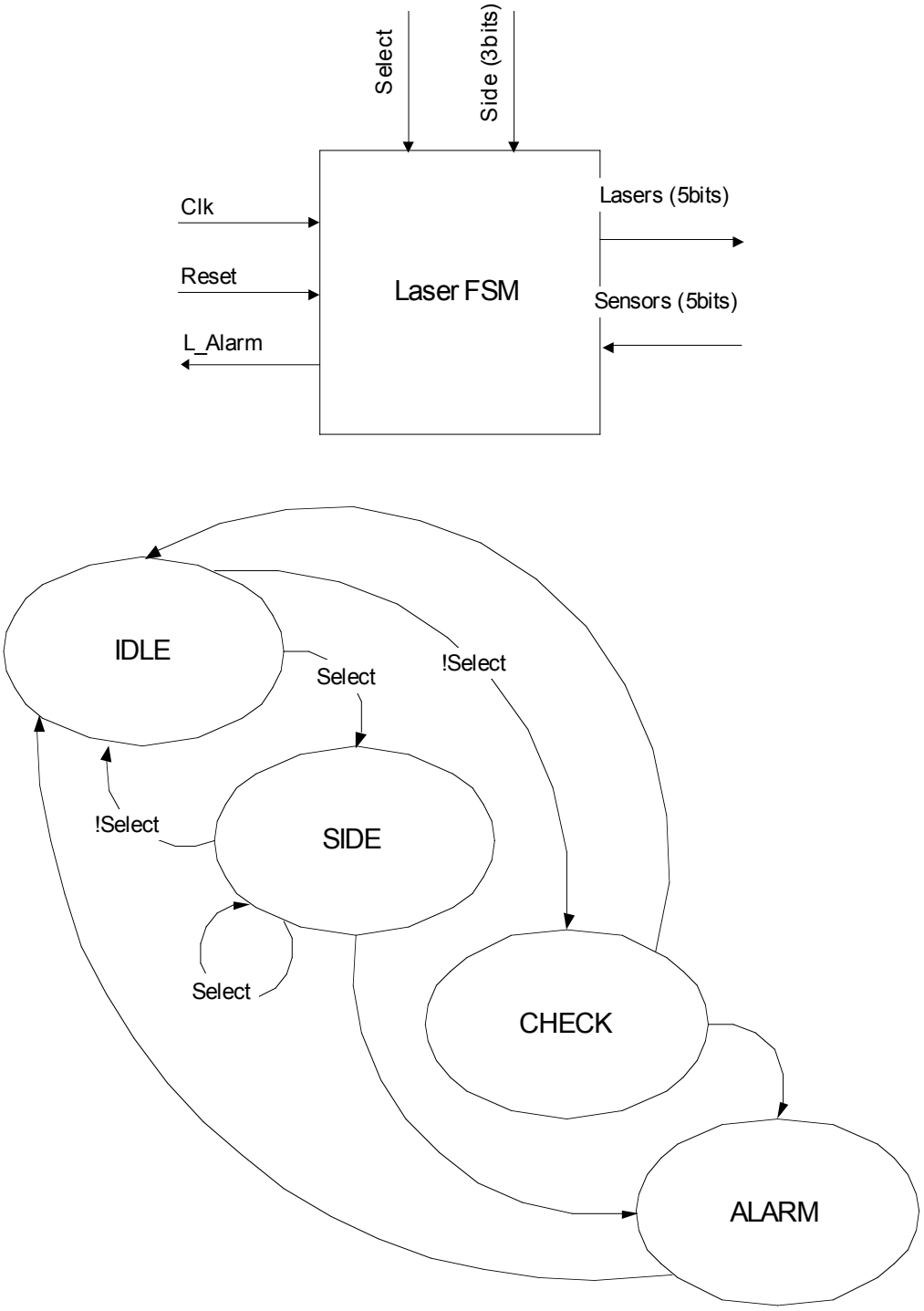

Figure 8. Laser Block and FSM

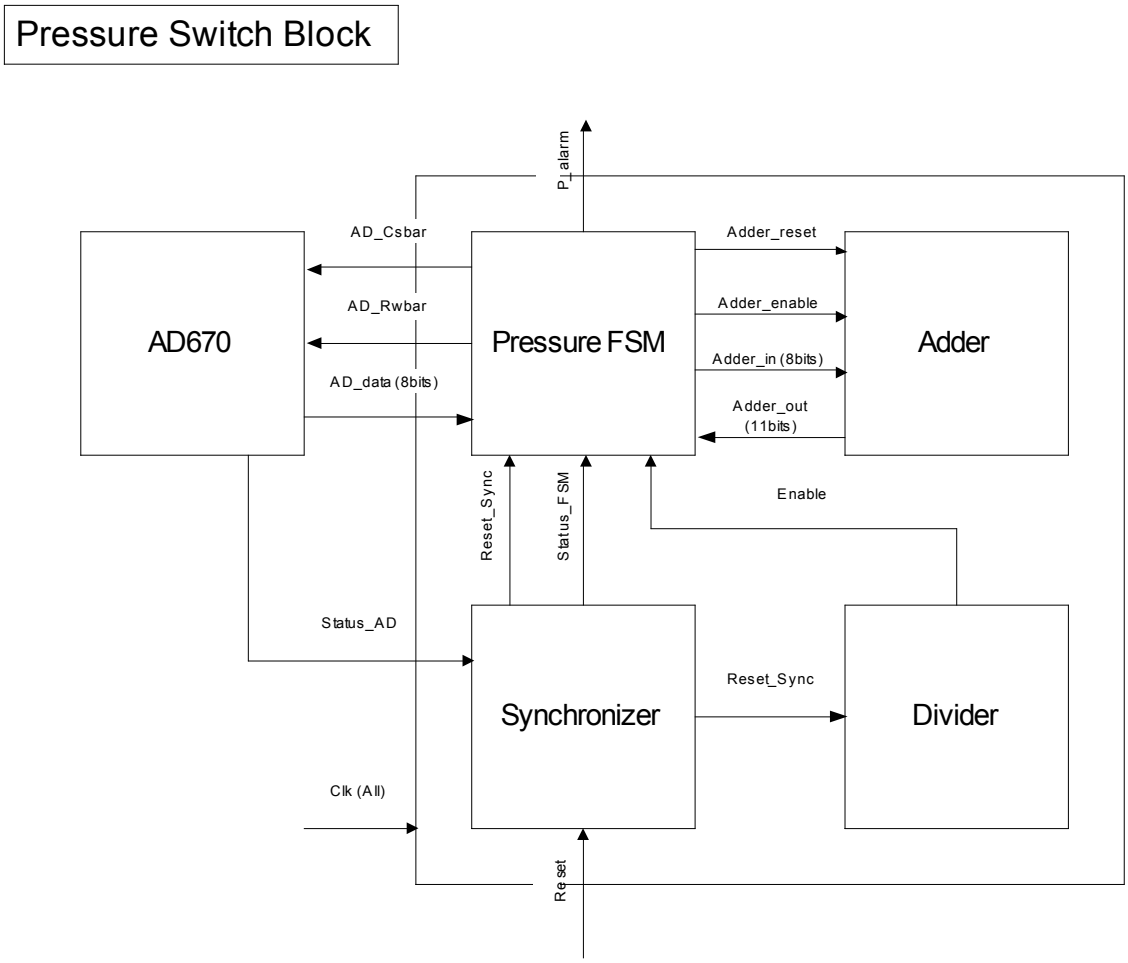

Figure 9. Pressure Block

Pressure Switch State Transition Diagram

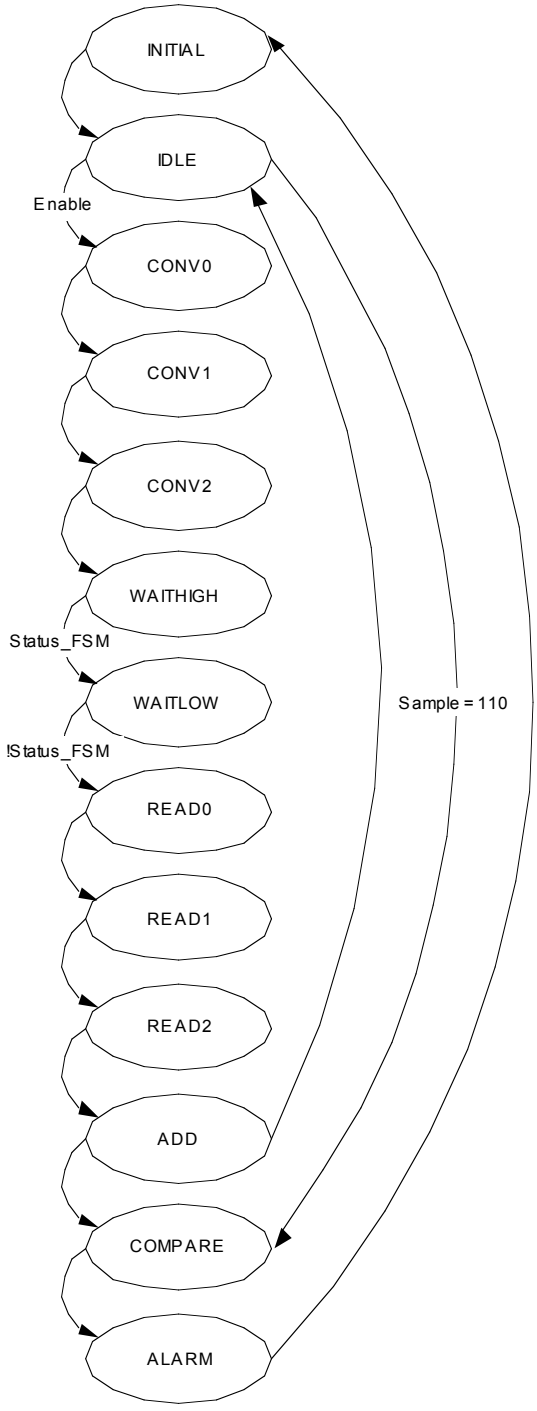

Figure 10. Pressure FSM

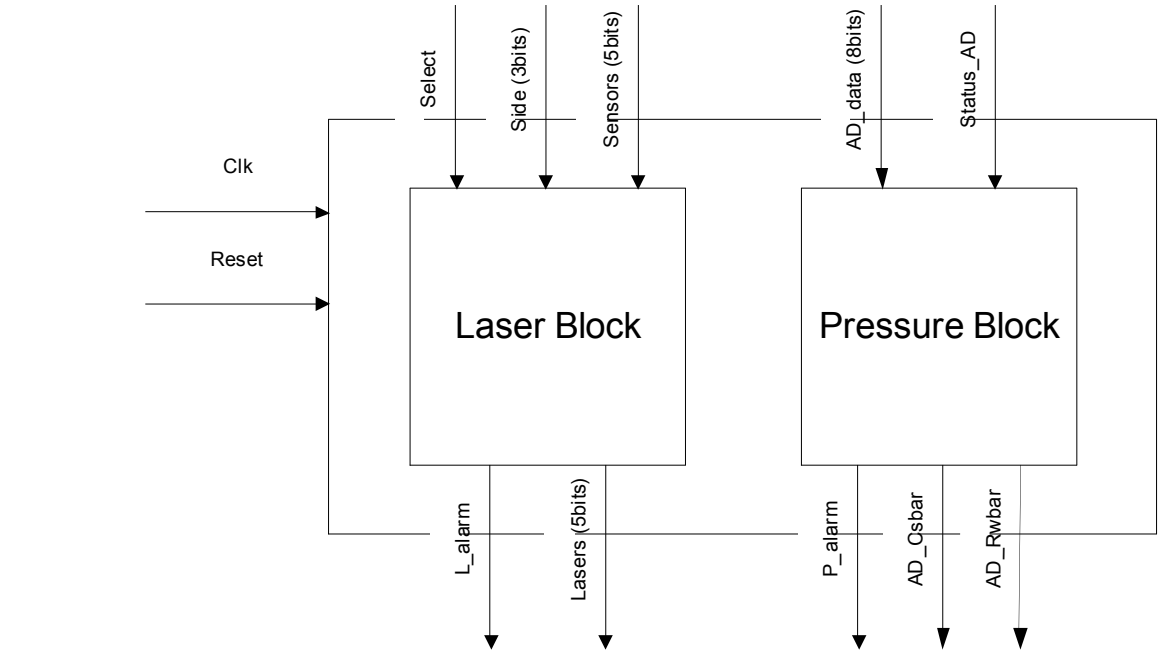

Figure 11. Top (Secondary Labkit)

```
//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Audio/Video Test
//
// For Labkit Revision 004
//
//
// Created: November 3, 2004
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////
//
//      MODIFIED BY: CHRISTIAN DEONIER
//      May 12' 2005
//
//////////////////////////////////////////////////////////////////////////////////


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_i2c_data, tv_in_fifo_read, tv_in_fifo_clock,
       tv_in_iso, tv_in_reset_b, tv_in_clock;

inout  [35:0] ram0_data;
output [20:0] 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 [20:0] 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 [24:0] 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;
output  disp_data_out;
input  disp_data_in;

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;

//////////////////////////////////////////////////////////////////////////
//
```

```
   // Reset Generation
   //
   // A shift register primitive is used to generate an active-high reset
   // signal that remains high for 16 clock cycles after configuration finishes
   // and the FPGA's internal clocks begin toggling.
   //
   ////////////////////////////////////////////////////////////////////////////

   wire reset;
   SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(reset),
                   .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;


   ////////////////////////////////////////////////////////////////////////////
   //
   // reset wire
   //
   ////////////////////////////////////////////////////////////////////////////

   wire reset_sync;


   ////////////////////////////////////////////////////////////////////////////
   //
   // Alphanumeric displays
   //
   ////////////////////////////////////////////////////////////////////////////
   //
   // 0123456701234567
   // S:  M:  L:  C:
   //

   wire [39:0] system_dots, motion_dots, lasers_dots,
                       scale_dots, sounds_dots, access_dots,
                       change_combo_dots;
   wire [639:0] dots;

   assign dots = {        40'b00100110_01001001_01001001_01001001_00110010, // 'S'
                            system_dots,
                            40'b00000000_00000000_00000000_00000000_00000000, // ' '
                            40'b01111111_00000010_00000100_00000010_01111111, // 'M'
                            motion_dots,
                            40'b00000000_00000000_00000000_00000000_00000000, // ' '
                            40'b01111111_01000000_01000000_01000000_01000000, // 'L'
                            lasers_dots,
                                 40'b00000000_00000000_00000000_00000000_00000000, // ' '
                            40'b01111111_00001001_00001001_00001001_00000110, // 'P'
                            scale_dots,
                                 40'b00000000_00000000_00000000_00000000_00000000, // ' '
//
                                 40'b01111110_00001001_00001001_00001001_01111110, // 'A'
        //TRACK
                                 sounds_dots,
                                 change_combo_dots,
                            access_dots

                          };

   display disp (reset_sync, clock_27mhz, disp_blank, disp_clock, disp_rs,
                 disp_ce_b, disp_reset_b, disp_data_out, dots);

   ////////////////////////////////////////////////////////////////////////////
   //
   // PS/2 Interface Logic
   //
   ////////////////////////////////////////////////////////////////////////////


   wire system, motion, lasers, scale, sounds, access,
           reset_system, change_combo, first, second,
```

```
          third, fourth;


   wire[4:0] keypad_state;
   wire[7:0] keycode;
   wire[3:0] keypad_interface_state;
   wire break;
   wire[3:0] value;
   wire value_ready;
   wire[7:0] keycode_sync;

   keypad_function key_mod(keyboard_clock, clock_27mhz, reset_sync, keyboard_data,
system,
                           lasers, motion, scale, sounds,  access, reset_system,
change_combo,
                                   first, second, third, fourth, keypad_state, keycode,
                                   keypad_interface_state, keycode_ready,
keycode_ready_sync, break,
                                   value, value_ready);

   dot_analyzer dot_analy(system, motion, lasers, scale, sounds, access,
                                    change_combo, first, second, third, fourth,
                            system_dots, motion_dots, lasers_dots,
                                    scale_dots, sounds_dots, access_dots,
change_combo_dots);

        assign led[5] = ~system;
        assign led[4] = ~motion;
        assign led[3] = ~lasers;
        assign led[2] = ~scale;
        assign led[1] = ~sounds;
        assign led[0] = ~access;


   ////////////////////////////////////////////////////////////////////////
   //
   // Video I/O
   //
   ////////////////////////////////////////////////////////////////////////

   wire [1:0] videomode;
   wire alert, alert_reg;
   wire motion_pulse;

   debounce vmode0 (reset, clock_27mhz, switch[3], videomode[0]);
   debounce vmode1 (reset, clock_27mhz, switch[4], videomode[1]);

   video video_test (reset, clock_27mhz, 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[19:10],
                     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, videomode);

   image_analyzer image_analyzer(tv_in_line_clock1, reset_sync, tv_in_ycrcb[19:10],
motion_alarm
        , even, odd, difference, curr_image, hist_image,
        pixel, odd_line, even_line, state);

   PULLUP pu_in (.O(tv_in_i2c_data));
   PULLUP pu_out (.O(tv_out_i2c_data));


   assign led[7] = alert_reg ? 0 : 1;

   alert_sync alert_syncer(clock_27mhz, motion_alarm, motion_pulse, ~button0,
reset_sync);

   alert_register alert_register(clock_27mhz, reset_sync, alert_sync, alert_reg);
```

```
   ////////////////////////////////////////////////////////////////////////
   //
   // Audio Input and Output
   //
   ////////////////////////////////////////////////////////////////////////

   wire [3:0] vol;
   wire [39:0] vdisp;
   wire volume_up, volume_down;
   wire[19:0] right_in_data;
   wire ready;

   audio audio1 (reset, clock_27mhz, audio_reset_b, ac97_sdata_out,
                 ac97_sdata_in, ac97_synch, ac97_bit_clock, switch[1:0],
                 {vol, 1'b0}, switch[2], right_in_data, ready);

   beeper beep1 (reset, clock_27mhz, beep, ~button_enter);

   debounce vol_up (reset, clock_27mhz, button_up, volume_up);

   debounce vol_down (reset, clock_27mhz, button_down, volume_down);



   ////////////////////////////////////////////////////////////////////////
   //
   //  Sound Detector
   //
   ////////////////////////////////////////////////////////////////////////

   wire sounds_pulse;         //TRACK

   sound_detector sound_detector(clock_27mhz, reset_sync, right_in_data, ready,
sounds_pulse);

   assign user1[31] = sounds_pulse;

   ////////////////////////////////////////////////////////////////////////
   //
   // Top module
   //
   ////////////////////////////////////////////////////////////////////////

   wire alarm_signal;
   wire lasers_pulse;
   wire scale_pulse;

   top_module top_module(clock_27mhz, reset_sync, system, motion, lasers, scale, sounds,
access,
                                    motion_pulse, lasers_pulse, scale_pulse,
sounds_pulse, alarm_signal, reset_system);

   assign user1[0] = alarm_signal;
   //assign lasers_pulse = user1[1];
   //assign scale_pulse  = user1[2];
   assign lasers_pulse = ~button1;
   assign scale_pulse  = ~button2;
   assign analyzer1_data[0] = user1[1];

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

   // SRAMs
   assign ram0_data = 36'hZ;
   assign ram0_address = 21'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 = 21'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 = 15'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;

   // Buttons, Switches, and Individual LEDs
   assign led[6] = 2'b1;

   // User I/Os
   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;


   ////////////////////////////////////////////////////////////////////////
   //
   //   Analyzer connections
   //
   ////////////////////////////////////////////////////////////////////////




endmodule

////////////////////////////////////////////////////////////////////////////////
//
// Switch Debounce Module
//
////////////////////////////////////////////////////////////////////////////////

module debounce (reset, clock, noisy, clean);
```

```
   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 == 270000)
       clean <= new;
     else
       count <= count+1;

endmodule

////////////////////////////////////////////////////////////////////////////////
//
//      Synchronizer
//
//
////////////////////////////////////////////////////////////////////////////////


 module synchronizer(clk, reset, reset_tv);

        input clk;
        input reset;
        output reset_tv;

        reg reset_tv;
        reg temp1;

        always @ (posedge clk) begin
                temp1 <= reset;
                reset_tv <= temp1;
                end

endmodule


////////////////////////////////////////////////////////////////////////////
//
//              Alert Register
//
////////////////////////////////////////////////////////////////////////////

        module alert_sync(clk, alert, alert_sync, reset, reset_sync);
                input clk;
                input alert;
                input reset;
                output alert_sync;
                output reset_sync;

                reg alert_sync;
                reg reset_sync;
                reg temp1;
                reg temp2;

                always @ (posedge clk) begin
                        temp1 <= alert;
                        temp2 <= reset;
```

```
                        alert_sync <= temp1;
                        reset_sync <= temp2;
                        end
        endmodule

        module alert_register(clk, reset_sync, alert_sync, alert_reg);
                input clk;
                input alert_sync;
                input reset_sync;
                output alert_reg;

                reg alert_reg;

                always @ (posedge clk) begin
                        if(reset_sync) alert_reg <= 0;
                        else
                                if(alert_sync) alert_reg <= 1;
                end
        endmodule


//////////////////////////////////////////////////////////////////
//
//          PS/2 Interface and System Control
//          Author:    Christian Deonier
//          Modified:  April 19th, 2005
//
//////////////////////////////////////////////////////////////////



module keypad_function(keypad_clk, clk, reset, data, system, lasers, motion, scale,
sounds,
                                        access, reset_system, change_combo, first, second,
third, fourth
, keypad_state, keycode, keypad_interface_state, keycode_ready, keycode_ready_sync,
break,
value, value_ready, keycode_sync);
    input keypad_clk;
    input clk;
    input reset;
    input data;

    output system;
    output lasers;
    output motion;
    output scale;
    output sounds;                      //TRACK
    output access;
    output reset_system;
    output change_combo;
    output first;
    output second;
    output third;
    output fourth;

    output keycode_ready;
    output keycode_ready_sync;
    output[7:0] keycode;
    output[4:0] keypad_state;
    output[3:0] keypad_interface_state;
    output[3:0] value;
    output value_ready;
    output[7:0] keycode_sync;
    output break;

    wire[7:0] keycode;
    wire[3:0] value;
    wire keycode_ready;
    wire value_ready;
```

```
    wire keycode_ready_sync;
    wire keypad_clk_sync;
    wire data_sync;
    wire break;
    wire reset_system;
    wire change_combo;
    wire first;
    wire second;
    wire third;
    wire fourth;



keypad_fsm2 key_fsm(clk, reset, value, value_ready, system,
                            lasers, motion, scale, sounds, access,
                            keypad_state, reset_system, change_combo,
                            first, second, third, fourth);
keypad_interface key_int(keypad_clk_sync, reset, data_sync, keycode,
                                    keycode_ready, keypad_interface_state, break);
keypad_decoder key_dec(clk, reset, keycode_sync, keycode_ready_sync,
                                value, value_ready);
synchronizer_key sync_key(clk, keycode, keycode_ready, keycode_sync,
                                     keycode_ready_sync);
keypad_syncer keypad_syncer(clk, keypad_clk, keypad_clk_sync, data, data_sync);

endmodule

module keypad_syncer(clk, keypad_clk, keypad_clk_sync, data, data_sync);
        input clk;
        input keypad_clk;
        input data;
        output keypad_clk_sync;
        output data_sync;

        reg keypad_clk_sync;
        reg data_sync;
        reg temp1;
        reg temp2;

        always @ (posedge clk) begin
                temp1 <= data;
                temp2 <= keypad_clk;
                data_sync <= temp1;
                keypad_clk_sync <= temp2;
        end
endmodule



module synchronizer_key(clk, keycode, keycode_ready, keycode_sync, keycode_ready_sync);
        input clk;
        input[7:0] keycode;
        input keycode_ready;
        output[7:0] keycode_sync;
        output keycode_ready_sync;

        reg[7:0] temp1;
        reg temp2;
        reg[7:0] keycode_sync;
        reg keycode_ready_sync;

        always @ (posedge clk) begin
                temp1 <= keycode;
                temp2 <= keycode_ready;
                keycode_sync <= temp1;
                keycode_ready_sync <= temp2;
        end
endmodule

//////////////////////////////////////////////////////////////
//
```

```
//      Created By: Christian Deonier
//      May 12, 2005
//
////////////////////////////////////////////////////////////

module keypad_interface(clk, reset, data, keycode, keycode_ready, state, break_code);
    input clk;
    input reset;
    input data;
    output[7:0] keycode;
    output keycode_ready;
    output[3:0] state;

    output break_code;

    reg[7:0] keycode;
    reg[3:0] state;
    reg   keycode_ready;
    reg   break_code;

    parameter START       = 0;
    parameter DATA_ZERO   = 1;
    parameter DATA_ONE    = 2;
    parameter DATA_TWO        = 3;
    parameter DATA_THREE  = 4;
    parameter DATA_FOUR   = 5;
    parameter DATA_FIVE   = 6;
    parameter DATA_SIX        = 7;
    parameter DATA_SEVEN  = 8;
    parameter PARITY      = 9;
    parameter STOP        = 10;

    always @ (negedge clk or posedge reset)   begin
        if(reset)   begin
                    state <= START;
                    keycode_ready <= 0;
                    break_code <= 0;
                    keycode <= 0;
                    end
        else   begin
            case(state)
                START:      begin
                                    if(!data)   begin
                            state <= DATA_ZERO;
                                    end
                            keycode_ready <= 0;
                            end

                DATA_ZERO:  begin
                            state <= DATA_ONE;
                            keycode[0] <= data;
                            end

                DATA_ONE:   begin
                            state <= DATA_TWO;
                            keycode[1] <= data;
                            end

                DATA_TWO:   begin
                            state <= DATA_THREE;
                            keycode[2] <= data;
                            end

                DATA_THREE: begin
                            state <= DATA_FOUR;
                            keycode[3] <= data;
                            end

                DATA_FOUR:  begin
                            state <= DATA_FIVE;
                            keycode[4] <= data;
```

```
                            end

                DATA_FIVE:  begin
                            state <= DATA_SIX;
                            keycode[5] <= data;
                            end

                DATA_SIX:   begin
                            state <= DATA_SEVEN;
                            keycode[6] <= data;
                            end

                DATA_SEVEN: begin
                            state <= PARITY;
                            keycode[7] <= data;
                            end

                PARITY:     begin
                            state <= STOP;
                            if(!break_code)   begin
                                if(keycode == 8'hF0)   break_code <= 1;
                                else if(keycode == 8'h70 || keycode == 8'h69 || keycode
== 8'h7A ||
                                                        keycode == 8'h6B || keycode ==
8'h73 || keycode == 8'h74 ||
                                                        keycode == 8'h6C || keycode ==
8'h75 || keycode == 8'h7D ||
                                                        keycode == 8'h7B || keycode ==
8'h79 || keycode == 8'h72)   begin
                                                                   keycode_ready <= 1;
                                                                   end
                                                end
                                        else   break_code <= 0;
                            end

                STOP:       begin
                            state <= START;
                            keycode_ready <= 0;
                            end

                default:    state <= START;
            endcase
        end
    end

endmodule

//////////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
//////////////////////////////////////////////////////////////////////

module keypad_fsm2(clk, reset, value, value_ready, system,
                           lasers, motion, scale, sounds, access,
                           state, reset_system, change_combo, first,
                           second, third, fourth);
    input clk;
    input reset;
    input value_ready;
    input[3:0] value;

    output system;
    output lasers;
    output motion;
    output scale;
    output sounds;                  //TRACK
    output access;
    output reset_system;
    output change_combo;
```

```
output first;
output second;
output third;
output fourth;

output[4:0] state;


reg[3:0] keypad;
reg[4:0] state;
reg[3:0] first_digit;
reg[3:0] second_digit;
reg[3:0] third_digit;
reg[3:0] fourth_digit;
reg checked;
reg system;
reg motion;
reg lasers;
reg scale;
reg sounds;                  //TRACK
reg access;
reg reset_system;
reg change_combo;
reg first;
reg second;
reg third;
reg fourth;

parameter IDLE          = 0;
parameter COMBO_1       = 1;
parameter COMBO_2       = 2;
parameter COMBO_3       = 3;
parameter COMBO_SUCCESS = 4;
parameter SYSTEM_S      = 5;
parameter MOTION_S      = 6;
parameter LASERS_S      = 7;
parameter SCALE_S       = 8;
parameter SYSTEM_ON     = 9;
parameter SYSTEM_OFF    = 10;
parameter MOTION_ON     = 11;
parameter MOTION_OFF    = 12;
parameter LASERS_ON     = 13;
parameter LASERS_OFF    = 14;
parameter SCALE_ON      = 15;
parameter SCALE_OFF     = 16;
parameter SOUNDS_S    = 17;              //TRACK
parameter SOUNDS_ON          = 18;              //TRACK
parameter SOUNDS_OFF         = 19;              //TRACK
parameter CHANGE_COMBO1 = 20;            //TRACK2
parameter CHANGE_COMBO2 = 21;            //TRACK2
parameter CHANGE_COMBO3 = 22;            //TRACK2
parameter CHANGE_COMBO4 = 23;            //TRACK2

always @ (posedge clk)   begin
    if(reset)   begin
        state <= IDLE;
        first_digit    <= 6;
        second_digit   <= 1;
        third_digit    <= 1;
        fourth_digit   <= 1;
        system         <= 1;
        motion         <= 1;
        lasers         <= 1;
        scale          <= 1;
             sounds                <= 1;        //TRACK
        checked        <= 1;
        access         <= 0;
             change_combo   <= 0;
             first          <= 0;
             second                <= 0;
             third          <= 0;
```

```
                fourth                <= 0;
            end
        else if(value_ready)   begin
            keypad <= value;
            checked <= 0;
            end
        else   begin
            case(state)
                IDLE:   begin
                         access <= 0;
                                    change_combo <= 0;
                         if(!checked)   begin
                            checked <= 1;
                            if(keypad == first_digit)   state <= COMBO_1;
                            else                        state <= IDLE;
                         end
                         else                           state <= state;
                        end

                COMBO_1:   begin
                            if(!checked)   begin
                                checked <= 1;
                               if(keypad == second_digit)   state <= COMBO_2;
                               else                         state <= IDLE;
                            end
                            else                            state <= state;
                           end

                COMBO_2:   begin
                            if(!checked)   begin
                                checked <= 1;
                               if(keypad == third_digit)    state <= COMBO_3;
                               else                         state <= IDLE;
                            end
                            else                            state <= state;
                           end

                COMBO_3:   begin
                            if(!checked)   begin
                                checked <= 1;
                               if(keypad == fourth_digit)   state <= COMBO_SUCCESS;
                               else                         state <= IDLE;
                            end
                            else                            state <= state;
                           end

                COMBO_SUCCESS:   begin
                                   access <= 1;
                                   if(!checked)   begin
                                       checked <= 1;
                                       if(keypad == 1)        state <= SYSTEM_S;
                                       else if(keypad == 2)   state <= MOTION_S;
                                       else if(keypad == 3)   state <= LASERS_S;
                                       else if(keypad == 4)   state <= SCALE_S;
                                                              else if(keypad == 5)        state
<= SOUNDS_S;           //TRACK
                                                              else if(keypad == 6)        begin

state <= CHANGE_COMBO1;       //TRACK2

change_combo <= 1;

first <= 1;

end
                                       else if(keypad == 11)  state <= IDLE;
                                       else                   state <= state;
                                   end
                                   else                       state <= state;
                                  end
```

```
        SYSTEM_S:          begin
                              if(!checked)   begin
                                  checked <= 1;
                                  if(keypad == 10)       state <= SYSTEM_ON;
                                  else if(keypad == 11)  state <= SYSTEM_OFF;
                                  else                   state <= state;
                              end
                              else                       state <= state;
                           end

        MOTION_S:          begin
                              if(!checked)   begin
                                  checked <= 1;
                                  if(keypad == 10)       state <= MOTION_ON;
                                  else if(keypad == 11)  state <= MOTION_OFF;
                                  else                   state <= state;
                              end
                              else                       state <= state;
                           end

        LASERS_S:          begin
                              if(!checked)   begin
                                  checked <= 1;
                                  if(keypad == 10)       state <= LASERS_ON;
                                  else if(keypad == 11)  state <= LASERS_OFF;
                                  else                   state <= state;
                              end
                              else                       state <= state;
                           end

        SCALE_S:           begin

                              if(!checked)   begin
                                  checked <= 1;
                                  if(keypad == 10)       state <= SCALE_ON;
                                  else if(keypad == 11)  state <= SCALE_OFF;
                                  else                   state <= state;
                              end
                              else                       state <= state;
                           end

        SYSTEM_ON:         begin
                           state <= COMBO_SUCCESS;
                           system <= 1;
                           motion <= 1;
                           lasers <= 1;
                           scale  <= 1;
                                           sounds <= 1;
//TRACK
                                           reset_system <= 1;
                           end

        SYSTEM_OFF:        begin
                           state <= COMBO_SUCCESS;
                           system <= 0;
                           motion <= 0;
                           lasers <= 0;
                           scale  <= 0;
                                           sounds <= 0;
//TRACK
                           end

        MOTION_ON:         begin
                           state <= COMBO_SUCCESS;
                           motion <= 1;
                                           reset_system <= 1;
                           end

        MOTION_OFF:        begin
```

```
                        state <= COMBO_SUCCESS;
                        motion <= 0;
                        end

        LASERS_ON:      begin
                        state <= COMBO_SUCCESS;
                        lasers <= 1;
                                        reset_system <= 1;
                        end

        LASERS_OFF:     begin
                        state <= COMBO_SUCCESS;
                        lasers <= 0;
                        end

        SCALE_ON:       begin
                        state <= COMBO_SUCCESS;
                        scale <= 1;
                                        reset_system <= 1;
                        end

        SCALE_OFF:      begin
                        state <= COMBO_SUCCESS;
                        scale <= 0;
                        end

                SOUNDS_S:           begin
                        //TRACK
                                                if(!checked)   begin
                                checked <= 1;
                                if(keypad == 10)       state <= SOUNDS_ON;
                                else if(keypad == 11)  state <= SOUNDS_OFF;
                                else                   state <= state;
                            end
                            else                       state <= state;
                                        end

                SOUNDS_ON:          begin
                                        state <= COMBO_SUCCESS;
                                        sounds <= 1;
                                        reset_system <= 1;
                                        end

                SOUNDS_OFF:         begin
                                        state <= COMBO_SUCCESS;
                                        sounds <= 0;
                                        end

                CHANGE_COMBO1:      begin
                                            if(!checked)   begin
                                checked <= 1;
                                if(keypad == 1)        begin

state <= CHANGE_COMBO2;

first_digit <= 1;

first  <= 0;

second <= 1;

end
                                                    else if(keypad == 2) begin

state <= CHANGE_COMBO2;

first_digit <= 2;

first  <= 0;

second <= 1;
```

```
end
                                                    else if(keypad == 3) begin
state <= CHANGE_COMBO2;
first_digit <= 3;
first  <= 0;
second <= 1;
end
                                                    else if(keypad == 4) begin
state <= CHANGE_COMBO2;
first_digit <= 4;
first  <= 0;
second <= 1;
end
                                                    else if(keypad == 5) begin
state <= CHANGE_COMBO2;
first_digit <= 5;
first  <= 0;
second <= 1;
end
                                                    else if(keypad == 6) begin
state <= CHANGE_COMBO2;
first_digit <= 6;
first  <= 0;
second <= 1;
end
                                                    else if(keypad == 7) begin
state <= CHANGE_COMBO2;
first_digit <= 7;
first  <= 0;
second <= 1;
end
                                                    else if(keypad == 8) begin
state <= CHANGE_COMBO2;
first_digit <= 8;
first  <= 0;
second <= 1;
end
                                                    else if(keypad == 9) begin
state <= CHANGE_COMBO2;
```

```
first_digit <= 9;

first  <= 0;

second <= 1;

end
                                                        else if(keypad == 0) begin

state <= CHANGE_COMBO2;

first_digit <= 0;

first  <= 0;

second <= 1;

end
                                    else                    state <= state;
                                end
                                else                        state <= state;
                                                end

                CHANGE_COMBO2:        begin
                                                    if(!checked)   begin
                                    checked <= 1;
                                    if(keypad == 1)         begin

state <= CHANGE_COMBO3;

second_digit <= 1;

second <= 0;

third  <= 1;

end
                                                        else if(keypad == 2) begin

state <= CHANGE_COMBO3;

second_digit <= 2;

second <= 0;

third  <= 1;

end
                                                        else if(keypad == 3) begin

state <= CHANGE_COMBO3;

second_digit <= 3;

second <= 0;

third  <= 1;

end
                                                        else if(keypad == 4) begin

state <= CHANGE_COMBO3;

second_digit <= 4;

second <= 0;

third  <= 1;

end
```

```
                                                    else if(keypad == 5) begin
state <= CHANGE_COMBO3;
second_digit <= 5;
second <= 0;
third  <= 1;
end
                                                    else if(keypad == 6) begin
state <= CHANGE_COMBO3;
second_digit <= 6;
second <= 0;
third  <= 1;
end
                                                    else if(keypad == 7) begin
state <= CHANGE_COMBO3;
second_digit <= 7;
second <= 0;
third  <= 1;
end
                                                    else if(keypad == 8) begin
state <= CHANGE_COMBO3;
second_digit <= 8;
second <= 0;
third  <= 1;
end
                                                    else if(keypad == 9) begin
state <= CHANGE_COMBO3;
second_digit <= 9;
second <= 0;
third  <= 1;
end
                                                    else if(keypad == 0) begin
state <= CHANGE_COMBO3;
second_digit <= 0;
second <= 0;
third  <= 1;
end
                                    else                    state <= state;
                                end
                                else                        state <= state;
                                            end
```

```
                CHANGE_COMBO3:        begin
                                                if(!checked)   begin
                                      checked <= 1;
                                      if(keypad == 1)        begin

state <= CHANGE_COMBO4;

third_digit <= 1;

third  <= 0;

fourth <= 1;

end
                                                                  else if(keypad == 2) begin

state <= CHANGE_COMBO4;

third_digit <= 2;

third  <= 0;

fourth <= 1;

end
                                                                  else if(keypad == 3) begin

state <= CHANGE_COMBO4;

third_digit <= 3;

third  <= 0;

fourth <= 1;

end
                                                                  else if(keypad == 4) begin

state <= CHANGE_COMBO4;

third_digit <= 4;

third  <= 0;

fourth <= 1;

end
                                                                  else if(keypad == 5) begin

state <= CHANGE_COMBO4;

third_digit <= 5;

third  <= 0;

fourth <= 1;

end
                                                                  else if(keypad == 6) begin

state <= CHANGE_COMBO4;

third_digit <= 6;

third  <= 0;

fourth <= 1;

end
                                                                  else if(keypad == 7) begin
```

```
state <= CHANGE_COMBO4;

third_digit <= 7;

third  <= 0;

fourth <= 1;

end
                                                    else if(keypad == 8) begin

state <= CHANGE_COMBO4;

third_digit <= 8;

third  <= 0;

fourth <= 1;

end
                                                    else if(keypad == 9) begin

state <= CHANGE_COMBO4;

third_digit <= 9;

third  <= 0;

fourth <= 1;

end
                                                    else if(keypad == 0) begin

state <= CHANGE_COMBO4;

third_digit <= 0;

third  <= 0;

fourth <= 1;

end
                                    else                    state <= state;
                                end
                                else                        state <= state;
                                                end

                CHANGE_COMBO4:        begin
                                                if(!checked)   begin
                                            checked <= 1;
                                            if(keypad == 1)        begin

state <= COMBO_SUCCESS;

fourth_digit <= 1;

fourth <= 0;

change_combo <= 0;

end
                                                    else if(keypad == 2) begin

state <= COMBO_SUCCESS;

fourth_digit <= 2;

fourth <= 0;

change_combo <= 0;
```

```
end
                                                    else if(keypad == 3) begin
state <= COMBO_SUCCESS;
fourth_digit <= 3;
fourth <= 0;
change_combo <= 0;
end
                                                    else if(keypad == 4) begin
state <= COMBO_SUCCESS;
fourth_digit <= 4;
fourth <= 0;
change_combo <= 0;
end
                                                    else if(keypad == 5) begin
state <= COMBO_SUCCESS;
fourth_digit <= 5;
fourth <= 0;
change_combo <= 0;
end
                                                    else if(keypad == 6) begin
state <= COMBO_SUCCESS;
fourth_digit <= 6;
fourth <= 0;
change_combo <= 0;
end
                                                    else if(keypad == 7) begin
state <= COMBO_SUCCESS;
fourth_digit <= 7;
fourth <= 0;
change_combo <= 0;
end
                                                    else if(keypad == 8) begin
state <= COMBO_SUCCESS;
fourth_digit <= 8;
fourth <= 0;
change_combo <= 0;
end
                                                    else if(keypad == 9) begin
state <= COMBO_SUCCESS;
```

```
        fourth_digit <= 9;

        fourth <= 0;

        change_combo <= 0;

        end
                                                                  else if(keypad == 0) begin

        state <= COMBO_SUCCESS;

        fourth_digit <= 0;

        fourth <= 0;

        change_combo <= 0;

        end
                                        else                    state <= state;
                                    end
                                    else                        state <= state;
                                                end


                default:           state <= IDLE;
           endcase
        end
   end
endmodule


//////////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
//////////////////////////////////////////////////////////////////////

module keypad_decoder(clk, reset, keycode, keycode_ready, value, value_ready);
    input clk;
    input reset;
    input keycode_ready;
    input[7:0] keycode;
    output[3:0] value;
    output value_ready;

    reg[7:0] stored_keycode;
    reg[3:0] value;//, value_int;
    reg value_ready;

    always @ (posedge clk)  begin
        if(reset)   begin
                    stored_keycode <= 0;
                    value_ready <= 0;
                    end
        else if(!keycode_ready)     begin
                                                value_ready <= 0;
                                                end
        else if(keycode_ready)   begin
                                 value_ready <= 1;
                                 stored_keycode <= keycode;
                                 end
        else                     stored_keycode <= 8'hzz;
    end

    //8'h79 = +  stored as value 10
    //8'h7B = -  stored as value 11
    //unrecognised stored as value 12

    always @ (stored_keycode)   begin
```

```
        if(stored_keycode == 8'h70)         value = 0;
        else if(stored_keycode == 8'h69)    value = 1;
        else if(stored_keycode == 8'h72)    value = 2;
        else if(stored_keycode == 8'h7A)    value = 3;
        else if(stored_keycode == 8'h6B)    value = 4;
        else if(stored_keycode == 8'h73)    value = 5;
        else if(stored_keycode == 8'h74)    value = 6;
        else if(stored_keycode == 8'h6C)    value = 7;
        else if(stored_keycode == 8'h75)    value = 8;
        else if(stored_keycode == 8'h7D)    value = 9;
        else if(stored_keycode == 8'h79)    value = 10;
        else if(stored_keycode == 8'h7B)    value = 11;
        else                                value = 4'hz;
    end
endmodule


//////////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
//////////////////////////////////////////////////////////////////////

module dot_analyzer(system, motion, lasers, scale, sounds, access,
                            change_combo, first, second, third, fourth,
                    system_dots, motion_dots, lasers_dots, scale_dots,
                            sounds_dots, access_dots, change_combo_dots);
        input system;
        input motion;
        input lasers;
        input scale;
          input sounds;                             //TRACK
        input access;
          input first;
          input second;
          input third;
          input fourth;
          input change_combo;

        output[39:0] system_dots;
        output[39:0] motion_dots;
        output[39:0] lasers_dots;
        output[39:0] scale_dots;
          output[39:0] sounds_dots;         //TRACK
        output[39:0] access_dots;
          output[39:0] change_combo_dots;

        reg[39:0] system_dots;
        reg[39:0] motion_dots;
        reg[39:0] lasers_dots;
        reg[39:0] scale_dots;
          reg[39:0] sounds_dots;            //TRACK
        reg[39:0] access_dots;
          reg[39:0] change_combo_dots;

        always @ (system or motion or lasers or scale or sounds)   begin
                 system_dots = 40'b00000000_00000000_00000000_00000000_00000000;
                 motion_dots = 40'b00000000_00000000_00000000_00000000_00000000;
                 scale_dots  = 40'b00000000_00000000_00000000_00000000_00000000;
                 lasers_dots = 40'b00000000_00000000_00000000_00000000_00000000;
                 sounds_dots = 40'b00000000_00000000_00000000_00000000_00000000;
                 access_dots = 40'b00000000_00000000_00000000_00000000_00000000;
                 change_combo_dots = 40'b00000000_00000000_00000000_00000000_00000000;

            if(system)   system_dots = 40'b00111110_01000001_01000001_01000001_00111110;
            else         system_dots = 40'b01111111_00001001_00001001_00000001_00000001;

            if(motion)   motion_dots = 40'b00111110_01000001_01000001_01000001_00111110;
            else         motion_dots = 40'b01111111_00001001_00001001_00000001_00000001;
```

```
            if(scale)    scale_dots =  40'b00111110_01000001_01000001_01000001_00111110;
            else         scale_dots =  40'b01111111_00001001_00001001_00000001_00000001;

            if(lasers)   lasers_dots = 40'b00111110_01000001_01000001_01000001_00111110;
            else         lasers_dots = 40'b01111111_00001001_00001001_00000001_00000001;

                 if(sounds)   sounds_dots =
40'b00111110_01000001_01000001_01000001_00111110;    //TRACK
            else         sounds_dots = 40'b01111111_00001001_00001001_00000001_00000001;
       //TRACK

                 if(change_combo)     change_combo_dots =
40'b00000000_00000000_01011111_00000000_00000000;
                 else                 change_combo_dots =
40'b00000000_00000000_00000000_00000000_00000000;

            //+ = access
                 if(access && first)         access_dots =
40'b00000000_01000010_01111111_01000000_00000000; //1
                 else if(access && second)   access_dots =
40'b01100010_01010001_01001001_01001001_01000110; //2
                 else if(access && third)    access_dots =
40'b00100010_01000001_01001001_01001001_00110110; //3
                 else if(access && fourth)   access_dots =
40'b00011000_00010100_00010010_01111111_00010000; //4
            else if(access)                  access_dots =
40'b00001000_00001000_00111110_00001000_00001000;
            else         access_dots = 40'b00100100_00010100_00001111_00010100_00100100;

        end
endmodule


//////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
//////////////////////////////////////////////////////////////////

module top_module(clk, reset, system, motion, lasers, scale, sounds, access,
motion_pulse,
                  lasers_pulse, scale_pulse, sounds_pulse, alarm_signal, reset_system);

        input clk;
        input reset;
        input system;
        input motion;
        input lasers;
        input scale;
          input sounds;                      //TRACK
        input access;
        input motion_pulse;
        input lasers_pulse;
        input scale_pulse;
          input sounds_pulse;        //TRACK
          input reset_system;

        output alarm_signal;

        wire lasers_pulse_sync;
        wire scale_pulse_sync;
        wire motion_alert_reg;
        wire lasers_alert_reg;
        wire scale_alert_reg;
          wire sounds_alert_reg;     //TRACK
        wire motion_reset;
        wire lasers_reset;
        wire scale_reset;
          wire sounds_reset;         //TRACK
        wire alert;
```

```
        wire alert_reset;
        wire alarm_reset;

major_fsm major_fsm(clk, reset, system, motion, lasers, scale, sounds, access,
                    motion_alert_reg, lasers_alert_reg, scale_alert_reg,
                              sounds_alert_reg, motion_reset, lasers_reset, scale_reset,
                              sounds_reset, alert, alert_reset);

component_registers com_reg(clk, motion_pulse, lasers_pulse_sync, scale_pulse_sync,
sounds_pulse,
                            motion_reset, lasers_reset, scale_reset,
sounds_reset,motion_alert_reg,
                                        lasers_alert_reg, scale_alert_reg,
sounds_alert_reg, reset_system);

top_synchronizer top_sync(clk, lasers_pulse, scale_pulse, lasers_pulse_sync,
scale_pulse_sync);

alarm alarm(clk, alarm_reset, alarm_signal);

alarm_reg alarm_reg(clk, alert_reset, alert, alarm_reset);

endmodule

//////////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
//////////////////////////////////////////////////////////////////////

module top_synchronizer(clk, lasers_pulse, scale_pulse, lasers_pulse_sync,
scale_pulse_sync);
    input lasers_pulse;
    input scale_pulse;
    input clk;

    output lasers_pulse_sync;
    output scale_pulse_sync;

    reg lasers_pulse_sync;
    reg scale_pulse_sync;
    reg temp1;
    reg temp2;

    always @ (posedge clk)   begin
        temp1 <= lasers_pulse;
        temp2 <= scale_pulse;
        lasers_pulse_sync <= temp1;
        scale_pulse_sync  <= temp2;
        end
endmodule


//////////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
//////////////////////////////////////////////////////////////////////

module top_synchronizer(clk, lasers_pulse, scale_pulse, lasers_pulse_sync,
scale_pulse_sync);
    input lasers_pulse;
    input scale_pulse;
    input clk;

    output lasers_pulse_sync;
    output scale_pulse_sync;

    reg lasers_pulse_sync;
```

```
    reg scale_pulse_sync;
    reg temp1;
    reg temp2;

    always @ (posedge clk)   begin
        temp1 <= lasers_pulse;
        temp2 <= scale_pulse;
        lasers_pulse_sync <= temp1;
        scale_pulse_sync  <= temp2;
        end
endmodule


module major_fsm(clk, reset, system, motion, lasers, scale, sounds, access,
                motion_alert_reg, lasers_alert_reg, scale_alert_reg,
                        sounds_alert_reg, motion_reset, lasers_reset,scale_reset,
                        sounds_reset, alert, alert_reset);

    input clk;
    input reset;
    input system;
    input motion;
    input lasers;
    input scale;
    input sounds;                               //TRACK
    input access;
    input motion_alert_reg;
    input lasers_alert_reg;
    input scale_alert_reg;
    input sounds_alert_reg;          //TRACK

    output motion_reset;
    output lasers_reset;
    output scale_reset;
    output sounds_reset;                        //TRACK
    output alert;
    output alert_reset;

    reg motion_reset;
    reg lasers_reset;
    reg scale_reset;
    reg sounds_reset;                //TRACK
    reg alert;
    reg alert_reset;
    reg[2:0] state;

    parameter IDLE         = 0;
    parameter CHECK_MOTION = 1;
    parameter CHECK_LASERS = 2;
    parameter CHECK_SCALE  = 3;
    parameter CHECK_SOUNDS = 4;       //TRACK
    parameter TRIGGER      = 5;       //TRACK
    parameter ALERT        = 6;       //TRACK
    parameter DEACTIVATE   = 7;       //TRACK


    always @ (posedge clk or posedge reset)   begin
        if(reset)   begin
            state        <= IDLE;
            alert_reset  <= 1;
            motion_reset <= 1;
            lasers_reset <= 1;
            scale_reset  <= 1;
                 sounds_reset <= 1;                                    //TRACK
        end
        else   begin
            case(state)
                IDLE:   begin
                        alert_reset  <= 0;
                        motion_reset <= 0;
                        lasers_reset <= 0;
```

```
                        scale_reset  <= 0;
                                  sounds_reset <= 0;                //TRACK
                        state        <= CHECK_MOTION;
                        end

                CHECK_MOTION:   begin
                                if(motion_alert_reg && motion)   state <= TRIGGER;
                                else                             state <= CHECK_LASERS;
                                end

                CHECK_LASERS:   begin
                                if(lasers_alert_reg && lasers)   state <= TRIGGER;
                                else                             state <= CHECK_SCALE;
                                end

                CHECK_SCALE:    begin
                                if(scale_alert_reg && scale)     state <= TRIGGER;
                                else                             state <= CHECK_SOUNDS;
                                end

                        CHECK_SOUNDS:   begin
                                      //TRACK
                                                if(sounds_alert_reg && sounds)      state
<= TRIGGER;            //TRACK
                                                else
        state <= CHECK_MOTION; //TRACK
                                                end
                                           //TRACK

                TRIGGER:        begin
                                alert <= 1;
                                state <= ALERT;
                                end

                ALERT:          begin
                                alert <= 0;
                                if(access && !system)            state <= DEACTIVATE;
                                else                             state <= ALERT;
                                end

                DEACTIVATE:     begin
                                alert_reset  <= 1;
                                motion_reset <= 1;
                                lasers_reset <= 1;
                                scale_reset  <= 1;
                                              sounds_reset <= 1;                  //TRACK
                                state        <= IDLE;
                                end

                default:        state <= IDLE;
            endcase
         end
     end
endmodule



//////////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
//////////////////////////////////////////////////////////////////////

module sound_detector(clk, reset_sync, right_in_data, ready, alert);
   input clk;
   input reset_sync;
   input[19:0] right_in_data;
   input ready;
   output alert;
```

```
   reg alert;

   parameter SENSITIVITY = 8'h11;

   always @ (posedge clk)     begin
        if(reset_sync) begin
               alert <= 0;
               end
        else   begin
               if(right_in_data[19:16] == 4'hF)      begin
                      alert <= alert;
                      end
               else   begin
                      if(right_in_data[15:8] > SENSITIVITY && ready)      begin
                             alert <= 1;
                             end
                      else   begin
                             alert <= 0;
                             end
                      end
               end
        end
endmodule


//////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
//////////////////////////////////////////////////////////////////

module image_analyzer(clk, reset, data, alert
, even, odd, difference, curr_image, hist_image,
pixel, odd_line, even_line, state);
    input clk;
    input reset;
    input[9:0] data;        //multiplexed 10-bit pixel data
    output alert;

    output even, odd;
    output[31:0] difference, curr_image, hist_image;
    output[9:0] pixel;
    output[7:0] odd_line, even_line;
    output[3:0] state;

    reg alert;              //alert to system
    reg even;               //keeps track if we finished analyzing even lines
    reg odd;                //keeps track if we finished analyzing odd lines
    reg[31:0] difference;   //keeps track of luminance difference between historical and
current image
    reg[31:0] curr_image;   //keeps track of current image's luminance
    reg[31:0] hist_image;   //keeps track of historical image's luminance
    reg[9:0] pixel;         //keeps track of amount of pixels per line
    reg[7:0] odd_line;      //keeps track of number odd lines analyzed
    reg[7:0] even_line;     //keeps track of number even lines analyzed
    reg[3:0] state;

    parameter SENSITIVITY = 32'h00180000;   //smaller = less sensitive

    parameter IDLE       = 0;
    parameter TRS1       = 1;
    parameter TRS2       = 2;
    parameter OK         = 3;
    parameter ODD_SKIP   = 4;
    parameter ODD        = 5;
    parameter EVEN_SKIP  = 6;
    parameter EVEN       = 7;
    parameter COMPARE    = 8;

    always @ (posedge clk)
```

```
        if(reset)   begin
                    state      <= IDLE;
                    alert      <= 0;
                    even       <= 0;
                    odd        <= 0;
                    curr_image <= 0;
                    hist_image <= 0;
                    pixel      <= 0;
                    odd_line   <= 0;
                    even_line  <= 0;
                    end
        else   begin
            case(state)
                IDLE:   begin
                        if(data == 10'h3FF)      state <= TRS1;
                        else if(even && odd)   state <= COMPARE;
                        else                   state <= IDLE;
                        pixel <= 0;
                        alert <= 0;
                        //Update difference
                        if(hist_image > curr_image)  difference <= hist_image -
curr_image;
                                    else
        difference <= curr_image - hist_image;
                        end

                TRS1:   begin
                        if(data == 10'h000)      state <= TRS2;
                        else                   state <= IDLE;
                        end

                TRS2:   begin
                        if(data == 10'h000)      state <= OK;
                        else                   state <= IDLE;
                        end

                OK:     begin
                        if(data[8] && !data[7])         begin
                                                        state <= EVEN_SKIP;
                                                        even_line <= even_line + 1;
                                                        end
                        else if(!data[8] && !data[7])   begin
                                                        state <= ODD_SKIP;
                                                        odd_line <= odd_line + 1;
                                                        end
                        else                            state <= IDLE;
                        end

                ODD_SKIP:     begin
                              state <= ODD;
                              pixel <= pixel + 1;
                              if(odd_line == 254)         odd <= 1;
                              end

                ODD:          begin
                              if(pixel == 720)   state <= IDLE;
                              else               state <= ODD_SKIP;
                              curr_image <= curr_image + data;
                              end

                EVEN_SKIP:    begin
                              state <= EVEN;
                              pixel <= pixel + 1;
                              if(even_line == 253)       even <= 1;
                              end

                EVEN:         begin
                              if(pixel == 720)   state <= IDLE;
                              else               state <= EVEN_SKIP;
                              curr_image <= curr_image + data;
                              end
```

```
                COMPARE:      begin
                              state       <= IDLE;
                              even        <= 0;
                              odd         <= 0;
                              even_line   <= 0;
                              odd_line    <= 0;
                              hist_image  <= curr_image;
                              curr_image  <= 0;
                              //Compare luminance
                              if(hist_image != 0)
                                 if(difference > SENSITIVITY)      alert <= 1;
                              end
            endcase
         end
endmodule



////////////////////////////////////////////////////////////////////
//
//      Created By:  Christian Deonier
//      May 12, 2005
//
////////////////////////////////////////////////////////////////////

module component_registers(clk, motion_pulse, lasers_pulse_sync, scale_pulse_sync,
sounds_pulse,
                           motion_reset, lasers_reset, scale_reset, sounds_reset,
                           motion_alert_reg, lasers_alert_reg, scale_alert_reg,
sounds_alert_reg,
                                      reset_system);

         input clk;
         input motion_pulse;
         input lasers_pulse_sync;
         input scale_pulse_sync;
           input sounds_pulse;               //TRACK
         input motion_reset;
         input lasers_reset;
         input scale_reset;
           input reset_system;
           input sounds_reset;                       //TRACK

         output motion_alert_reg;
         output lasers_alert_reg;
         output scale_alert_reg;
           output sounds_alert_reg;

         reg motion_alert_reg;
         reg lasers_alert_reg;
         reg scale_alert_reg;
           reg sounds_alert_reg;

         always @ (posedge clk)   begin
             if(motion_reset || reset_system)   motion_alert_reg <= 0;
             if(lasers_reset || reset_system)   lasers_alert_reg <= 0;
             if(scale_reset  || reset_system)   scale_alert_reg  <= 0;
                  if(sounds_reset || reset_system)     sounds_alert_reg <= 0;
        //TRACK
             if(motion_pulse)        motion_alert_reg <= 1;
             if(lasers_pulse_sync)   lasers_alert_reg <= 1;
             if(scale_pulse_sync)    scale_alert_reg  <= 1;
                  if(sounds_pulse)             sounds_alert_reg <= 1;
        //TRACK
         end
endmodule

//Nathan Ickes's Code

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);
```

```
input reset;
input clock4x;
input [7:0] data;
input load;
output ack;
output idle;
output scl;
output sda;

reg [7:0] ldata;
reg ack, idle;
reg scl;
reg sdai;

reg [7:0] state;

assign sda = sdai ? 1'bZ : 1'b0;

always @(posedge clock4x)
  if (reset)
    begin
       state <= 0;
       ack <= 0;
    end
  else
    case (state)
      8'h00: // idle
        begin
           scl <= 1'b1;
           sdai <= 1'b1;
           ack <= 1'b0;
           idle <= 1'b1;
           if (load)
             begin
                ldata <= data;
                ack <= 1'b1;
                state <= state+1;
             end
        end
      8'h01: // Start
        begin
           ack <= 1'b0;
           idle <= 1'b0;
           sdai <= 1'b0;
           state <= state+1;
        end
      8'h02:
        begin
           scl <= 1'b0;
           state <= state+1;
        end
      8'h03: // Send bit 7
        begin
           ack <= 1'b0;
           sdai <= ldata[7];
           state <= state+1;
        end
      8'h04:
        begin
           scl <= 1'b1;
           state <= state+1;
        end
      8'h05:
        begin
           state <= state+1;
        end
      8'h06:
        begin
           scl <= 1'b0;
           state <= state+1;
```

```
  end
8'h07:
  begin
     sdai <= ldata[6];
     state <= state+1;
  end
8'h08:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h09:
  begin
     state <= state+1;
  end
8'h0A:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h0B:
  begin
     sdai <= ldata[5];
     state <= state+1;
  end
8'h0C:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h0D:
  begin
     state <= state+1;
  end
8'h0E:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h0F:
  begin
     sdai <= ldata[4];
     state <= state+1;
  end
8'h10:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h11:
  begin
     state <= state+1;
  end
8'h12:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h13:
  begin
     sdai <= ldata[3];
     state <= state+1;
  end
8'h14:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h15:
  begin
     state <= state+1;
```

```
  end
8'h16:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h17:
  begin
     sdai <= ldata[2];
     state <= state+1;
  end
8'h18:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h19:
  begin
     state <= state+1;
  end
8'h1A:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h1B:
  begin
     sdai <= ldata[1];
     state <= state+1;
  end
8'h1C:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h1D:
  begin
     state <= state+1;
  end
8'h1E:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h1F:
  begin
     sdai <= ldata[0];
     state <= state+1;
  end
8'h20:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h21:
  begin
     state <= state+1;
  end
8'h22:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h23: // Acknowledge bit
  begin
     state <= state+1;
  end
8'h24:
  begin
     scl <= 1'b1;
     state <= state+1;
```

```
          end
        8'h25:
          begin
             state <= state+1;
          end
        8'h26:
          begin
             scl <= 1'b0;
             if (load)
              begin
                 ldata <= data;
                 ack <= 1'b1;
                 state <= 3;
              end
             else
              state <= state+1;
          end
        8'h27:
          begin
             sdai <= 1'b0;
             state <= state+1;
          end
        8'h28:
          begin
             scl <= 1'b1;
             state <= state+1;
          end
        8'h29:
          begin
             sdai <= 1'b1;
             state <= 0;
          end
       endcase

endmodule


//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Video (TV) Test Code
//
// For Labkit Revision 004
//
//
// Created: November 3, 2004
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////

//`include "i2c.v"
//`include "adv7194init.v"
//`include "adv7185init.v"

module video (reset, clock_27mhz, 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, mode);

   input reset;
   input clock_27mhz;

   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 [9:0] 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_i2c_data, tv_in_fifo_read, tv_in_fifo_clock,
          tv_in_iso, tv_in_reset_b, tv_in_clock;
   input [1:0] mode;

   // Mode Decoding
   //
   // 0 = colorbars, 1 = MIT logo, 2 = composite passthrough,
   // 3 = s-video passthrough

   wire colorbars, logo, svideo;
   assign colorbars = (mode == 0);
   assign logo = (mode == 1);
   assign svideo = (mode == 3);

   //
   // Logo generator
   //

   wire [9:0] logo_ycrcb;
   //videologo logo1 (reset, clock_27mhz, logo_ycrcb);

   //
   // ADV7194 (Output)
   //

   assign tv_out_clock = (logo||colorbars) ? clock_27mhz : tv_in_line_clock1;
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_vsync_b = 0;
   assign tv_out_hsync_b = 0;
   assign tv_out_subcar_reset = 0;
   assign tv_out_blank_b = 0;

   adv7194init init7194 (reset, clock_27mhz, colorbars,
                         tv_out_reset_b, tv_out_i2c_clock, tv_out_i2c_data);

   assign tv_out_ycrcb = tv_in_ycrcb;

   //
   // ADV7185 (Input)
   //

   assign tv_in_fifo_read = 1'b1;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b1;
   assign tv_in_clock = clock_27mhz;

   adv7185init init7185 (reset, clock_27mhz, svideo, tv_in_reset_b,
                         tv_in_i2c_clock, tv_in_i2c_data);

endmodule


//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration
//
// Created:
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
// Register 0
//////////////////////////////////////////////////////////////////////////////////

`define INPUT_SELECT                            4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                              4'h0
```

```
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

////////////////////////////////////////////////////////////////////////////////
// Register 1
////////////////////////////////////////////////////////////////////////////////

`define VIDEO_QUALITY                           2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                    1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT                      1'b0
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING                     1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                                 1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE                1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM,
`FOUR_TIMES_SAMPLING, `DIFFERENTIAL_INPUT, `SQUARE_PIXEL_IN_MODE, `VIDEO_QUALITY}

////////////////////////////////////////////////////////////////////////////////
// Register 2
////////////////////////////////////////////////////////////////////////////////

`define Y_PEAKING_FILTER                        3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                  2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

////////////////////////////////////////////////////////////////////////////////
// Register 3
////////////////////////////////////////////////////////////////////////////////
```

```
`define INTERFACE_SELECT                        2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                           4'h0      //change  *************
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS                 1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                              1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS, `OUTPUT_FORMAT,
`INTERFACE_SELECT}

////////////////////////////////////////////////////////////////////////////////
// Register 4
////////////////////////////////////////////////////////////////////////////////

`define OUTPUT_DATA_RANGE                       1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                              1'b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110, `OUTPUT_DATA_RANGE}

////////////////////////////////////////////////////////////////////////////////
// Register 5
////////////////////////////////////////////////////////////////////////////////


`define GENERAL_PURPOSE_OUTPUTS                 4'b0000
`define GPO_0_1_ENABLE                          1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                          1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI                     1'b1                        //changed
**********
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                            1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI, `GPO_2_3_ENABLE,
`GPO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}

////////////////////////////////////////////////////////////////////////////////
// Register 7
////////////////////////////////////////////////////////////////////////////////

`define FIFO_FLAG_MARGIN                        5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                              1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET                    1'b0
  // 0: No automatic reset
```

```
  // 1: FIFO is autmatically reset at the end of each video field
`define FIFO_FLAG_SELF_TIME                     1'b1
  // 0: FIFO flags are synchronized to CLKIN
  // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET, `FIFO_RESET,
`FIFO_FLAG_MARGIN}

////////////////////////////////////////////////////////////////////////////
// Register 8
////////////////////////////////////////////////////////////////////////////

`define INPUT_CONTRAST_ADJUST                       8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register 9
////////////////////////////////////////////////////////////////////////////

`define INPUT_SATURATION_ADJUST                     8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register A
////////////////////////////////////////////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST                     8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register B
////////////////////////////////////////////////////////////////////////////

`define INPUT_HUE_ADJUST                            8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register C
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_VALUE_ENABLE                    1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE          1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                         6'h0C
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE, `DEFAULT_VALUE_AUTOMATIC_ENABLE,
`DEFAULT_VALUE_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register D
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_CR_VALUE                        4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                        4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////////
// Register E
////////////////////////////////////////////////////////////////////////////
```

```
`define TEMPORAL_DECIMATION_ENABLE              1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL             2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE                4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE,
`TEMPORAL_DECIMATION_CONTROL, `TEMPORAL_DECIMATION_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register F
////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL                      2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY              1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE                    1'b0
  // 0: Reference is functional
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR                1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                         1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                        1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                              1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP,
`POWER_DOWN_LLC_GENERATOR, `POWER_DOWN_REFERENCE, `POWER_DOWN_SOURCE_PRIORITY,
`POWER_SAVE_CONTROL}

////////////////////////////////////////////////////////////////////////////
// Register 33
////////////////////////////////////////////////////////////////////////////

`define PEAK_WHITE_UPDATE                       1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES               1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                             3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                              1'b0     //changed *******
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE,
`AVERAGE_BIRIGHTNESS_LINES, `PEAK_WHITE_UPDATE}
```

```
`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80


module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                    tv_in_i2c_clock, tv_in_i2c_data);

   input reset;
   input clock_27mhz;
   output tv_in_reset_b; // Reset signal to ADV7185
   output tv_in_i2c_clock; // I2C clock output to ADV7185
   output tv_in_i2c_data; // I2C data line to ADV7185
   input source; // 0: composite, 1: s-video

   initial begin
      $display("ADV7185 Initialization values:");
      $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
      $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
      $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
      $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
      $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
      $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
      $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
      $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
```

```
   $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
   $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
   $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
   $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
   $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
   $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
   $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
   $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
end

//
// Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
//

reg [7:0] clk_div_count, reset_count;
reg clock_slow;
wire reset_slow;

initial
  begin
     clk_div_count <= 8'h00;
     // synthesis attribute init of clk_div_count is "00";
     clock_slow <= 1'b0;
     // synthesis attribute init of clock_slow is "0";
  end

always @(posedge clock_27mhz)
  if (clk_div_count == 26)
    begin
       clock_slow <= ~clock_slow;
       clk_div_count <= 0;
    end
  else
    clk_div_count <= clk_div_count+1;

always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
        .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
        .sda(tv_in_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_in_reset_b;
reg old_source;

always @(posedge clock_slow)
   if (reset_slow)
     begin
        state <= 0;
        load <= 0;
        tv_in_reset_b <= 0;
        old_source <= 0;
     end
```

```
else
  case (state)
    8'h00:
      begin
         // Assert reset
         load <= 1'b0;
         tv_in_reset_b <= 1'b0;
         if (!ack)
           state <= state+1;
      end
    8'h01:
      state <= state+1;
    8'h02:
      begin
         // Release reset
         tv_in_reset_b <= 1'b1;
         state <= state+1;
             end
    8'h03:
      begin
         // Send ADV7185 address
         data <= 8'h8A;
         load <= 1'b1;
         if (ack)
           state <= state+1;
      end
    8'h04:
      begin
         // Send subaddress of first register
         data <= 8'h00;
         if (ack)
           state <= state+1;
      end
    8'h05:
      begin
         // Write to register 0
         data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
         if (ack)
           state <= state+1;
      end
    8'h06:
      begin
         // Write to register 1
         data <= `ADV7185_REGISTER_1;
         if (ack)
           state <= state+1;
      end
    8'h07:
      begin
         // Write to register 2
         data <= `ADV7185_REGISTER_2;
         if (ack)
           state <= state+1;
      end
    8'h08:
      begin
         // Write to register 3
         data <= `ADV7185_REGISTER_3;
         if (ack)
           state <= state+1;
      end
    8'h09:
      begin
         // Write to register 4
         data <= `ADV7185_REGISTER_4;
         if (ack)
           state <= state+1;
      end
    8'h0A:
      begin
         // Write to register 5
```

```
      data <= `ADV7185_REGISTER_5;
      if (ack)
        state <= state+1;
    end
  8'h0B:
    begin
      // Write to register 6
      data <= 8'h00; // Reserved register, write all zeros
      if (ack)
        state <= state+1;
    end
  8'h0C:
    begin
      // Write to register 7
      data <= `ADV7185_REGISTER_7;
      if (ack)
        state <= state+1;
    end
  8'h0D:
    begin
      // Write to register 8
      data <= `ADV7185_REGISTER_8;
      if (ack)
        state <= state+1;
    end
  8'h0E:
    begin
      // Write to register 9
      data <= `ADV7185_REGISTER_9;
      if (ack)
        state <= state+1;
    end
  8'h0F: begin
     // Write to register A
     data <= `ADV7185_REGISTER_A;
   if (ack)
     state <= state+1;
  end
  8'h10:
    begin
      // Write to register B
      data <= `ADV7185_REGISTER_B;
      if (ack)
        state <= state+1;
    end
  8'h11:
    begin
      // Write to register C
      data <= `ADV7185_REGISTER_C;
      if (ack)
        state <= state+1;
    end
  8'h12:
    begin
      // Write to register D
      data <= `ADV7185_REGISTER_D;
      if (ack)
        state <= state+1;
    end
  8'h13:
    begin
      // Write to register E
      data <= `ADV7185_REGISTER_E;
      if (ack)
        state <= state+1;
    end
  8'h14:
    begin
      // Write to register F
      data <= `ADV7185_REGISTER_F;
      if (ack)
```

```
      state <= state+1;
  end
8'h15:
  begin
     // Wait for I2C transmitter to finish
     load <= 1'b0;
     if (idle)
      state <= state+1;
  end
8'h16:
  begin
     // Write address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
      state <= state+1;
  end
8'h17:
  begin
     data <= 8'h33;
     if (ack)
      state <= state+1;
  end
8'h18:
  begin
     data <= `ADV7185_REGISTER_33;
     if (ack)
      state <= state+1;
  end
8'h19:
  begin
     load <= 1'b0;
     if (idle)
      state <= state+1;
  end

8'h1A: begin
   data <= 8'h8A;
   load <= 1'b1;
   if (ack)
     state <= state+1;
end
8'h1B:
  begin
     data <= 8'h33;
     if (ack)
      state <= state+1;
  end
8'h1C:
  begin
     load <= 1'b0;
     if (idle)
      state <= state+1;
  end
8'h1D:
  begin
     load <= 1'b1;
     data <= 8'h8B;
     if (ack)
      state <= state+1;
  end
8'h1E:
  begin
     data <= 8'hFF;
     if (ack)
      state <= state+1;
  end
8'h1F:
  begin
     load <= 1'b0;
     if (idle)
```

```
            state <= state+1;
        end
      8'h20:
        begin
          // Idle
          if (old_source != source) state <= state+1;
          old_source <= source;
        end
      8'h21: begin
         // Send ADV7185 address
         data <= 8'h8A;
         load <= 1'b1;
         if (ack) state <= state+1;
      end
      8'h22: begin
         // Send subaddress of register 0
         data <= 8'h00;
         if (ack) state <= state+1;
      end
      8'h23: begin
         // Write to register 0
         data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
         if (ack) state <= state+1;
      end
      8'h24: begin
         // Wait for I2C transmitter to finish
         load <= 1'b0;
         if (idle) state <= 8'h20;
      end
    endcase

endmodule


//
// Register 0
//

`define CHROMA_FILTER_SELECT                    3'h0
  // 0: 1.3MHz low-pass
  // 1: 0.65MHz low-pass
  // 2: 1.0MHz low-pass
  // 3: 2.0MHz low-pass
  // 4: [Not valid]
  // 5: CIF
  // 6: QCIF
  // 7: 3.0MHz low-pass
`define LUMA_FILTER_SELECT                      3'h0
  // 0: Low-pass (NTSC)
  // 1: Low-pass (PAL)
  // 2: Notch (NTSC)
  // 3: Notch (PAL)
  // 4: Extended mode
  // 5: CIF
  // 6: QCIF
  // 7: [Not valid]
`define VIDEO_STANDARD_SELECT                   2'h0
  // 0: NTSC
  // 1: PAL (B, D, G, H, I)
  // 2: [Not valid]
  // 3: PAL (N)

`define ADV7194_REGISTER_0 {`CHROMA_FILTER_SELECT, `LUMA_FILTER_SELECT,
`VIDEO_STANDARD_SELECT}

//
// Register 1
//

`define FOUR_TIMES_OVERSAMPLING                 1'b1
`define DAC_A_ENABLE                            1'b0
```

```
`define DAC_B_ENABLE                            1'b0
`define DAC_C_ENABLE                            1'b0
`define DAC_D_ENABLE                            1'b1
  // Composite video DAC
`define DAC_E_ENABLE                            1'b1
  // S-video cluma DAC
`define DAC_F_ENABLE                            1'b1
  // S-video chroma DAC

`define ADV7194_REGISTER_1 {1'b0, `FOUR_TIMES_OVERSAMPLING, `DAC_A_ENABLE, `DAC_B_ENABLE,
`DAC_C_ENABLE, `DAC_D_ENABLE, `DAC_E_ENABLE, `DAC_F_ENABLE}

//
// Register 2
//

`define SLEEP_MODE                              1'b0
  // 0: Normal mode
  // 1: Sleep mode
`define PIXEL_DATA_VALID                        1'b1
  // 1: Enables the YCrCb data port
`define I2C_CONTROL                             1'b0
  // 0: Video standard set by NTSC/PAL pin (low=NTSC, high=PAL)
  // 1: Video standard set by register 0
`define SQUARE_PIXEL_MODE                       1'b0
  // 0: Normal
  // 1: Square pixel mode (requires special clocks)
`define PEDESTAL_CONTROL                        1'b0
  // 0: Pedestal off
  // 1: Pedestal on (NTSC only)
`define DAC_OUTPUT_CONTROL                      3'h0
  // 0: Composite output on DAC D, s-video on DACs E and F
  // (this is the only configuration supported by the 6.111 labkit hardware)

`define ADV7194_REGISTER_2 {`SLEEP_MODE, `PIXEL_DATA_VALID, `I2C_CONTROL,
`SQUARE_PIXEL_MODE, `PEDESTAL_CONTROL, `DAC_OUTPUT_CONTROL}

//
// Register 3
//

`define CLOSED_CAPTIONING_CONTROL               2'h0
  // 0: No CC data
  // 1: Odd field only
  // 2: Even field only
  // 3: Both fields
`define TELETEXT_REQUEST_MODE                   1'b0
  // ???
`define TELETEXT_ENABLE                         1'b0
  // 0: Disabled
  // 1: Teletext data on TTX pin
`define VBI_OPEN                                1'b0
  // 0: DACs blanked during vertival blanking interval
  // 1: DACs enabled during vertival blanking interval

`define ADV7194_REGISTER_3 {1'b0, `CLOSED_CAPTIONING_CONTROL, `TELETEXT_REQUEST_MODE,
`TELETEXT_ENABLE, `VBI_OPEN, 2'b00}

//
// Register 4
//

`define INTERLACE_MODE                          1'b0
  // 0: Interlaced
  // 1: Progressive
`define COLOR_BARS                              1'b0
  // 0: Normal
  // 1: Display colorbars
`define BURST_CONTROL                           1'b0
  // 0: Enable color burst on composite and chrominance channels
  // 1: Disable color burst
```

```
`define CHROMINANCE_CONTROL                     1'b0
  // 0: Enable color
  // 1: Disable color
`define ACTIVE_VIDEO_LINE_DURATION              1'b0
  // 0: CCIR Rec. 601 standard: 720 pixels
  // 1: ITU-R BT.470 standard: 710 pixels (NTSC) / 702 pixels (PAL)
`define GENLOCK_CONTROL                         2'h0
  // 0: Disable genlock
  // 1: Enable subcarrier reset pin
  // 2: Timing reset
  // 3: Enable RTC pin
`define THREE_LINE_VSYNC                        1'b0
  // 0: Disabled
  // 1: Enabled

`define ADV7194_REGISTER_4 {`INTERLACE_MODE, `COLOR_BARS, `BURST_CONTROL,
`CHROMINANCE_CONTROL, `ACTIVE_VIDEO_LINE_DURATION, `GENLOCK_CONTROL, `THREE_LINE_VSYNC}

//
// Register 5
//

`define CLAMP_POSITION                          1'b0
  // 0: Front porch
  // 1: Back porch
`define CLAMP_DELAY_DIRECTION                   1'b0
  // 0: Positive
  // 1: Negative
`define CLAMP_DELAY                             2'h0
  // 0-3: Clamp delay, in clock cycles
`define RGB_SYNC                                1'b1
  // 0: Disabled
  // 1: Enabled
`define UV_LEVEL                                2'h0
  // 0: Default levels (934mV NTSC, 700mV PAL)
  // 1: 700mV
  // 2: 1000mV
  // 3: [Not valid]
`define Y_LEVEL                                 1'b1
  // 0: Betacam levels
  // 1: SMPTE levels

`define ADV7194_REGISTER_5 {`CLAMP_POSITION, `CLAMP_DELAY_DIRECTION, `CLAMP_DELAY,
`RGB_SYNC, `UV_LEVEL, `Y_LEVEL}

//
// Register 6
//

`define PLL_ENABLE                              1'b0
  // 0: Enabled
  // 1: Disabled
`define POWER_UP_SLEEP_MODE                     1'b1
  // ??

`define ADV7194_REGISTER_6 {3'b000, 3'b000, `PLL_ENABLE, `POWER_UP_SLEEP_MODE}

//
// Register 7
//

`define PIN_62_MODE                             2'b0
  // 0: Teletext input
  // 1: ~VSO output
  // 2: Teletext input
  // 3: CLAMP output
`define CSO_HSO_CONTROL                         1'b0
  // 0: ~HSO output
  // 1: ~CSO output
`define SHARPNESS_FILTER                        1'b0
  // 0: Disable
```

```
  // 1: Enable
`define BRIGHTNESS_ADJUST                       1'b0
  // 0: Disable
  // 1: Enable
`define HUE_ADJUST                              1'b0
  // 0: Disable
  // 1: Enable
`define LUMA_SATURATION_CONTROL                 1'b0
  // 0: Disable
  // 1: Enable
`define COLOR_CONTROL                           1'b0
  // 0: Disable
  // 1: Enable

`define ADV7194_REGISTER_7 {`PIN_62_MODE, `CSO_HSO_CONTROL, `SHARPNESS_FILTER,
`BRIGHTNESS_ADJUST, `HUE_ADJUST, `LUMA_SATURATION_CONTROL, `COLOR_CONTROL}

//
// Register 8
//


module adv7194init (reset, clock_27mhz, colorbars, tv_out_reset_b,
                    tv_out_i2c_clock, tv_out_i2c_data);

   input reset;
   input clock_27mhz;
   input colorbars;
   output tv_out_reset_b; // Reset signal to ADV7194
   output tv_out_i2c_clock; // I2C clock output to ADV7194
   output tv_out_i2c_data; // I2C data line to ADV7194


   initial begin
      $display("ADV7194 Initialization values:");
      $display("  Register 0: 0x%X", `ADV7194_REGISTER_0);
      $display("  Register 1: 0x%X", `ADV7194_REGISTER_1);
      $display("  Register 2: 0x%X", `ADV7194_REGISTER_2);
      $display("  Register 3: 0x%X", `ADV7194_REGISTER_3);
      $display("  Register 4: 0x%X", `ADV7194_REGISTER_4);
      $display("  Register 5: 0x%X", `ADV7194_REGISTER_5);
      $display("  Register 6: 0x%X", `ADV7194_REGISTER_6);
      $display("  Register 7: 0x%X", `ADV7194_REGISTER_7);
   end

   //
   // Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
   //

   reg [7:0] clk_div_count, reset_count;
   reg clock_slow;
   wire reset_slow;

   initial
     begin
        clk_div_count <= 8'h00;
        // synthesis attribute init of clk_div_count is "00";
        clock_slow <= 1'b0;
        // synthesis attribute init of clock_slow is "0";
     end

   always @(posedge clock_27mhz)
     if (clk_div_count == 26)
       begin
          clock_slow <= ~clock_slow;
          clk_div_count <= 0;
       end
     else
       clk_div_count <= clk_div_count+1;

   always @(posedge clock_27mhz)
```

```
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
        .ack(ack), .idle(idle), .scl(tv_out_i2c_clock),
        .sda(tv_out_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_out_reset_b;
reg old_colorbars;

always @(posedge clock_slow)
  if (reset_slow)
    begin
       state <= 0;
       data <= 0;
       load <= 0;
       tv_out_reset_b <= 0;
    end
  else
    case (state)
      8'h00: begin
         // Assert reset
         load <= 1'b0;
         tv_out_reset_b <= 1'b0;
         if (!ack) state <= state+1;
      end
      8'h01: begin
         state <= state+1;
      end
      8'h02: begin
         // Release reset
         tv_out_reset_b <= 1'b1;
         state <= state+1;
      end
      8'h03: begin
         // Send ADV7194 address
         data <= 8'h56;
         load <= 1'b1;
         if (ack) state <= state+1;
      end
      8'h04: begin
         // Send subaddress of first register
         data <= 8'h00;
         if (ack) state <= state+1;
      end
      8'h05: begin
         // Write to register 0
         data <= `ADV7194_REGISTER_0;
         if (ack) state <= state+1;
      end
      8'h06: begin
         // Write to register 1
         data <= `ADV7194_REGISTER_1;
         if (ack) state <= state+1;
```

```
         end
         8'h07: begin
            // Write to register 2
            data <= `ADV7194_REGISTER_2;
            if (ack) state <= state+1;
         end
         8'h08: begin
            // Write to register 3
            data <= `ADV7194_REGISTER_3;
            if (ack) state <= state+1;
         end
         8'h09: begin
            // Write to register 4
            data <= `ADV7194_REGISTER_4 | {1'b0, colorbars, 6'b000000};
            if (ack) state <= state+1;
         end
         8'h0A: begin
            // Write to register 5
            data <= `ADV7194_REGISTER_5;
            if (ack) state <= state+1;
         end
         8'h0B: begin
            // Write to register 6
            data <= `ADV7194_REGISTER_6;
            if (ack) state <= state+1;
         end
         8'h0C: begin
            // Write to register 7
            data <= `ADV7194_REGISTER_7;
            if (ack) state <= state+1;
         end
         8'h0D: begin
            // Wait for I2C transmitter to finish
            load <= 1'b0;
            if (idle) state <= state+1;
         end
         8'h0E: begin
            // Idle
            if (old_colorbars != colorbars) state <= state+1;
            old_colorbars <= colorbars;
         end
         8'h0F: begin
            // Send ADV7194 address
            data <= 8'h56;
            load <= 1'b1;
            if (ack) state <= state+1;
         end
         8'h10: begin
            // Send subaddress of register 4
            data <= 8'h04;
            if (ack) state <= state+1;
         end
         8'h11: begin
            // Write to register 4
            data <= `ADV7194_REGISTER_4 | {1'b0, colorbars, 6'b000000};
            if (ack) state <= state+1;
         end
         8'h12: begin
            // Wait for I2C transmitter to finish
            load <= 1'b0;
            if (idle) state <= 8'h0E;
         end
       endcase

endmodule


//Nathan Ickes's code

module audio (reset, clock_27mhz, audio_reset_b, ac97_sdata_out, ac97_sdata_in,
```

```
            ac97_synch, ac97_bit_clock, mode, volume, source, right_in_data, ready);

   input reset, clock_27mhz;
   output audio_reset_b;
   output ac97_sdata_out;
   input ac97_sdata_in;
   output ac97_synch;
   input ac97_bit_clock;
   input [1:0] mode;
   input [4:0] volume;
   input source;
   output[19:0] right_in_data;
   output ready;

   wire ready;
   wire [7:0] command_address;
   wire [15:0] command_data;
   wire command_valid;
   reg [19:0] left_out_data, right_out_data;
   wire [19:0] left_in_data, right_in_data, sine_data, square_data;

   //
   // Reset controller
   //

   reg audio_reset_b;
   reg [9:0] reset_count;

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

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

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

   //sinewave sine(clock_27mhz, ready, sine_data);

   //squarewave square(clock_27mhz, ready, square_data);

   always @(mode or left_in_data or right_in_data/* or sine_data or square_data*/)
     case (mode)
       2'd0: begin
          left_out_data = left_in_data;
          right_out_data = right_in_data;
       end
       2'd1: begin
          left_out_data = 20'h00000;
          right_out_data = 20'h00000;
       end
     endcase

endmodule

module beeper (reset, clock_27mhz, beep, enable);

   input reset, clock_27mhz, enable;
   output beep;
```

```
   reg [15:0] count;
   reg clock_1khz;

   always @(posedge clock_27mhz)
     if (reset)
       begin
          count <= 0;
          clock_1khz <= 0;
       end
     else if (count == 13499)
       begin
          clock_1khz <= ~clock_1khz;
          count <= 0;
      end
     else
       count <= count+1;

   assign beep = enable && clock_1khz;

endmodule

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

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

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

   reg ready;

   reg ac97_sdata_out;
   reg ac97_synch;

   reg [7:0] bit_count;

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

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

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

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

always @(posedge ac97_bit_clock) begin
   // Generate the sync signal
   if (bit_count == 255)
     ac97_synch <= 1'b1;
   if (bit_count == 15)
     ac97_synch <= 1'b0;

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

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

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

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

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

   else if ((bit_count >= 56) && (bit_count <= 75))
     begin
        // Slot 3: Left channel
        ac97_sdata_out <= l_left_v ? l_left_data[19] : 1'b0;
        l_left_data <= { l_left_data[18:0], l_left_data[19] };
     end
   else if ((bit_count >= 76) && (bit_count <= 95))
     // Slot 4: Right channel
        ac97_sdata_out <= l_right_v ? l_right_data[95-bit_count] : 1'b0;
   else
     ac97_sdata_out <= 1'b0;

   bit_count <= bit_count+1;

end // always @ (posedge ac97_bit_clock)

always @(negedge ac97_bit_clock) begin
   if ((bit_count >= 57) && (bit_count <= 76))
     // Slot 3: Left channel
     left_in_data <= { left_in_data[18:0], ac97_sdata_in };
```

```
      else if ((bit_count >= 77) && (bit_count <= 96))
        // Slot 4: Right channel
        right_in_data <= { right_in_data[18:0], ac97_sdata_in };
   end

endmodule

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

module ac97commands (clock, ready, command_address, command_data,
                     command_valid, volume, source);

   input clock;
   input ready;
   output [7:0] command_address;
   output [15:0] command_data;
   output command_valid;
   input [4:0] volume;
   input source;

   reg [23:0] command;
   reg command_valid;

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

   initial begin
      command <= 4'h0;
      // synthesis attribute init of command is "0";
      command_valid <= 1'b0;
      // synthesis attribute init of command_valid is "0";
      done <= 1'b0;
      // synthesis attribute init of done is "0";
      old_ready <= 1'b0;
      // synthesis attribute init of old_ready is "0";
      state <= 16'h0000;
      // synthesis attribute init of state is "0000";
   end

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

   wire [4:0] vol;
   assign vol = 31-volume;

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

      case (state)
        4'h0: // Read ID
          begin
             command <= 24'h80_0000;
             command_valid <= 1'b1;
          end
        4'h1: // Read ID
          command <= 24'h80_0000;
        4'h2: // Master volume
          command <= { 8'h02, 3'b000, vol, 3'b000, vol };
        4'h3: // Aux volume
          command <= { 8'h04, 3'b000, vol, 3'b000, vol };
        4'h4: // Mono volume
          command <= 24'h06_8000;
        4'h5: // PCM volume
          command <= 24'h18_0808;
        4'h6: // Record source select
          if (source)
            command <= 24'h1A_0000; // microphone
          else
            command <= 24'h1A_0404; // line-in
```

```
        4'h7: // Record gain
          command <= 24'h1C_0000;
        4'h8: // Line in gain
          command <= 24'h10_8000;
        //4'h9: // Set jack sense pins
          //command <= 24'h72_3F00;
        4'hA: // Set beep volume
          command <= 24'h0A_0000;
        //4'hB: //Microphone gain
          //command <= 42'h0E_8048;
        //4'hF: // Misc control bits
          //command <= 24'h76_8000;
        default:
          command <= 24'h80_0000;
      endcase // case(state)

      old_ready <= ready;

   end // always @ (posedge clock)

endmodule // ac97commands

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Alphanumeric Display Interface
//
//
// Created: November 5, 2003
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////

module display (reset, clock_27mhz,
                disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out, dots);

   input  reset, clock_27mhz;
   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
          disp_reset_b;
   input [639:0] dots;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   //////////////////////////////////////////////////////////////////////////////
   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //
   //////////////////////////////////////////////////////////////////////////////

   reg [4:0] count;
   reg [7:0] reset_count;
   reg clock;
   wire dreset;

   always @(posedge clock_27mhz)
     begin
        if (reset)
          begin
             count = 0;
             clock = 0;
          end
        else if (count == 26)
          begin
             clock = ~clock;
             count = 5'h00;
          end
        else
          count = count+1;
     end
```

```
always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign dreset = (reset_count != 0);

assign disp_clock = ~clock;

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

reg [7:0] state;
reg [9:0] dot_index;
reg [31:0] control;

assign disp_blank = 1'b0; // low <= not blanked

always @(posedge clock)
  if (dreset)
    begin
       state <= 0;
       dot_index <= 0;
       control <= 32'h7F7F7F7F;
    end
  else
    casex (state)
      8'h00:
        begin
           // Reset displays
           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 <= state+1;
        end

      8'h01:
        begin
           // End reset
           disp_reset_b <= 1'b1;
           state <= state+1;
        end

      8'h02:
        begin
           // Initialize dot register
           disp_ce_b <= 1'b0;
           disp_data_out <= 1'b0; // dot_index[0];
           if (dot_index == 639)
             state <= state+1;
           else
             dot_index <= dot_index+1;
        end

      8'h03:
        begin
           // Latch dot data
           disp_ce_b <= 1'b1;
           dot_index <= 31;
           state <= state+1;
        end

      8'h04:
        begin
```

```
            // Setup the control register
            disp_rs <= 1'b1; // Select the control register
            disp_ce_b <= 1'b0;
            disp_data_out <= control[31];
            control <= {control[30:0], 1'b0};
            if (dot_index == 0)
              state <= state+1;
            else
              dot_index <= dot_index-1;
          end

         8'h05:
          begin
            // Latch the control register data
            disp_ce_b <= 1'b1;
            dot_index <= 639;
            state <= state+1;
          end

         8'h06:
          begin
            // Load the user's dot data into the dot register
            disp_rs <= 1'b0; // Select the dot register
            disp_ce_b <= 1'b0;
            disp_data_out <= dots[dot_index];
            if (dot_index == 0)
              state <= 5;
            else
              dot_index <= dot_index-1;
          end
       endcase

endmodule



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

module adder(clk, adder_reset, adder_enable, adder_in, adder_out);

input clk, adder_reset, adder_enable;
input[7:0] adder_in;
output[10:0] adder_out;
reg[10:0] adder_out;

always @(posedge clk)
        if(adder_reset)
                adder_out = 0;
        else if(adder_enable)
                adder_out = adder_out + adder_in;

endmodule

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

module alarm(clk, alarm, signal);

input clk, alarm;
output signal;
reg signal;
reg[20:0] count;
//parameter count_to_me = 1843199;
parameter count_to_me = 922;

always @(posedge clk)
        begin
                if(alarm)
                        begin
                                count <= 0;
                                signal <= 0;
                        end
```

```
		else if(count == count_to_me)
			begin
				count <= 21'd0;
				signal <= ~signal;
			end
		else
				count <= count + 1;
	end
endmodule

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

module alarm_reg(clk, Reset, alarm, alarm_reset);

input clk, Reset, alarm;
output alarm_reset;
reg alarm_reset;

always @(posedge clk)
	if(Reset)
		alarm_reset = 1;
	else if(alarm)
		alarm_reset = 0;

endmodule

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

module divider(clk, Reset_Sync, enable);

input clk, Reset_Sync;
output enable;
reg enable;
reg[20:0] count;
//parameter count_to_me = 9290;
parameter count_to_me = 1843199;

always @(posedge clk)
	begin
		if(Reset_Sync)
			begin
				count <= 21'd0;
				enable <= 0;
			end
		else if(count == count_to_me)
			begin
				count <= 21'd0;
				enable <= 1;
			end
		else
			begin
				count <= count + 1;
				enable <= 0;
			end
	end
endmodule

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

module laser_FSM1(clk, Reset, lasers,  sensors, alarm, select, side);

input clk, Reset, select;
input[4:0] sensors;
input[2:0] side;
output[4:0] lasers;
output alarm;
reg[4:0] lasers;
reg[1:0] state, next;
reg alarm, alarm_int;

parameter IDLE = 0;
```

```
parameter SIDE = 1;
parameter CHECK = 2;
parameter ALARM = 3;

always @(posedge clk)
	begin
		if(Reset)
			state = IDLE;
		else
			state = next;
		alarm = alarm_int;
	end

always @(state)
	begin
		next = 2'b00;
		alarm_int = 0;
		case(state)
			IDLE:	begin
					lasers = 5'b11111;
					if(select)
						next = SIDE;
					else
						next = CHECK;
				end

			SIDE:	begin
					if(select == 0)
						next = IDLE;
					else if(side == 3'b000)
						begin
							if(sensors[1] &
sensors[2] & sensors [3] & sensors[4])
								next = SIDE;
							else
								next = ALARM;
						end
					else if(side == 3'b001)
						begin
							if(sensors[0] &
sensors[2] & sensors [3] & sensors[4])
								next = SIDE;
							else
								next = ALARM;
						end
					else if(side == 3'b010)
						begin
							if(sensors[0] &
sensors[1] & sensors [3] & sensors[4])
								next = SIDE;
							else
								next = ALARM;
						end
					else if(side == 3'b011)
						begin
							if(sensors[0] &
sensors[1] & sensors [2] & sensors[4])
								next = SIDE;
							else
								next = ALARM;
						end
					else if(side == 3'b100)
						begin
							if(sensors[0] &
sensors[1] & sensors [2] & sensors[3])
								next = SIDE;
							else
								next = ALARM;
						end
				end
```

```
                        CHECK:  begin
                                                lasers = 5'b11111;
                                                if(sensors != 5'b11111)
                                                        next = ALARM;
                                                else
                                                        next = IDLE;
                                        end

                        ALARM:  begin
                                                lasers = 5'b00000;
                                                alarm_int = 1;
                                                next = IDLE;
                                        end
                endcase
        end
endmodule

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

module lasertest1(clk, select, side, lasers, sensors, Reset, alarm);

input clk, select, Reset;
input[2:0] side;
input[4:0] sensors;
output alarm;
output[4:0] lasers;

laser_FSM1 laser_FSM11(clk, Reset, lasers, sensors, alarm, select, side);

endmodule

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

module pressure(clk, Reset, bump, p_alarm);

input clk, Reset, bump;
output p_alarm;
reg p_alarm, p_alarm_int;
reg[1:0] state, next;

parameter IDLE = 0;
parameter CHECK = 1;
parameter ALARM = 2;

always @(posedge clk)
        begin
                if(Reset)
                        state = IDLE;
                else
                        state = next;
                p_alarm = p_alarm_int;
        end

always @(state)
        begin
                next = 2'b00;
                p_alarm_int = 0;
                case(state)
                        IDLE:   begin
                                                next = CHECK;
                                        end

                        CHECK:  begin
                                                if(bump)
                                                        next = IDLE;
                                                else
                                                        next = ALARM;
                                        end

                        ALARM:  begin
                                                p_alarm_int = 1;
```

```
                                        next = IDLE;
                                end

                endcase
        end
endmodule

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

module top(clk, Reset, status_ad, AD_CSbar, AD_RWbar, p_alarm, alarm, select, side,
lasers, sensors);

input clk, Reset, status_ad, select;
input[2:0] side;
input[4:0] sensors;
output alarm, p_alarm, AD_CSbar, AD_RWbar;
output[4:0] lasers;

laser_FSM1 laser_FSM11(clk, Reset, lasers, sensors, alarm, select, side);
scaleFSM2 scaleFSM21(clk, Reset_Sync, alarm, enable, status_fsm, AD_RWbar, AD_CSbar,
AD_data, adder_enable, adder_in, adder_out, adder_reset);
endmodule

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

module scaleFSM2(clk, Reset_Sync, alarm, enable, status_fsm, AD_RWbar, AD_CSbar, AD_data,
adder_enable, adder_in, adder_out, adder_reset);

input clk, Reset_Sync, status_fsm, enable;
input[7:0] AD_data;
input[10:0] adder_out;
output AD_RWbar, AD_CSbar, adder_enable, alarm, adder_reset;
output[7:0] adder_in;
reg AD_RWbar, AD_RWbar_int, AD_CSbar, AD_CSbar_int, adder_enable, adder_enable_int,
alarm, alarm_int, store, adder_reset, adder_reset_int;
reg[3:0] state, next;
reg[2:0] sample;
reg[7:0] data, adder_in;
reg[10:0] sum;

parameter INITIAL = 0;
parameter IDLE = 1;
parameter CONV0 = 2;
parameter CONV1 = 3;
parameter CONV2 = 4;
parameter WAITHIGH = 5;
parameter WAITLOW = 6;
parameter READ0 = 7;
parameter READ1 = 8;
parameter READ2 = 9;
parameter ADD = 10;
parameter COMPARE = 11;
parameter ALARM = 12;

always @(posedge clk)
        begin
                if(Reset_Sync)
                        state <= INITIAL;
                else
                        state <= next;
                AD_RWbar <= AD_RWbar_int;
                AD_CSbar <= AD_CSbar_int;
                adder_enable <= adder_enable_int;
                adder_reset <= adder_reset_int;
                alarm <= alarm_int;
        end

always @(state)
        begin
                AD_RWbar_int = 1;
                AD_CSbar_int = 1;
```

```
adder_enable_int = 0;
adder_reset_int = 0;
alarm_int = 0;
case(state)
	INITIAL:
			begin
				store = 0;
				sample = 0;
				next = IDLE;
			end

	IDLE:
			begin
				if(enable)
					if(sample == 3'b001)
						begin
							sample = 3'b000;
							next = COMPARE;
						end
					else
						next = CONV0;
				else
					next = IDLE;
			end

	CONV0:  begin
				AD_RWbar_int = 0;
				AD_CSbar_int = 0;
				next = CONV1;
			end

	CONV1:  begin
				AD_RWbar_int = 0;
				AD_CSbar_int = 0;
				next = CONV2;
			end

	CONV2:  begin
				AD_RWbar_int = 0;
				AD_CSbar_int = 0;
				next = WAITHIGH;
			end

	WAITHIGH:
			begin
				AD_CSbar_int = 0;
				if(status_fsm)
					next = WAITLOW;
				else
					next = WAITHIGH;
			end

	WAITLOW:
			begin
				AD_CSbar_int = 0;
				if(!status_fsm)
					next = READ0;
				else
					next = WAITLOW;
			end

	READ0:  begin
				AD_RWbar_int = 1;
				AD_CSbar_int = 0;
				next = READ1;
			end

	READ1:  begin
				AD_RWbar_int = 1;
				AD_CSbar_int = 0;
				next = READ2;
```

```
                                end

                        READ2:  begin
                                        AD_RWbar_int = 1;
                                        AD_CSbar_int = 0;
                                        data = AD_data;
                                        next = ADD;
                                end

                        ADD:    begin
                                        adder_in = data;
                                        adder_enable_int = 1;
                                        sample = sample + 1;
                                        next = IDLE;
                                end

                        COMPARE:
                                begin
                                        if(store)
                                                if(adder_out == sum)
                                                        begin
                                                                adder_reset_int = 1;
                                                                next = IDLE;
                                                        end
                                                else
                                                        next = ALARM;
                                        else
                                                begin
                                                        sum = adder_out;
                                                        adder_reset_int = 1;
                                                        store = 1;
                                                        next = IDLE;
                                                end
                                end

                        ALARM:  begin
                                        alarm_int = 1;
                                        next = alarm;
                                end
                endcase
        end
endmodule

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

module scaletest1(clk, Reset, status_ad, AD_RWbar, AD_CSbar, alarm, AD_data);

input clk, Reset, status_ad;
input[7:0] AD_data;
output AD_RWbar, AD_CSbar, alarm;
wire enable, adder_enable, adder_reset, status_fsm, Reset_Sync;
wire[7:0] adder_in;
wire[10:0] adder_out;

synchronizer synchronizer1(clk, Reset, Reset_Sync, status_ad, status_fsm);
divider divider1(clk, Reset_Sync, enable);
adder adder1(clk, adder_reset, adder_enable, adder_in, adder_out);
scaleFSM2 scaleFSM21(clk, Reset_Sync, alarm, enable, status_fsm, AD_RWbar, AD_CSbar,
AD_data, adder_enable, adder_in, adder_out, adder_reset);

endmodule

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

module synchronizer(clk, Reset, Reset_Sync, status_ad, status_fsm);

input clk, Reset, status_ad;
output Reset_Sync, status_fsm;
reg Reset_Sync, Reset_Sync_int, status_fsm, status_fsm_int;

always @(posedge clk)
```

```
        begin
                Reset_Sync_int <= Reset;
                status_fsm_int <= status_ad;
                Reset_Sync <= Reset_Sync_int;
                status_fsm <= status_fsm_int;
        end

endmodule
```