```
1   ////////////////////////////////////////////////////////////////////
2   // Date: 5/12/05
3   // Group 4: David Dryjanski && Andrew Pinkham
4   ////////////////////////////////////////////////////////////////////////////
5   //
6   // 6.111 FPGA Labkit -- Template Toplevel Module
7   //
8   // For Labkit Revision 004
9   //
10  //
11  // Created: October 31, 2004, from revision 003 file
12  // Author: Nathan Ickes
13  //
14  ////////////////////////////////////////////////////////////////////////////
15  //
16  // CHANGES FOR BOARD REVISION 004
17  //
18  // 1) Added signals for logic analyzer pods 2-4.
19  // 2) Expanded "tv_in_ycrcb" to 20 bits.
20  // 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
21  //    "tv_out_i2c_clock".
22  // 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
23  //    output of the FPGA, and "in" is an input.
24  //
25  // CHANGES FOR BOARD REVISION 003
26  //
27  // 1) Combined flash chip enables into a single signal, flash_ce_b.
28  //
29  // CHANGES FOR BOARD REVISION 002
30  //
31  // 1) Added SRAM clock feedback path input and output
32  // 2) Renamed "mousedata" to "mouse_data"
33  // 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
34  //    the data bus, and the byte write enables have been combined into the
35  //    4-bit ram#_bwe_b bus.
36  // 4) Removed the "systemace_clock" net, since the SystemACE clock is now
37  //    hardwired on the PCB to the oscillator.
38  //
39  ////////////////////////////////////////////////////////////////////////////
40  //
41  // Complete change history (including bug fixes)
42  //
43  // 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
44  //              actually populated on the boards. (The boards support up to
45  //              256Mb devices, with 25 address lines.)
46  //
47  // 2004-Apr-29: Change history started
48  //
49  // 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
50  //              actually populated on the boards. (The boards support up to
51  //              72Mb devices, with 21 address lines.)
52  //
53  // 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
54  //              value. (Previous versions of this file declared this port to
55  //              be an input.)
56  //
57  // 2004-Oct-31: Adapted to new revision 004 board.
58  //
59  ////////////////////////////////////////////////////////////////////////////
60
61  module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
62                 ac97_bit_clock,
63
```

```
64              vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
65              vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
66              vga_out_vsync,
67
68              tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
69              tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
70              tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
71
72              tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
73              tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
74              tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
75              tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
76
77              ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
78              ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
79
80              ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
81              ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
82
83              clock_feedback_out, clock_feedback_in,
84
85              flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
86              flash_reset_b, flash_sts, flash_byte_b,
87
88              rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
89
90              mouse_clock, mouse_data, keyboard_clock, keyboard_data,
91
92              clock_27mhz, clock1, clock2,
93
94              disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
95              disp_reset_b, disp_data_in,
96
97              button0, button1, button2, button3, button_enter, button_right,
98              button_left, button_down, button_up,
99
100             switch,
101
102             led,
103
104             user1, user2, user3, user4,
105
106             daughtercard,
107
108             systemace_data, systemace_address, systemace_ce_b,
109             systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
110
111             analyzer1_data, analyzer1_clock,
112             analyzer2_data, analyzer2_clock,
113             analyzer3_data, analyzer3_clock,
114             analyzer4_data, analyzer4_clock);
115
116     output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
117     input  ac97_bit_clock, ac97_sdata_in;
118
119     output [7:0] vga_out_red, vga_out_green, vga_out_blue;
120     output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
121         vga_out_hsync, vga_out_vsync;
122
123     output [9:0] tv_out_ycrcb;
124     output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
125         tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
126         tv_out_subcar_reset;
```

```
127
128     input  [19:0] tv_in_ycrcb;
129     input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
130         tv_in_hff, tv_in_aff;
131     output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
132         tv_in_reset_b, tv_in_clock;
133     inout  tv_in_i2c_data;
134
135     inout  [35:0] ram0_data;
136     output [18:0] ram0_address;
137     output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
138     output [3:0] ram0_bwe_b;
139
140     inout  [35:0] ram1_data;
141     output [18:0] ram1_address;
142     output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
143     output [3:0] ram1_bwe_b;
144
145     input  clock_feedback_in;
146     output clock_feedback_out;
147
148     inout  [15:0] flash_data;
149     output [23:0] flash_address;
150     output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
151     input  flash_sts;
152
153     output rs232_txd, rs232_rts;
154     input  rs232_rxd, rs232_cts;
155
156     input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
157
158     input  clock_27mhz, clock1, clock2;
159
160     output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
161     input  disp_data_in;
162     output  disp_data_out;
163
164     input  button0, button1, button2, button3, button_enter, button_right,
165         button_left, button_down, button_up;
166     input  [7:0] switch;
167     output [7:0] led;
168
169     output [31:0] user1, user4, user2;
170     input [31:0] user3;
171
172     inout [43:0] daughtercard;
173
174     inout  [15:0] systemace_data;
175     output [6:0]  systemace_address;
176     output systemace_ce_b, systemace_we_b, systemace_oe_b;
177     input  systemace_irq, systemace_mpbrdy;
178
179     output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
180             analyzer4_data;
181     output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;
182
183
184     //output [9:0] video_data;
185     //////////////////////////////////////////////////////////////////////////
186     //
187     // I/O Assignments
188     //
189     //////////////////////////////////////////////////////////////////////////
```

```
190
191     // Audio Input and Output
192     assign beep= 1'b0;
193     assign audio_reset_b = 1'b0;
194     assign ac97_synch = 1'b0;
195     // ac97_sdata_out and ac97_sdata_out are inputs;
196
197     // VGA Output
198     assign vga_out_red = 10'h0;
199     assign vga_out_green = 10'h0;
200     assign vga_out_blue = 10'h0;
201     assign vga_out_sync_b = 1'b1;
202     assign vga_out_blank_b = 1'b1;
203     assign vga_out_pixel_clock = 1'b0;
204     assign vga_out_hsync = 1'b0;
205     assign vga_out_vsync = 1'b0;
206
207     // SRAMs
208     assign ram0_data = 36'hZ;
209     assign ram0_address = 19'h0;
210     assign ram0_adv_ld = 1'b0;
211     assign ram0_clk = 1'b0;
212     assign ram0_cen_b = 1'b1;
213     assign ram0_ce_b = 1'b1;
214     assign ram0_oe_b = 1'b1;
215     assign ram0_we_b = 1'b1;
216     assign ram0_bwe_b = 4'hF;
217     assign ram1_data = 36'hZ;
218     assign ram1_address = 19'h0;
219     assign ram1_adv_ld = 1'b0;
220     assign ram1_clk = 1'b0;
221     assign ram1_cen_b = 1'b1;
222     assign ram1_ce_b = 1'b1;
223     assign ram1_oe_b = 1'b1;
224     assign ram1_we_b = 1'b1;
225     assign ram1_bwe_b = 4'hF;
226     assign clock_feedback_out = 1'b0;
227     // clock_feedback_in is an input
228
229     // Flash ROM
230     assign flash_data = 16'hZ;
231     assign flash_address = 24'h0;
232     assign flash_ce_b = 1'b1;
233     assign flash_oe_b = 1'b1;
234     assign flash_we_b = 1'b1;
235     assign flash_reset_b = 1'b0;
236     assign flash_byte_b = 1'b1;
237     // flash_sts is an input
238
239     // RS-232 Interface
240     assign rs232_txd = 1'b1;
241     assign rs232_rts = 1'b1;
242     // rs232_rxd and rs232_cts are inputs
243
244     // PS/2 Ports
245     // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs
246
247     // Daughtercard Connectors
248     assign daughtercard = 44'hZ;
249
250     // SystemACE Microprocessor Port
251     assign systemace_data = 16'hZ;
252     assign systemace_address = 7'h0;
```

```
253      assign systemace_ce_b = 1'b1;
254      assign systemace_we_b = 1'b1;
255      assign systemace_oe_b = 1'b1;
256      // systemace_irq and systemace_mpbrdy are inputs
257
258
259      assign analyzer4_data = 16'h0;
260      assign analyzer4_clock = 1'b1;
261
262      ////////////////////////////////////////////////////////////////
263      ///Video Implementation
264      ////////////////////////////////////////////////////////////////
265
266      wire reset;
267      SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(reset),
268                .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
269      defparam reset_sr.INIT = 16'hFFFF;
270
271      PULLUP pu_in (.O(tv_in_i2c_data));
272      PULLUP pu_out (.O(tv_out_i2c_data));
273
274      wire [9:0] video_data, tv_out_ycrcb, lum_data, x_min, x_max, x_count;   //y_max, y_min,
275      wire [7:0] y_max, y_min;
276      wire [9:0] line_count, real_xmin, real_xmax;
277      wire[3:0] position, pos;
278      wire [2:0] find_state;
279      wire odd_start, toggle, frame_done;
280      wire edge_state;
281      wire tv_in_line_clock1, reset_sync;
282
283      mod_sync sync(.clk(tv_in_line_clock1), .in(reset), .out(reset_sync)); //reset synchroni
    zer to video interface
284
285      //video module that outputs camera input
286      video_test video(.reset(reset), .clock_27mhz(clock_27mhz), .tv_out_ycrcb(tv_out_ycrcb),
     .tv_out_reset_b(tv_out_reset_b), .tv_out_clock(tv_out_clock),
287               .tv_out_i2c_clock(tv_out_i2c_clock), .tv_out_i2c_data(tv_out_i2c_data), .tv_out
    _pal_ntsc(tv_out_pal_ntsc),
288               .tv_out_hsync_b(tv_out_hsync_b), .tv_out_vsync_b(tv_out_vsync_b), .tv_out_blank
    _b(tv_out_blank_b),
289               .tv_out_subcar_reset(tv_out_subcar_reset), .tv_in_ycrcb(tv_in_ycrcb[19:10]), .t
    v_in_data_valid(tv_in_data_valid),
290               .tv_in_line_clock1(tv_in_line_clock1), .tv_in_line_clock2(tv_in_line_clock2), .
    tv_in_aef(tv_in_aef), .tv_in_hff(tv_in_hff),
291               .tv_in_aff(tv_in_aff), .tv_in_i2c_clock(tv_in_i2c_clock), .tv_in_i2c_data(tv_in
    _i2c_data), .tv_in_fifo_read(tv_in_fifo_read),
292               .tv_in_fifo_clock(tv_in_fifo_clock), .tv_in_iso(tv_in_iso), .tv_in_reset_b(tv_i
    n_reset_b), .tv_in_clock(tv_in_clock));
293
294      //video interface top level module
295      video_top2 top(.clk(tv_in_line_clock1), .reset(reset_sync), .data(tv_in_ycrcb[19:10]),
    .video_data(video_data),
296                     .odd_start(odd_start),  .line_count(line_count),
297                     .find_state(find_state), .lum_data(lum_data), .switch(toggle), .edge_stat
    e(edge_state),
298                     .x_max(x_max), .x_min(x_min), .y_max(y_max), .y_min(y_min), .frame_done(f
    rame_done),
299                     .x_count(x_count), .position(position), .pos(pos));
300
301   wire [7:0] xin, yin, zin, x, y, z, xjoy, yjoy, q1, q2, q3, q4;
302   wire jump, crouch;
303   wire status1, status2, status3, statussync1, statussync2, statussync3;
304   wire resetsync;
```

```
305   wire [2:0] control;
306   wire CE, CS;
307   wire adstart, adbusy, dastart, dabusy, sample, adload, daload;
308   wire xgo, ygo, xdir, ydir, lclick;
309   wire xsig1, xsig2, ysig1, ysig2;
310   wire [639:0] dots;
311   wire [7:0] xmove, ymove;
312
313   scroll scroll(resetsync, clock_27mhz, dots);
314   display display(resetsync, clock_27mhz, disp_blank, disp_clock, disp_rs, disp_ce_b, disp_r
      eset_b, disp_data_out, dots);
315   bigmajor bigmajor1 (.adstart(adstart), .adbusy(adbusy), .dastart(dastart), .dabusy(dabusy), .xac
316   ad ad1 (.status1(statussync1), .status2(statussync2), .status3(statussync3), .control(cont
      rol), .adload(adload), .start(adstart), .busy(adbusy), .reset(resetsync), .clk(clock_27mhz
      ));
317   da da1 (.CEbar(CE), .CSbar(CS), .daload(daload), .start(dastart), .busy(dabusy), .reset(re
      setsync), .clk(clock_27mhz));
318   sampledivider sampledivider1 (.clk(clock_27mhz), .reset(resetsync), .sample(sample));
319   synchronizer synchronizer1 (.reset(reset), .status1(status1), .status2(status2), .status3(
      status3), .resetsync(resetsync), .statussync1(statussync1), .statussync2(statussync2), .st
      atussync3(statussync3), .clock(clock_27mhz));
320   adreg adreg1 (.da(xin), .qa(x), .db(yin), .qb(y), .dc(zin), .qc(z), .clk(clock_27mhz), .lo
      ad(adload));
321   dareg dareg1 (.d1(xjoy), .d2(yjoy), .d3(xmove), .d4(ymove), .q1(q1), .q2(q2), .q3(q3), .q4
      (q4), .clk(clock_27mhz), .load(daload));
322
323   photo2 photo2(.xsig1(xsig1), .xsig2(xsig2), .ysig1(ysig1), .ysig2(ysig2), .gox(xgo), .goy(
      ygo), .ydir(ydir), .xdir(xdir), .clk(clock_27mhz), .reset(resetsync));
324
325   //inputs
326   assign xin = user3[7:0];
327   assign yin = user3[15:8];
328   assign zin = user3[23:16];
329   assign status1 = user3[31];
330   assign status2 = user3[30];
331   assign status3 = user3[29];
332
333   //outputs
334   assign user4[7:0] = q1[7:0];
335   assign user4[15:8] = q2[7:0];
336   assign user4[23:16] = q3[7:0];
337   assign user1[7:0] = q4[7:0];
338   assign user2[0] = CS;
339   assign user2[1] = CE;
340   assign user1[31:29] = control[2:0];
341   assign led = ~{4'b0000, pos[3:0]};
342   assign user2[7] = jump;
343   assign user2[8] = ~crouch;
344
345   assign user2[2] = xgo;
346   assign user2[3] = ygo;
347   assign user2[4] = xdir;
348   assign user2[5] = ydir;
349   assign user2[6] = lclick;
350
351   // synthesis attribute keep of clock_27mhz is true;
352   // synthesis attribute keep of tv_in_line_clock1 is true;
353     //synthesis attribute period of clock_27mhz is 37ns;
354     //synthesis attribute period of tv_in_line_clock1 is 37ns;
355
356   endmodule
357
358
```

```
1     ////////////////////////////
2     // Name: Group 4
3     // Date: 4/2/2005
4     // Module: Synchronizer
5     ////////////////////////////
6
7     module synchronizer (reset, status1, status2, status3, resetsync, statussync1, statussync2
      , statussync3, clock);
8           input reset, status1, status2, status3, clock;
9           output resetsync, statussync1, statussync2, statussync3;
10
11          reg resetsync, statussync1, statussync2, statussync3;
12          reg q1, q2, q3, q4;
13
14    always @ (posedge clock)
15    begin
16          q1 <= reset;
17          resetsync <= q1;
18    end
19
20    always @ (posedge clock)
21    begin
22          q2 <= status1;
23          statussync1 <= q2;
24    end
25
26    always @ (posedge clock)
27    begin
28          q3 <= status2;
29          statussync2 <= q3;
30    end
31
32    always @ (posedge clock)
33    begin
34          q4 <= status3;
35          statussync3 <= q4;
36    end
37
38    endmodule
39
```

```
1   ////////////////////////////////
2   // Name: Andrew Pinkham
3   // Date: 5/05/2005
4   // Module: complete A/D FSM
5   ////////////////////////////////
6
7   module ad (status1, status2, status3, control, adload, start, busy, reset, clk);
8        input status1, status2, status3;
9        output [2:0] control;
10       output adload;
11       input reset, clk;
12       input start;
13       output busy;
14
15       reg [2:0] control;
16       reg adload, busy;
17       reg [3:0] next;
18
19       parameter initialize = 0;
20       parameter adenable = 1;
21       parameter waitconversion = 2;
22       parameter adread = 3;
23
24  //control[2:0] = 2.CEbar, 1.CSbar, 0.RWbar
25
26  always @ (posedge clk) begin       //synchronous reset
27  if (reset) next <= initialize;          //global reset
28
29  case (next)
30
31  initialize:
32  begin
33  control[2:0] = 3'b111;
34  adload <= 0;
35  busy <= 0;
36  if (start) next <= adenable;
37  else next <= initialize;
38  end
39
40  adenable:
41  begin
42  busy <= 1;
43  control[2:0] = 3'b000;
44  if ((status1 == 1) && (status2 == 1) && (status3 == 1)) next <= waitconversion;
45  else next <= adenable;
46  end
47
48  waitconversion:
49  begin
50  control[2:0] = 3'b001;
51  if ((status1 == 0) && (status2 == 0) && (status3 == 0)) next <= adread;
52  else next <= waitconversion;
53  end
54
55  adread:
56  begin
57  adload <= 1;
58  next <= initialize;
59  end
60
61  endcase
62
63  end  //always
```

```
64
65      endmodule
66
67
```

```
1     ////////////////////////////////
2     // Name: Andrew Pinkham
3     // Date: 4/2/2005
4     // Module: major FSM
5     ////////////////////////////////
6
7     module bigmajor (adstart, adbusy, dastart, dabusy, xaccel, xjoy, yaccel, yjoy, zaccel, jum
      p, crouch, movex, movey, xgo, ygo, xdir, ydir, lclick, position, sample, clk, reset);
8          input adbusy, dabusy;
9          input sample, clk, reset;
10         input [7:0] xaccel, yaccel, zaccel;
11         input [3:0] position;
12         output adstart, dastart;
13         output [7:0] xjoy, yjoy, movex, movey;
14         output lclick, xgo, ygo, xdir, ydir;
15         output jump, crouch;
16
17         reg adstart, dastart;
18         reg [7:0] xjoy, yjoy, movex, movey;
19         reg jump, crouch;
20         reg [9:0] xcalc, ycalc;
21         reg lclick, xgo, ygo, xdir, ydir;
22         reg [7:0] samp1;
23         reg [15:0] clickcount;
24
25         reg [3:0] next;
26
27         parameter accelthresh = 3;
28
29    always @ (posedge clk) begin
30    if (reset) next <= 0;
31    case (next)
32
33    0:
34    begin
35    adstart <= 0;
36    dastart <= 0;
37    if (sample) next <= 1;
38    else next <= 0;
39    end
40
41    1:
42    begin
43    if (clickcount > 1000) begin
44         lclick <= 0;
45         jump <= 0;
46         clickcount <= 0;
47         end
48    else begin
49         clickcount <= clickcount + 1;
50         lclick <= lclick;
51         jump <= jump;
52         end
53    adstart <= 1;
54    next <= 2;
55    end
56
57    2:
58    begin
59    adstart <= 0;
60    if (!adbusy) next <= 3;
61    else next <= 2;
62    end
```

```
63
64    3:
65    //simple first difference
66    begin
67    if (samp1 == 0) begin
68         samp1 <= xaccel;
69         end
70    else begin
71         if ((xaccel >= samp1 + accelthresh) || (xaccel <= samp1 - accelthresh)) begin
72              lclick <= 1;
73              clickcount <= 0;
74              samp1 <= 0;
75              end
76         else begin
77              samp1 <= 0;
78              end
79         end
80    next <= 4;
81    end
82
83    4:
84    begin
85    xcalc[9:0] <= 4*xaccel[7:0] - 190;
86    ycalc[9:0] <= 4*yaccel[7:0] - 190;
87    xjoy[7:0] <= xcalc[7:0];
88    yjoy[7:0] <= ycalc[7:0];
89    next <= 5;
90    end
91
92    5:
93    begin
94    if ((xjoy > 90) && (xjoy < 180)) begin xjoy[7:0] <= 86; end     //set upper limit of 90 ->
       3.6 volts
95    if ((xjoy < 10) || (xjoy > 180)) begin xjoy[7:0] <= 5; end      //set lower limit of 5 ->
      100 mV
96    if ((yjoy > 90) && (yjoy < 180)) begin yjoy[7:0] <= 86; end
97    if ((yjoy < 10) || (yjoy > 180)) begin yjoy[7:0] <= 5; end
98    if ((zaccel > 100) && (zaccel < 200)) begin
99                                             jump <= 1;
100                                            clickcount <= 0;
101                                            crouch <= 0;
102                                            end
103   else jump <= 0;
104   if ((zaccel < 40) || (zaccel > 200)) begin
105                                            crouch <= 1;
106                                            jump <= 0;
107                                            end
108   next <= 6;
109   end
110
111   6:
112   begin
113   //mouse movement
114   jump <= jump;
115   crouch <= crouch;
116   if (xaccel > 64) begin
117        xgo <= 1;
118        xdir <= 0;
119        end
120   else if (xaccel < 52) begin
121        xgo <= 1;
122        xdir <= 1;
123        end
```

```
124   else begin
125        xgo <= 0;
126        xdir <= 0;
127        end
128
129   if (yaccel > 64) begin
130        ygo <= 1;
131        ydir <= 1;
132        end
133   else if (yaccel < 52) begin
134        ygo <= 1;
135        ydir <= 0;
136        end
137   else begin
138        ygo <= 0;
139        ydir <= 0;
140        end
141
142   ///xbox movement
143   case (position)
144   0: begin
145   movex <= 45;
146   movey <= 45;
147   end
148   1: begin
149   movex <= 0;
150   movey <= 0;
151   end
152   2: begin
153   movex <= 45;
154   movey <= 0;
155   end
156   3: begin
157   movex <= 95;
158   movey <= 0;
159   end
160   4: begin
161   movex <= 0;
162   movey <= 45;
163   end
164   5: begin
165   movex <= 95;
166   movey <= 45;
167   end
168   6: begin
169   movex <= 0;
170   movey <= 95;
171   end
172   7: begin
173   movex <= 45;
174   movey <= 95;
175   end
176   8: begin
177   movex <= 95;
178   movey <= 95;
179   end
180   endcase
181
182   next <= 7;
183   end
184
185   7:
186   begin
```

```
187   dastart <= 1;
188   next <= 8;
189   end
190
191   8:
192   begin
193   dastart <= 0;
194   if (!dabusy) next <= 0;
195   else next <= 8;
196   end
197
198   endcase
199
200   end
201
202   endmodule
203
```

```
1   ////////////////////////////////
2   // Name: Andrew Pinkham
3   // Date: 4/2/2005
4   // Module: Approximation FSM 4 compass
5   ////////////////////////////////
6
7   module cmps (start, busy, acurve, bcurve, heading, dots, sample, clk, reset);
8        input busy;
9        input sample, clk, reset;
10       input [7:0] acurve, bcurve;
11       output start;
12       output [8:0] heading;
13
14       output [639:0] dots;
15
16       reg start;
17       reg [8:0] heading;
18       reg [3:0] dec_dig;
19       reg [39:0] dig0, dig1, dig2, dig;
20
21       parameter initialize = 0;
22       parameter ad = 1;
23       parameter adwait = 2;
24       parameter detsect = 3;
25       parameter section1 = 4;
26       parameter section2 = 5;
27       parameter section3 = 6;
28       parameter section4 = 7;
29       parameter disp1 = 8;
30       parameter disp2 = 9;
31       parameter disp3 = 10;
32
33       reg [4:0] next;
34
35  always @ (posedge clk) begin
36  if (reset) next = initialize;
37
38  case (next)
39
40  initialize:
41  begin
42  start = 0;
43  if (sample) next = ad;
44  else next = initialize;
45  end
46
47  ad:
48  begin
49  start = 1;
50  next = adwait;
51  end
52
53  adwait:
54  begin
55  start = 0;
56  if (!busy) next = detsect;
57  else next = adwait;
58  end
59
60  detsect:
61  begin
62  if ((acurve > bcurve) && (acurve >= 27)) next = section1;
63  else if ((acurve > bcurve) && (acurve < 27)) next = section2;
```

```
64    else if ((bcurve > acurve) && (bcurve < 27)) next = section3;
65    else if ((bcurve > acurve) && (bcurve >= 27)) next = section4;
66    else next = initialize;
67    end
68
69    section1:
70    begin
71    //heading = 45degs + (2.9V - bcurve)*90degs/.8V
72    heading = 45 + (27 - bcurve)*9;
73    next = disp1;
74    end
75
76    section2:
77    begin
78    //heading = 135degs + (2.9V - acurve)*90degs/.8V
79    heading = 135 + (27 - acurve)*9;
80    next = disp1;
81    end
82
83    section3:
84    begin
85    //heading = 225degs + (bcurve - 2.1V)*90degs/.8V
86    heading = 225 + (bcurve - 18)*9;
87    next = disp1;
88    end
89
90    section4:
91    begin
92    //heading = 315degs + (acurve - 2.1V)*90degs/.8V
93    heading = 315 + (acurve - 18)*9;
94    if (heading >= 360) heading = (heading - 360);
95    next = disp1;
96    end
97
98    disp1:
99    begin
100   dec_dig = heading%8;
101   dig0 = dig;
102   heading = heading / 8;
103   next = disp2;
104   end
105
106   disp2:
107   begin
108   dec_dig = heading%8;
109   dig1 = dig;
110   heading = heading / 8;
111   next = disp3;
112   end
113
114   disp3:
115   begin
116   dec_dig = heading%8;
117   dig2 = dig;
118   next = initialize;
119   end
120
121   endcase
122
123   end
124
125      always @(dec_dig)
126        case (dec_dig)
```

```
127         5'd09:
128        dig <= 40'b00000110_01001001_01001001_00101001_00011110;  //9
129         5'd08:
130        dig <= 40'b00110110_01001001_01001001_01001001_00110110;  //8
131         5'd07:
132        dig <= 40'b00000001_01110001_00001001_00000101_00000011;  //7
133         5'd06:
134        dig <= 40'b00111100_01001010_01001001_01001001_00110000;  //6
135         5'd05:
136        dig <= 40'b00100111_01000101_01000101_01000101_00111001; //5
137         5'd04:
138        dig <= 40'b00011000_00010100_00010010_01111111_00010000; //4
139         5'd03:
140        dig <= 40'b00100010_01000001_01001001_01001001_00110110; //3
141         5'd02:
142        dig <= 40'b01100010_01010001_01001001_01001001_01000110; //2
143         5'd01:
144        dig <= 40'b00000000_01000010_01111111_01000000_00000000; //1
145         5'd00:
146        dig <= 40'b00111110_01010001_01001001_01000101_00111110; //0
147         default:
148        dig <= 40'b00000010_00000001_01010001_00001001_00000110; //?
149       endcase
150
151     assign dots = {  40'b00000000_00000000_00000000_00000000_00000000, // ' '
152                      40'b00000000_00000000_00000000_00000000_00000000, // ' '
153                      40'b00000000_00000000_00000000_00000000_00000000, // ' '
154                      40'b00000000_00000000_00000000_00000000_00000000, // ' '
155                      40'b00000000_00000000_00000000_00000000_00000000, // ' '
156                      40'b00111110_01000001_01000001_01000001_00100010, // C
157                      40'b00111110_01000001_01000001_01000001_00111110, // O
158                      40'b01111111_00000010_00001100_00000010_01111111, // M
159                      40'b01111111_00001001_00001001_00001001_00000110, // P
160                      40'b01111110_00001001_00001001_00001001_01111110, // A
161                      40'b00100110_01001001_01001001_01001001_00110010, // S
162                      40'b00100110_01001001_01001001_01001001_00110010, // S
163                      40'b00000000_00110110_00110110_00000000_00000000, // ':'
164                      dig2,
165                      dig1,
166                      dig0 };
167
168   endmodule
169
```

```
1    //David Dryjanski and Andrew Pinkham
2    //Enhanced Gaming and Pointing - 6.111 Spring 2005
3
4    module edge_detect2(clk, reset, data, odd_lines, x_max, x_min, y_max, y_min,
5                        lum_data, switch, state, x_count, frame_done, line_count);
6
7    // Edge Detect takes in the video_data from the find_frame module and analyzes the luminan
     ce data to determine
8    // the x_min, x_max, y_min, and y_max positions of the person.
9
10   input clk, reset, odd_lines;
11   input [9:0] data, line_count;
12   output [9:0] x_min, x_max, lum_data, x_count; //y_max, y_min,, y_count
13   reg [9:0] x_count, threshold, x_min, x_max,  lum_data; //y_max, y_min, y_count,
14   output [7:0] y_max, y_min;
15   reg [7:0] y_max, y_min;
16   output [3:0] state;
17   reg [3:0] state;
18   output switch, frame_done;
19   reg switch, frame_done;
20
21   //modified
22   reg x_min_set;
23   reg x_max_set;
24   reg y_min_set;
25   reg y_max_set;
26
27   parameter idle = 0;
28   parameter odd_line_scan = 1;
29   parameter odd_update = 2;
30
31   parameter num_odd_lines = 10'd254;  //253 number of odd lines
32
33   parameter lum_init_val = 10'h3AC; //very light
34
35   always @ (posedge clk)
36
37   begin
38
39   if (reset)
40        begin
41        state <= idle;
42        end
43   case (state)
44        idle:begin
45             threshold <= 10'h100;//13E;
46             x_min <= 719;
47             x_max <= 0;     //initial values for the x/y extremes
48             y_max <= 0;
49             y_min <= 243;
50             x_count <= 0;   //pixel counter for each line
51             frame_done <= 0; //signals at end of a complete frame
52             switch <= 0;    //variable that keeps track of only luminance data (not chromina
     nce)
53             x_max_set <= 0;
54             x_min_set <= 0; //indicator variables
55             y_max_set <= 0;
56             y_min_set <= 0;
57             lum_data <= lum_init_val;
58             state <= (odd_lines) ? odd_line_scan : state;
59             end
60
61        odd_line_scan: begin
```

```
62                      frame_done <= 0;
63                      if (switch) //720 samples, 2 words per sample(ie clock ticks)
64                           begin
65                           lum_data <= data;
66                           x_count <= x_count + 1;
67                           if(((x_count > 10'd30) && (x_count < 10'd700)) && ((line_count > 10'd9) && (line_count < 10'd245)))
68                                //eliminates edges of the picture to ensure sole video recognition of the person
69                                //updates the current x/y extreme position as each line is traversed
70                                begin
71                                if ((data < threshold) && (x_min_set == 0))
72                                     begin
73                                     x_min_set <= 1;
74                                     x_min <= x_count;
75                                     end
76                                if ((data > threshold) && (x_min_set == 1) && (x_max_set == 0))
77                                     begin
78                                     x_max_set <= 1;
79                                     x_max <= x_count;
80                                     end
81                                if ((data < threshold) && (line_count < y_min))
82                                     begin
83                                     y_min_set <= 1;
84                                     y_min <= line_count;
85                                     end
86                                if ((data < threshold) && (line_count > y_max) && (y_min_set ==1)) //
87                                     begin
88                                     y_max_set <= 1;
89                                     y_max <= line_count;
90                                     end
91                                end
92                           switch <= 0;
93                           end
94                      else begin      y_min <= y_min;
95                                      y_max <= y_max;
96                                      switch <= ~switch; //if chrominance data, do nothing
97                                      lum_data <= lum_data;
98                       end
99                      state <= (x_count != 10'd720) ?  state : odd_update;
100                     end
101
102          odd_update:    begin
103                     y_min <= y_min;
104                     y_max <= y_max;
105                     frame_done <= 0;
106                     x_max_set <= 0;
107                     x_min_set <= 0;  //reset all variables
108                     x_count <= 0;
109                     lum_data <= lum_init_val;
110                     switch <= 1;
111                     if ((line_count == num_odd_lines)&&(odd_lines)) //check if at end of frame
112                          begin
113                          frame_done <= 1;
114                          state <= idle;
115                          end
116                     else if(odd_lines) //begin analyzing incoming data
117                               begin
118                               state <= odd_line_scan;
```

```
119                                 end
120                            else begin
121                                 state <= state;
122                                 end
123                       end
124          endcase
125
126    end
127    endmodule
128
129
```

```
1   //David Dryjanski and Andrew Pinkham
2   //Enhanced Gaming and Pointing - 6.111 Spring 2005
3
4   module find_frame2(clk, reset, start_odd_load, data_in, data_out,
5                   state, line_count);
6
7   //This module takes in the video decoder output tv_in_ycrcb and locates the odd active vid
    eo region data.
8
9   input clk, reset;
10  input [9:0] data_in;
11  output start_odd_load;
12  reg start_odd_load;
13  output [9:0] data_out;
14  output [9:0] line_count;
15  reg [9:0] line_count;
16  reg [9:0] data_out;
17  output [2:0] state;
18  reg[2:0] state;
19
20  parameter idle = 0;
21  parameter find_time_code = 1;
22  parameter find_first_z = 2;
23  parameter find_second_z = 3;
24  parameter find_xyz = 4;
25  parameter throughput = 5;
26
27
28  always @ (posedge clk)
29  begin
30       if (reset)
31         begin
32         state <= idle;
33         end
34
35       else case (state)
36            idle: begin
37                 start_odd_load <=0;  //initializes all variables
38                 line_count <= 0;
39                 data_out <= 0;
40                 state <= find_time_code;
41                 end
42
43            find_time_code:
44                 begin
45                 start_odd_load <=0;
46                 line_count <= line_count;
47                 data_out <= 0;             // checks for the first word in the Video Timeco
    de
48                 if (data_in == 10'h3FF)
49                      begin
50                      state <= find_first_z;
51                      end
52                 else state <= state;
53                 end
54
55            find_first_z:      //checks for the first zero in the video timecode
56                 begin
57                 start_odd_load <=0;
58                 line_count <= line_count;
59                 data_out <= 0;
60                 if (data_in == 10'h0)
61                      begin
```

```
62                    state <= find_second_z;
63                    end
64               else state <= find_time_code;
65               end
66
67          find_second_z:  //checks for the second zero in the video timecode
68               begin
69               start_odd_load <=0;
70               line_count <= line_count;
71               data_out <= 0;
72               if (data_in == 10'h0)
73                    begin
74                    state <= find_xyz;
75                    end
76               else state <= find_time_code;
77               end
78
79          find_xyz: //checks the xyz word in the video timecode
80               begin
81               start_odd_load <=0;
82               data_out <= 0;
83               if(data_in == 10'h2D8) //end of the odd active video region
84                    begin
85                    state <= idle;
86                    end
87               else if(data_in == 10'h200) //signals the start of odd active video
88                    begin
89                    data_out <= 0;
90                    line_count <= line_count + 1;
91                    state <= throughput;
92                    end
93                    else begin           //search for the next timecode
94                         data_out <=0;
95                         line_count <= line_count;
96                         state <= find_time_code;
97                         end
98               end
99
100         throughput:
101              begin
102              if (data_in == 10'h3FF)  //has reached end of a line
103                   begin
104                   line_count <= line_count;
105                   data_out <= 0;
106                   start_odd_load <= 0;
107                   state <= find_first_z;
108                   end
109              else begin     //copies the odd active video data
110                   data_out <= data_in;
111                   line_count <= line_count;
112                   start_odd_load <= 1;
113                   state <= state;
114                   end
115              end
116         endcase
117   end
118   endmodule
119
120
121
122
123
124
```

```
1   ////////////////////////////////////////////////////////////////////
2   // Date: 5/12/05
3   // Group 4: David Dryjanski && Andrew Pinkham
4   ////////////////////////////////////////////////////////////////////////////
5   //
6   // 6.111 FPGA Labkit -- Template Toplevel Module
7   //
8   // For Labkit Revision 004
9   //
10  //
11  // Created: October 31, 2004, from revision 003 file
12  // Author: Nathan Ickes
13  //
14  ////////////////////////////////////////////////////////////////////////////
15  //
16  // CHANGES FOR BOARD REVISION 004
17  //
18  // 1) Added signals for logic analyzer pods 2-4.
19  // 2) Expanded "tv_in_ycrcb" to 20 bits.
20  // 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
21  //    "tv_out_i2c_clock".
22  // 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
23  //    output of the FPGA, and "in" is an input.
24  //
25  // CHANGES FOR BOARD REVISION 003
26  //
27  // 1) Combined flash chip enables into a single signal, flash_ce_b.
28  //
29  // CHANGES FOR BOARD REVISION 002
30  //
31  // 1) Added SRAM clock feedback path input and output
32  // 2) Renamed "mousedata" to "mouse_data"
33  // 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
34  //    the data bus, and the byte write enables have been combined into the
35  //    4-bit ram#_bwe_b bus.
36  // 4) Removed the "systemace_clock" net, since the SystemACE clock is now
37  //    hardwired on the PCB to the oscillator.
38  //
39  ////////////////////////////////////////////////////////////////////////////
40  //
41  // Complete change history (including bug fixes)
42  //
43  // 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
44  //              actually populated on the boards. (The boards support up to
45  //              256Mb devices, with 25 address lines.)
46  //
47  // 2004-Apr-29: Change history started
48  //
49  // 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
50  //              actually populated on the boards. (The boards support up to
51  //              72Mb devices, with 21 address lines.)
52  //
53  // 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
54  //              value. (Previous versions of this file declared this port to
55  //              be an input.)
56  //
57  // 2004-Oct-31: Adapted to new revision 004 board.
58  //
59  ////////////////////////////////////////////////////////////////////////////
60
61  module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
62                 ac97_bit_clock,
63
```

```
64              vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
65              vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
66              vga_out_vsync,
67
68              tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
69              tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
70              tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
71
72              tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
73              tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
74              tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
75              tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
76
77              ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
78              ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
79
80              ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
81              ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
82
83              clock_feedback_out, clock_feedback_in,
84
85              flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
86              flash_reset_b, flash_sts, flash_byte_b,
87
88              rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
89
90              mouse_clock, mouse_data, keyboard_clock, keyboard_data,
91
92              clock_27mhz, clock1, clock2,
93
94              disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
95              disp_reset_b, disp_data_in,
96
97              button0, button1, button2, button3, button_enter, button_right,
98              button_left, button_down, button_up,
99
100             switch,
101
102             led,
103
104             user1, user2, user3, user4,
105
106             daughtercard,
107
108             systemace_data, systemace_address, systemace_ce_b,
109             systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
110
111             analyzer1_data, analyzer1_clock,
112             analyzer2_data, analyzer2_clock,
113             analyzer3_data, analyzer3_clock,
114             analyzer4_data, analyzer4_clock);
115
116     output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
117     input  ac97_bit_clock, ac97_sdata_in;
118
119     output [7:0] vga_out_red, vga_out_green, vga_out_blue;
120     output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
121         vga_out_hsync, vga_out_vsync;
122
123     output [9:0] tv_out_ycrcb;
124     output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
125         tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
126         tv_out_subcar_reset;
```

```
127
128     input  [19:0] tv_in_ycrcb;
129     input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
130         tv_in_hff, tv_in_aff;
131     output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
132         tv_in_reset_b, tv_in_clock;
133     inout  tv_in_i2c_data;
134
135     inout  [35:0] ram0_data;
136     output [18:0] ram0_address;
137     output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
138     output [3:0] ram0_bwe_b;
139
140     inout  [35:0] ram1_data;
141     output [18:0] ram1_address;
142     output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
143     output [3:0] ram1_bwe_b;
144
145     input  clock_feedback_in;
146     output clock_feedback_out;
147
148     inout  [15:0] flash_data;
149     output [23:0] flash_address;
150     output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
151     input  flash_sts;
152
153     output rs232_txd, rs232_rts;
154     input  rs232_rxd, rs232_cts;
155
156     input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
157
158     input  clock_27mhz, clock1, clock2;
159
160     output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
161     input  disp_data_in;
162     output  disp_data_out;
163
164     input  button0, button1, button2, button3, button_enter, button_right,
165         button_left, button_down, button_up;
166     input  [7:0] switch;
167     output [7:0] led;
168
169     output [31:0] user1, user4, user2;
170     input [31:0] user3;
171
172     inout [43:0] daughtercard;
173
174     inout  [15:0] systemace_data;
175     output [6:0]  systemace_address;
176     output systemace_ce_b, systemace_we_b, systemace_oe_b;
177     input  systemace_irq, systemace_mpbrdy;
178
179     output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
180             analyzer4_data;
181     output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;
182
183
184     //output [9:0] video_data;
185     //////////////////////////////////////////////////////////////////////////
186     //
187     // I/O Assignments
188     //
189     //////////////////////////////////////////////////////////////////////////
```

```
190
191     // Audio Input and Output
192     assign beep= 1'b0;
193     assign audio_reset_b = 1'b0;
194     assign ac97_synch = 1'b0;
195     // ac97_sdata_out and ac97_sdata_out are inputs;
196
197     // VGA Output
198     assign vga_out_red = 10'h0;
199     assign vga_out_green = 10'h0;
200     assign vga_out_blue = 10'h0;
201     assign vga_out_sync_b = 1'b1;
202     assign vga_out_blank_b = 1'b1;
203     assign vga_out_pixel_clock = 1'b0;
204     assign vga_out_hsync = 1'b0;
205     assign vga_out_vsync = 1'b0;
206
207     // SRAMs
208     assign ram0_data = 36'hZ;
209     assign ram0_address = 19'h0;
210     assign ram0_adv_ld = 1'b0;
211     assign ram0_clk = 1'b0;
212     assign ram0_cen_b = 1'b1;
213     assign ram0_ce_b = 1'b1;
214     assign ram0_oe_b = 1'b1;
215     assign ram0_we_b = 1'b1;
216     assign ram0_bwe_b = 4'hF;
217     assign ram1_data = 36'hZ;
218     assign ram1_address = 19'h0;
219     assign ram1_adv_ld = 1'b0;
220     assign ram1_clk = 1'b0;
221     assign ram1_cen_b = 1'b1;
222     assign ram1_ce_b = 1'b1;
223     assign ram1_oe_b = 1'b1;
224     assign ram1_we_b = 1'b1;
225     assign ram1_bwe_b = 4'hF;
226     assign clock_feedback_out = 1'b0;
227     // clock_feedback_in is an input
228
229     // Flash ROM
230     assign flash_data = 16'hZ;
231     assign flash_address = 24'h0;
232     assign flash_ce_b = 1'b1;
233     assign flash_oe_b = 1'b1;
234     assign flash_we_b = 1'b1;
235     assign flash_reset_b = 1'b0;
236     assign flash_byte_b = 1'b1;
237     // flash_sts is an input
238
239     // RS-232 Interface
240     assign rs232_txd = 1'b1;
241     assign rs232_rts = 1'b1;
242     // rs232_rxd and rs232_cts are inputs
243
244     // PS/2 Ports
245     // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs
246
247     // Daughtercard Connectors
248     assign daughtercard = 44'hZ;
249
250     // SystemACE Microprocessor Port
251     assign systemace_data = 16'hZ;
252     assign systemace_address = 7'h0;
```

```
253      assign systemace_ce_b = 1'b1;
254      assign systemace_we_b = 1'b1;
255      assign systemace_oe_b = 1'b1;
256      // systemace_irq and systemace_mpbrdy are inputs
257
258
259      assign analyzer4_data = 16'h0;
260      assign analyzer4_clock = 1'b1;
261
262      ////////////////////////////////////////////////////////////////
263      ///Video Implementation
264      ////////////////////////////////////////////////////////////////
265
266      wire reset;
267      SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(reset),
268                .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
269      defparam reset_sr.INIT = 16'hFFFF;
270
271      PULLUP pu_in (.O(tv_in_i2c_data));
272      PULLUP pu_out (.O(tv_out_i2c_data));
273
274      wire [9:0] video_data, tv_out_ycrcb, lum_data, x_min, x_max, x_count;   //y_max, y_min,
275      wire [7:0] y_max, y_min;
276      wire [9:0] line_count, real_xmin, real_xmax;
277      wire[3:0] position, pos;
278      wire [2:0] find_state;
279      wire odd_start, toggle, frame_done;
280      wire edge_state;
281      wire tv_in_line_clock1, reset_sync;
282
283      mod_sync sync(.clk(tv_in_line_clock1), .in(reset), .out(reset_sync)); //reset synchroni
    zer to video interface
284
285      //video module that outputs camera input
286      video_test video(.reset(reset), .clock_27mhz(clock_27mhz), .tv_out_ycrcb(tv_out_ycrcb),
     .tv_out_reset_b(tv_out_reset_b), .tv_out_clock(tv_out_clock),
287               .tv_out_i2c_clock(tv_out_i2c_clock), .tv_out_i2c_data(tv_out_i2c_data), .tv_out
    _pal_ntsc(tv_out_pal_ntsc),
288               .tv_out_hsync_b(tv_out_hsync_b), .tv_out_vsync_b(tv_out_vsync_b), .tv_out_blank
    _b(tv_out_blank_b),
289               .tv_out_subcar_reset(tv_out_subcar_reset), .tv_in_ycrcb(tv_in_ycrcb[19:10]), .t
    v_in_data_valid(tv_in_data_valid),
290               .tv_in_line_clock1(tv_in_line_clock1), .tv_in_line_clock2(tv_in_line_clock2), .
    tv_in_aef(tv_in_aef), .tv_in_hff(tv_in_hff),
291               .tv_in_aff(tv_in_aff), .tv_in_i2c_clock(tv_in_i2c_clock), .tv_in_i2c_data(tv_in
    _i2c_data), .tv_in_fifo_read(tv_in_fifo_read),
292               .tv_in_fifo_clock(tv_in_fifo_clock), .tv_in_iso(tv_in_iso), .tv_in_reset_b(tv_i
    n_reset_b), .tv_in_clock(tv_in_clock));
293
294      //video interface top level module
295      video_top2 top(.clk(tv_in_line_clock1), .reset(reset_sync), .data(tv_in_ycrcb[19:10]),
    .video_data(video_data),
296                     .odd_start(odd_start),  .line_count(line_count),
297                     .find_state(find_state), .lum_data(lum_data), .switch(toggle), .edge_stat
    e(edge_state),
298                     .x_max(x_max), .x_min(x_min), .y_max(y_max), .y_min(y_min), .frame_done(f
    rame_done),
299                     .x_count(x_count), .position(position), .pos(pos));
300
301   wire [7:0] xin, yin, zin, x, y, z, xjoy, yjoy, q1, q2, q3, q4;
302   wire jump, crouch;
303   wire status1, status2, status3, statussync1, statussync2, statussync3;
304   wire resetsync;
```

```
305   wire [2:0] control;
306   wire CE, CS;
307   wire adstart, adbusy, dastart, dabusy, sample, adload, daload;
308   wire xgo, ygo, xdir, ydir, lclick;
309   wire xsig1, xsig2, ysig1, ysig2;
310   wire [639:0] dots;
311   wire [7:0] xmove, ymove;
312
313   scroll scroll(resetsync, clock_27mhz, dots);
314   display display(resetsync, clock_27mhz, disp_blank, disp_clock, disp_rs, disp_ce_b, disp_r
      eset_b, disp_data_out, dots);
315   bigmajor bigmajor1 (.adstart(adstart), .adbusy(adbusy), .dastart(dastart), .dabusy(dabusy), .xac
316   ad ad1 (.status1(statussync1), .status2(statussync2), .status3(statussync3), .control(cont
      rol), .adload(adload), .start(adstart), .busy(adbusy), .reset(resetsync), .clk(clock_27mhz
      ));
317   da da1 (.CEbar(CE), .CSbar(CS), .daload(daload), .start(dastart), .busy(dabusy), .reset(re
      setsync), .clk(clock_27mhz));
318   sampledivider sampledivider1 (.clk(clock_27mhz), .reset(resetsync), .sample(sample));
319   synchronizer synchronizer1 (.reset(reset), .status1(status1), .status2(status2), .status3(
      status3), .resetsync(resetsync), .statussync1(statussync1), .statussync2(statussync2), .st
      atussync3(statussync3), .clock(clock_27mhz));
320   adreg adreg1 (.da(xin), .qa(x), .db(yin), .qb(y), .dc(zin), .qc(z), .clk(clock_27mhz), .lo
      ad(adload));
321   dareg dareg1 (.d1(xjoy), .d2(yjoy), .d3(xmove), .d4(ymove), .q1(q1), .q2(q2), .q3(q3), .q4
      (q4), .clk(clock_27mhz), .load(daload));
322
323   photo2 photo2(.xsig1(xsig1), .xsig2(xsig2), .ysig1(ysig1), .ysig2(ysig2), .gox(xgo), .goy(
      ygo), .ydir(ydir), .xdir(xdir), .clk(clock_27mhz), .reset(resetsync));
324
325   //inputs
326   assign xin = user3[7:0];
327   assign yin = user3[15:8];
328   assign zin = user3[23:16];
329   assign status1 = user3[31];
330   assign status2 = user3[30];
331   assign status3 = user3[29];
332
333   //outputs
334   assign user4[7:0] = q1[7:0];
335   assign user4[15:8] = q2[7:0];
336   assign user4[23:16] = q3[7:0];
337   assign user1[7:0] = q4[7:0];
338   assign user2[0] = CS;
339   assign user2[1] = CE;
340   assign user1[31:29] = control[2:0];
341   assign led = ~{4'b0000, pos[3:0]};
342   assign user2[7] = jump;
343   assign user2[8] = ~crouch;
344
345   assign user2[2] = xgo;
346   assign user2[3] = ygo;
347   assign user2[4] = xdir;
348   assign user2[5] = ydir;
349   assign user2[6] = lclick;
350
351   // synthesis attribute keep of clock_27mhz is true;
352   // synthesis attribute keep of tv_in_line_clock1 is true;
353     //synthesis attribute period of clock_27mhz is 37ns;
354     //synthesis attribute period of tv_in_line_clock1 is 37ns;
355
356   endmodule
357
358
```

```
1   //David Dryjanski and Andrew Pinkham
2   //Enhanced Gaming and Pointing - 6.111 Spring 2005
3
4   module mapper(clk, reset, x_min, x_max, y_max, y_min, position, pos, hold_val);
5
6   //The mapper module takes in the the x/y values from edge_detect and maps them into a dire
    ctional box.
7
8   input clk, reset, hold_val;
9   input [9:0] x_min, x_max;     //y_max, y_min
10  input [7:0] y_max, y_min;
11  output [3:0] position, pos;
12  reg [3:0] position, pos;
13  reg [9:0] x_crit, x_dist;     //y_crit, , y_dist
14  reg [7:0] y_crit, y_dist;
15  parameter L = 240;
16  parameter R = 480;     //divisions of the screen that correspond to the 9 directional boxe
    s.
17  parameter U = 75;
18  parameter D = 125;    //150
19  parameter x_mid = 360;
20  parameter y_mid = 122;
21
22  always @ (posedge clk)
23  begin
24  if (reset)
25  begin
26       position <= 0;
27       pos <= 0;
28  end
29  else begin
30
31  //if ((y_max - y_mid) > (y_mid - y_min))
32  //              y_crit <= y_max;
33  //else y_crit <= y_min;
34
35  //if ((x_max - x_mid) > (x_mid - x_min))
36  //              x_crit <= x_max;
37  //else x_crit <= x_min;
38  y_crit <= (y_min + y_max) / 2;
39  x_crit <= (x_min + x_max) / 2;
40
41  //mapping of the point to a box
42  if (x_crit < L)
43       begin
44       if (y_crit < U) position <= 1;
45       else if (y_crit > D) position <= 6;
46       else position <= 4;
47       end
48  else if (x_crit > R)
49       begin
50       if (y_crit < U) position <= 3;
51       else if (y_crit > D) position <= 8;
52       else position <= 5;
53       end
54  else begin
55       if (y_crit < U) position <= 2;
56       else if (y_crit > D) position <= 7;
57       else position <= 0;
58       end
59
60  pos <= (hold_val) ? position : pos; //pos gets set at the end of each frame
61  end
```

```
62    end
63    endmodule
64    /*   case (position)
65         1:
66              begin
67              x_dist <= L-x_crit;
68              y_dist <= U-y_crit;
69              end
70         4:
71              begin
72              x_dist <= L-x_crit;
73              y_dist <= 0;
74              end
75         6:
76              begin
77              x_dist <= L-x_crit;
78              y_dist <= y_crit-D;
79              end
80         3:
81              begin
82              x_dist <= x_crit-R;
83              y_dist <= U-y_crit;
84              end
85         5:
86              begin
87              x_dist <= x_crit-R;
88              y_dist <= 0;
89              end
90         8:
91              begin
92              x_dist <= x_crit-R;
93              y_dist <= y_crit-D;
94              end
95         2:
96              begin
97              x_dist <= 0;
98              y_dist <= U-y_crit;
99              end
100        7:
101             begin
102             x_dist <= 0;
103             y_dist <= y_crit-D;
104             end
105       endcase
106
107     if((x_dist > fast) || (y_dist > fast))
108             speed <= 2;
109       else if ((x_dist < slow) && (y_dist < slow))
110             speed <= 0;
111             else speed <= 1;
112       end
113       */
114
115
```

```
1    ////////////////////////////////
2    // Name: Andrew Pinkham
3    // Date: 4/2/2005
4    // Module: phototransistor emulation FSM
5    ////////////////////////////////
6
7    module photo2(xsig1, xsig2, ysig1, ysig2, gox, goy, ydir, xdir, clk, reset);
8
9         //xticks is number of ticks to go in horizontal axis, xdir is 0 for left and 1 for ri
     ght
10        //yticks is num ticks to go in vertical axis, ydir is 0 for down and 1 for up
11        output xsig1, xsig2, ysig1, ysig2;
12        input clk, reset, ydir, xdir, gox, goy;
13
14        reg [19:0] big_count, small_count;
15
16        reg big_cycle, small_cycle;
17
18        reg xsig1, xsig2, ysig1, ysig2;
19
20        reg [3:0] next1, next2;
21
22   //big_speed is the period of big_cycle that determines how fast the ticks ocurr
23   //big_speed is changed depending on how fast you want the mouse to move
24
25   //small_speed is the period of small_cycle that determines how fast the phototransistors g
     et pulsed
26   //small_speed is set so that it works and then doesnt get changed unless it stops working
27
28
29   always @ (posedge small_cycle) begin
30        if (reset == 1) next1 <= 0;
31
32        case (next1)
33        //wait for big_cycle to signal a tick
34        0:
35        begin
36             xsig1 <= 1;
37             xsig2 <= 1;
38             if ((gox == 1) && (big_cycle == 1) && (reset == 0)) next1 <= 1;
39             else next1 <= 0;
40        end
41        //turn one off, leave one on depending on which direction you want it to move
42        1:
43        begin
44        if (xdir == 1) begin
45             xsig1 <= 0;
46             xsig2 <= 1;
47             next1 <= 2;
48             end
49        else begin
50             xsig1 <= 1;
51             xsig2 <= 0;
52             next1 <= 2;
53             end
54        end
55        //turn both off
56        2:
57        begin
58             xsig1 <= 0;
59             xsig2 <= 0;
60             next1 <= 3;
61        end
```

```
62          //turn both on
63          3:
64          begin
65               xsig1 <= 1;
66               xsig2 <= 1;
67               next1 <= 0;
68          end
69          endcase
70     end
71
72
73
74     always @ (posedge small_cycle) begin
75          if (reset == 1) next2 <= 0;
76
77          case (next2)
78          //wait for big_cycle to signal a tick
79          0:
80          begin
81               ysig1 <= 1;
82               ysig2 <= 1;
83               if ((goy == 1) && (big_cycle == 1) && (reset == 0)) next2 <= 1;
84               else next2 <= 0;
85          end
86          //turn one off, leave one on depending on which direction you want it to move
87          1:
88          begin
89          if (ydir == 1) begin
90               ysig1 <= 0;
91               ysig2 <= 1;
92               next2 <= 2;
93               end
94          else begin
95               ysig1 <= 1;
96               ysig2 <= 0;
97               next2 <= 2;
98               end
99          end
100         //turn both off
101         2:
102         begin
103              ysig1 <= 0;
104              ysig2 <= 0;
105              next2 <= 3;
106         end
107         //turn both on
108         3:
109         begin
110              ysig1 <= 1;
111              ysig2 <= 1;
112              next2 <= 0;
113         end
114         endcase
115    end
116
117
118    //generate small_cycle clock signal, check reset, generate gox and goy enables
119    always @ (posedge clk)
120    begin
121    if (reset == 1) begin
122         big_count <= 0;
123         small_count <= 0;
124         end
```

```
125
126   if (small_count >= 500)       //500-dependent on digital logic in mouse
127        begin
128        small_cycle <= 1;
129        small_count <= 0;
130        end
131   else begin
132        small_cycle <= 0;
133        small_count <= small_count + 1;
134        end
135
136   if (big_count >= 70)          //70 is big speed
137        begin
138        big_cycle <= 1;
139        big_count <= 0;
140        end
141   else begin
142        big_cycle <= 0;
143        big_count <= big_count + 1;
144        end
145   end
146
147   endmodule
148
149
```

```
1     ////////////////////////////////
2     // Name: Andrew Pinkham
3     // Date: 4/2/2005
4     // Module: scrolling message
5     ////////////////////////////////
6
7     module scroll(reset, clk, dots);
8          input clk, reset;
9          output [639:0] dots;
10
11         reg [639:0] dots;
12
13         reg [24:0] count;
14
15         reg [1599:0] messagetemp;
16         wire [1599:0] message;
17
18    always @ (posedge clk)
19    begin
20    if (reset)
21         begin
22         count <= 0;
23         end
24    else if (count == 2000000)
25         begin
26         count <= 0;
27         if (messagetemp <= 0) begin
28              messagetemp[1599:0] <= message[1599:0];
29              end
30         else begin
31              dots[639:0] <= messagetemp[1599:960];
32              messagetemp[1599:0] <= {messagetemp[1560:0], 40'b00000000_00000000_00000000_0000
      0000_00000000};
33              end
34         end
35    else
36         begin
37         count <= count + 1;
38         end
39    end
40
41
42       assign message = {    40'b00000000_00000000_00000000_00000000_00000000, //_
43                             40'b00000000_00000000_00000000_00000000_00000000, //_
44                             40'b00000000_00000000_00000000_00000000_00000000, //_
45                             40'b00000000_00000000_00000000_00000000_00000000, //_
46                             40'b00000000_00000000_00000000_00000000_00000000, //_
47                             40'b00000000_00000000_00000000_00000000_00000000, //_
48                             40'b00000000_00000000_00000000_00000000_00000000, //_
49                             40'b00000000_00000000_00000000_00000000_00000000, //_
50                             40'b00000000_00000000_00000000_00000000_00000000, //_
51                             40'b00000000_00000000_00000000_00000000_00000000, //_
52                             40'b00000000_00000000_00000000_00000000_00000000, //_
53                             40'b00000000_00000000_00000000_00000000_00000000, //_
54                        40'b01111111_01001001_01001001_01001001_01000001, // E
55                        40'b01111111_00000010_00000100_00001000_01111111, //N
56                        40'b01111111_00011000_00011000_00011000_01111111, //H
57                        40'b01111110_00001001_00001001_00001001_01111110, // A
58                        40'b01111111_00000010_00000100_00001000_01111111, //N
59                        40'b00111110_01000001_01000001_01000001_00100010, // C
60                        40'b01111111_01001001_01001001_01001001_01000001, // E
61                        40'b01111111_01000001_01000001_01000001_00111110, // D
62                        40'b00000000_00000000_00000000_00000000_00000000, //_
```

```
63                      40'b00111110_01000001_01001001_01001001_00111010, // G
64                      40'b01111110_00001001_00001001_00001001_01111110, // A
65                      40'b01111111_00000010_00001100_00000010_01111111, // M
66                      40'b00000000_01000001_01111111_01000001_00000000, // I
67                      40'b01111111_00000010_00000100_00001000_01111111, //N
68                      40'b00111110_01000001_01001001_01001001_00111010, // G
69                      40'b00000000_00000000_00000000_00000000_00000000, //_
70                      40'b01111110_00001001_00001001_00001001_01111110, // A
71                      40'b01111111_00000010_00000100_00001000_01111111, //N
72                      40'b01111111_01000001_01000001_01000001_00111110, // D
73                      40'b00000000_00000000_00000000_00000000_00000000, //_
74                      40'b01111111_00001001_00001001_00001001_00000110, // P
75                      40'b00111110_01000001_01000001_01000001_00111110, // O
76                      40'b00000000_01000001_01111111_01000001_00000000, // I
77                      40'b01111111_00000010_00000100_00001000_01111111, //N
78                      40'b00000001_00000001_01111111_00000001_00000001, // T
79                      40'b00000000_01000001_01111111_01000001_00000000, // I
80                      40'b01111111_00000010_00000100_00001000_01111111, //N
81                      40'b00111110_01000001_01001001_01001001_00111010 // G
82                      };
83
84      endmodule
```

```
1   //David Dryjanski and Andrew Pinkham
2   //Enhanced Gaming and Pointing - 6.111 Spring 2005
3
4   module video_top2(clk, reset, data, video_data, odd_start, find_state, line_count,
5             x_max, x_min, y_max, y_min, lum_data, switch, edge_state, frame_done,
6             x_count, position, pos);
7
8   //This is the top level module for the video interface which performs edge detection on th
    e player's movement
9
10  input clk, reset;
11  input [9:0] data;
12  output [3:0] edge_state, position, pos;
13  output odd_start, switch, frame_done;
14  output [9:0] video_data, line_count, lum_data, x_count, x_max, x_min;//, y_max, y_min
15  output [7:0] y_max, y_min;
16  wire clk, reset, odd_start, switch, frame_done;
17  output [2:0] find_state;
18  wire [9:0] line_count, video_data;
19
20  find_frame2 finder(.clk(clk), .reset(reset), .start_odd_load(odd_start),
21        .data_in(data), .data_out(video_data), .state(find_state), .line_count(line_count));
22
23  edge_detect2 edger(.clk(clk), .reset(reset), .data(video_data), .odd_lines(odd_start), .x_
    max(x_max), .x_min(x_min), .y_max(y_max), .y_min(y_min),
24       .lum_data(lum_data), .switch(switch), .state(edge_state),
25       .x_count(x_count), .line_count(line_count), .frame_done(frame_done));
26
27  mapper mapper(.clk(clk), .reset(reset), .x_min(x_min), .x_max(x_max), .y_max(y_max), .y_mi
    n(y_min),
28        .position(position), .hold_val(frame_done), .pos(pos));
29
30  endmodule
31
```