Contents

internal.v

```
1    `timescale 1ns / 1ps
2    `include "defines.v"
3    module internal(
4        reset, clock,
5
6        model_select,
7
8        paddle_up, paddle_down, paddle_speed, ball_initial_speed, // game inputs
9
10       ram0_data, ram0_address, ram0_we_b, // ram 0 buss
11       ram1_data, ram1_address, ram1_we_b, // ram 1 buss
12
13       mouse0_clock, mouse0_data, // ps/2 0 buss
14       mouse1_clock, mouse1_data, // ps/2 1 buss
15       buttons,
16
17
18       vga_rgb, vga_blank_b, vga_hsync, vga_vsync // vga outputs
19       );
20
21       input clock;
22       input reset;
23
24       input [3:0] model_select;
25
26       input paddle_up;
27       input paddle_down;
28       input [3:0] paddle_speed;
29       input [3:0] ball_initial_speed;
30
31       inout [35:0] ram0_data;
32       output [18:0] ram0_address;
33       output ram0_we_b;
34
35       inout [35:0] ram1_data;
36       output [18:0] ram1_address;
37       output ram1_we_b;
38
39       input mouse0_clock;
40       input mouse0_data;
41       input mouse1_clock;
42       input mouse1_data;
43       input [`NUM_BUTTONS-1:0] buttons;
44
45       output [23:0] vga_rgb;
46       output vga_blank_b;
47       output vga_hsync;
48       output vga_vsync;
49
50       wire [23:0] read_rgb, df_rgb;
51       wire blank_b, hsync, vsync, enable, blanking;
52       wire [9:0]  pixel_count, line_count;
53       vga vga1(
54           .reset(reset),
55           .pixel_clock(clock),
56           .blank_b(blank_b),
57           .hsync(hsync),
58           .vsync(vsync),
59           .pixel_count(pixel_count),
60           .line_count(line_count),
61           .blanking(blanking),
62           .enable(enable));
63
```

internal.v

```
64       delay #(5) dy1 (reset, clock, blank_b, vga_blank_b);
65       delay #(5) dy2 (reset, clock, hsync, vga_hsync);
66       delay #(5) dy3 (reset, clock, vsync, vga_vsync);
67
68       wire [9:0] paddle_y;
69       wire [9:0] ball_x;
    display_field df(
70       wire [9:0] ball_y;
71
72           (reset, pixel_count, line_count,
73            paddle_y, ball_x, ball_y, 24'b0, df_rgb);
74       add_rgb #(0, 1) ar
75           (reset, clock, read_rgb,
76           df_rgb & (~{(24){model_select[0]||model_select[1]}}), vga_rgb);
77
78       wire switch_buffer;
79       wire [9:0] write_pixel_count, write_line_count;
80       wire [11:0] write_z;
81       wire [23:0] write_rgb;
82       wire pixel_next;
83       screen_buffer sb(
84           .reset(reset),
85           .clock(clock),
86           .switch_buffer(switch_buffer),
87
88           .write_pixel_count(write_pixel_count),
89           .write_line_count(write_line_count),
90           .write_z(write_z),
91           .write_rgb(write_rgb),
92           .write_noop(pixel_noop),
93           .write_next(pixel_next),
94
95           .read_pixel_count(pixel_count),
96           .read_line_count(line_count),
97           .read_rgb(read_rgb),
98
99           .ram0_data(ram0_data),
100          .ram0_address(ram0_address),
101          .ram0_we_b(ram0_we_b), // ram 0 buss
102
103          .ram1_data(ram1_data),
104          .ram1_address(ram1_address),
105          .ram1_we_b(ram1_we_b) // ram 1 buss
106          );
107
108      wire [`TRIANGLE_BITS+`NORMAL_BITS-1:0] triangle_data;
109      wire triangle_empty, triangle_noop, triangle_next, next_frame;
110      triangle_source ts (reset,
111                          clock,
112                          model_select,
113                          triangle_data,
114                          triangle_empty,
115                          triangle_noop,
116                          triangle_next,
117                          next_frame);
118
119
120      controller cr(
121          .reset(reset),
122          .clock(clock),
123          .blanking(blanking),
124          .empty(pixel_empty),
125          .next_frame(next_frame),
126          .switch_buffer(switch_buffer));
```

Page: 1

Page: 2

```
       internal.v
127
128
129
130     fsm fsm1 (.reset(reset),
131               .clock(clock),
132               .enable(enable),
133               .speed(ball_initial_speed),
134               .paddle_speed(paddle_speed),
135               .up_sync(paddle_up),
136               .down_sync(paddle_down),
137               .paddle_y(paddle_y),
138               .ball_x(ball_x),
139               .ball_y(ball_y));
140
141     wire [`COORD_BITS*3-1:0] translation_vector, light_vector,
142                              ball_vector, paddle_vector;
143     wire [9*`TRIG_OUT_BITS-1:0] rotation_matrix, rotor_matrix;
144     wire [`COORD_BITS-1:0] zoom;
145     orienter ornt(
146         reset,
147         clock,
148         mouse0_clock, mouse0_data, mouse1_clock, mouse1_data, buttons,
149         enable, switch_buffer,
150         paddle_y, ball_x, ball_y,
151         translation_vector,
152         rotation_matrix,
153         light_vector,
154         zoom,
155         rotor_matrix,
156         ball_vector,
157         paddle_vector);
158
159     wire [`TRIANGLE_BITS-1:0] out_triangle;
160     wire out_noop, out_empty, out_next;
161
162     triangle_pipeline tp(
163         .reset(reset),
164         .clock(clock),
165         .in_triangle(triangle_data),
166         .in_noop(triangle_noop),
167         .in_empty(triangle_empty),
168         .in_next(triangle_next),
169         .out_triangle(out_triangle),
170         .out_noop(out_noop),
171         .out_empty(out_empty),
172         .out_next(out_next),
173         .translation_vector(translation_vector),
174         .rotation_matrix(rotation_matrix),
175         .light_vector(light_vector),
176         .zoom(zoom),
177         .rotor_matrix(rotor_matrix),
178         .ball_vector(ball_vector),
179         .paddle_vector(paddle_vector));
180
181     triangle_shader pshdr(
182         .reset(reset),
183         .clock(clock),
184         .triangle_data(out_triangle),
185         .triangle_empty(out_empty),
186         .triangle_noop(out_noop),
187         .triangle_next(out_next),
188         .pixel_data({write_pixel_count,
189                      write_line_count,
```

```
                         write_z, write_rgb}),
190
191         .pixel_empty(pixel_empty),
192         .pixel_noop(pixel_noop),
193         .pixel_next(pixel_next));
194 endmodule
195
196 module add_rgb(reset, clock, a, b, c);
197     parameter A_SHIFT = 1;
198     parameter B_SHIFT = 1;
199
200     localparam T_WIDTH = 9;
201     input reset, clock;
202     input [23:0] a, b;
203     output [23:0] c;
204
205     reg [23:0] c;
206     wire [23:0] next_c;
207     wire [7:0] a_r, a_g, a_b,
208                b_r, b_g, b_b,
209                c_r, c_g, c_b;
210
211     assign {a_r, a_g, a_b} = a;
212     assign {b_r, b_g, b_b} = b;
213     assign next_c = {c_r, c_g, c_b};
214
215     wire [T_WIDTH-1:0] t_r, t_g, t_b;
216
217     assign t_r = a_r[7-A_SHIFT:0] + b_r[7-B_SHIFT:0];
218     assign t_g = a_g[7-A_SHIFT:0] + b_g[7-B_SHIFT:0];
219     assign t_b = a_b[7-A_SHIFT:0] + b_b[7-B_SHIFT:0];
220
221     assign c_r = t_r > 255 ? 255 : t_r[7:0];
222     assign c_g = t_g > 255 ? 255 : t_g[7:0];
223     assign c_b = t_b > 255 ? 255 : t_b[7:0];
224
225     always @(posedge clock)
226         if (reset)
227             c <= 24'b0;
228         else
229             c <= next_c;
230 endmodule
231
232 module controller(reset, clock, blanking, empty, next_frame, switch_buffer);
233     input reset;
234     input clock;
235     input blanking;
236     input empty;
237     output next_frame;
238     output switch_buffer;
239
240     reg [2:0] state;
241     localparam IDLE =    3'd0;
242     localparam WAIT_FOR_BLANKING = 3'd1;
243     localparam SEND_SIGNALS = 3'd2;
244     localparam NEITHER_LOW = 3'd3;
245     localparam EMPTY_LOW = 3'd4;
246     localparam BLANKING_LOW = 3'd5;
247
248     reg [2:0] next_state;
249
250     always @ (posedge clock)
251         if (reset)
252             state <= IDLE;
```

Page: 3

Page: 4

next_state

internal.v
next_state

```
253          else
254              state <= next_state;
255
256      always @ (state or blanking or empty) begin
257          next_state = state;
258          case (state)
259              IDLE:
260                  if (empty)
261                      if (blanking)
262                                  = SEND_SIGNALS;
263                      else
264                                  = WAIT_FOR_BLANKING;
265
266
267              WAIT_FOR_BLANKING:
268                  if (blanking)
269                                  = SEND_SIGNALS;
270              SEND_SIGNALS:
271                  if (!empty && !blanking)
272                                  = IDLE;
273                  else if (!empty)
274                                  = EMPTY_LOW;
275                  else if (!blanking)
276                                  = BLANKING_LOW;
277                  else
278                                  = NEITHER_LOW;
279              NEITHER_LOW:
280                  if (!empty && !blanking)
281                                  = IDLE;
282                  else if (!empty)
283                                  = EMPTY_LOW;
284                  else if (!blanking)
285                                  = BLANKING_LOW;
286              EMPTY_LOW:
287                  if (!blanking)
288                                  = IDLE;
289              BLANKING_LOW:
290                  if (!empty)
291                                  = IDLE;
292          endcase
293      end
294
295      assign next_frame = (state == SEND_SIGNALS);
296      assign switch_buffer = (state == SEND_SIGNALS);
297
298  endmodule
299
300  `timescale 1ns / 1ps
301  module delay(reset, clock, in, out);
302      input reset;
303      input clock;
304
305      parameter DELAY = 2;
306
307      input in;
308      output out;
309
310      reg [DELAY-1:0] data;
311
312      assign out = data[DELAY-1];
313
314      always @ (posedge clock) begin
315          if (reset)
```

Page: 5

internal.v

```
316              data <= {DELAY{1'b0}};
317          else
318              data <= {data[DELAY - 2: 0], in};
319      end
320
321  endmodule
322
323  //  VGA Module from Problem Set 3
324  module vga (reset, pixel_clock, sync_b,
325          blank_b, hsync, vsync, pixel_count,
326          line_count, enable, blanking);
327
328      input reset; // system reset
329      input pixel_clock; // 31.5 MHz pixel clock
330      output sync_b; // hardwired to Vdd
331      output blank_b; // composite blank
332      output hsync; // horizontal sync
333      output vsync; // vertical sync
334      output [9:0] pixel_count; // number of the current pixel
335      output [9:0] line_count; // number of the current line
336      output enable;  // 1-clock enable once per screen refresh
337      output blanking;
338
339  //  // 640x480 75Hz parameters
340  //  parameter PIXELS = 800;
341  //  parameter LINES = 525;
342  //  parameter HACTIVE_VIDEO = 640;
343  //  parameter HFRONT_PORCH = 16;
344  //  parameter HSYNC_PERIOD = 96;
345  //  parameter HBACK_PORCH = 48;
346  //  parameter VACTIVE_VIDEO = 480;
347  //  parameter VFRONT_PORCH = 11;
348  //  parameter VSYNC_PERIOD = 2;
349  //  parameter VBACK_PORCH = 32;
350  //    640x480, 75Hz  31.500 640 16  96  48  480 11  2   32
351  //    640x480, 60Hz  25.175 640 16  96  48  480 11  2   31
352      // 640x480 60Hz parameters
353      parameter PIXELS = 800;
354      parameter LINES = 524;
355      parameter HACTIVE_VIDEO = 640;
356      parameter HFRONT_PORCH = 16;
357      parameter HSYNC_PERIOD = 96;
358      parameter HBACK_PORCH = 48;
359      parameter VACTIVE_VIDEO = 480;
360      parameter VFRONT_PORCH = 11;
361      parameter VSYNC_PERIOD = 2;
362      parameter VBACK_PORCH = 31;
363
364
365      // current pixel count
366      reg [9:0] pixel_count = 10'b0;
367      reg [9:0] line_count = 10'b0;
368
369      reg hsync = 1'b1;
370      reg vsync = 1'b1;
371      reg blank_b = 1'b1;
372      reg enable = 1'b0;
373      reg blanking;
374      wire sync_b; // connected to Vdd
375
376      wire pixel_clock;
377      wire [9:0] next_pixel_count;
378      wire [9:0] next_line_count;
```

Page: 6

```
      internal.v
379       wire next_enable;
380
381       always @ (posedge pixel_clock)
382          begin
383          if (reset)
384             begin
385                pixel_count <= 10'b0;
386                line_count <= 10'b0;
387                hsync <= 1'b1;
388                vsync <= 1'b1;
389                blank_b <= 1'b1;
390                enable <= 1'b0;
391             end
392          else
393             begin
394                pixel_count <= next_pixel_count;
395                line_count <= next_line_count;
396                hsync <=
397                   (next_pixel_count < HACTIVE_VIDEO + HFRONT_PORCH) |
398                   (next_pixel_count >= HACTIVE_VIDEO+HFRONT_PORCH+
399                   HSYNC_PERIOD);
400                vsync <=
401                   (next_line_count < VACTIVE_VIDEO+VFRONT_PORCH) |
402                   (next_line_count >= VACTIVE_VIDEO+VFRONT_PORCH+
403                VSYNC_PERIOD);
404                // this is the and of hblank and vblank
405                blank_b <=
406                   (next_pixel_count < HACTIVE_VIDEO) &
407                   (next_line_count < VACTIVE_VIDEO);
408                enable <= next_enable;
409                blanking <=
410                   (next_line_count > VACTIVE_VIDEO &&
411                   next_line_count < LINES);
412             end
413          end
414
415       // next state is computed with combinational logic
416       assign next_pixel_count =
417          (pixel_count == PIXELS-1) ?
418           10'h000 : pixel_count + 1'b1;
419       assign next_line_count =
420          (pixel_count == PIXELS-1) ?
421           (line_count == LINES-1) ? 10'h000 :
422           line_count + 1'b1 : line_count;
423       // can hardwire composite sync to Vdd.
424       assign sync_b = 1'b1;
425
426       assign next_enable =
427          (pixel_count == 1)
428          && (line_count == VACTIVE_VIDEO - 2);
429
430    endmodule
431
```

Page: 7

```
      screen_buffer.v
1     `timescale 1ns / 1ps
2     `include "defines.v"
3     module screen_buffer(
4         reset, clock,
5
6         switch_buffer,
7
8         write_pixel_count, write_line_count,
9         write_z, write_rgb, write_noop, write_next,
10
11        read_pixel_count, read_line_count, read_rgb,
12
13        ram0_data, ram0_address, ram0_we_b, // ram 0 buss
14        ram1_data, ram1_address, ram1_we_b // ram 1 buss
15        );
16        input reset;
17        input clock;
18
19        input switch_buffer;
20
21        input [9:0] write_pixel_count;
22        input [9:0] write_line_count;
23        input [11:0] write_z;
24        input [23:0] write_rgb;
25        input write_noop;
26        output write_next;
27
28        input [9:0] read_pixel_count;
29        input [9:0] read_line_count;
30        output [23:0] read_rgb;
31
32        inout [35:0] ram0_data;
33        output [18:0] ram0_address;
34        output ram0_we_b;
35
36        inout [35:0] ram1_data;
37        output [18:0] ram1_address;
38        output ram1_we_b;
39
40        // wiring
41        wire [35:0] read_data;
42        wire [18:0] read_address;
43        wire read_we_b;
44
45        wire [18:0] write_address;
46        wire write_we_b;
47        wire hold;
48
49        // state
50        reg ram_select;
51
52        always @(posedge clock)
53            if (reset)
54                ram_select <= 0;
55            else
56                ram_select <= switch_buffer ? !ram_select : ram_select;
57
58        assign {ram0_we_b, ram1_we_b} = ram_select ?
59                                        {read_we_b, write_we_b}:
60                                        {write_we_b, read_we_b};
61        assign {ram0_address, ram1_address} = ram_select ?
62                                        {read_address, write_address}:
63                                        {write_address, read_address};
```

Page: 1

```
      screen_buffer.v
 _____

 64
 65      assign read_data = ram_select ? ram0_data: ram1_data;
 66
 67      zbt_reader reader(reset, clock, read_pixel_count, read_line_count, read_rgb,
 68                        read_we_b,
 69                        read_address,
 70                        read_data);
 71
 72
 73      zbt_out out(reset, clock, write_pixel_count, write_line_count,
 74                  write_z, write_rgb, write_noop,
 75                  write_we_b, write_address, ram_select,
 76                  ram0_data, ram1_data, switch_buffer, write_next);
 77  endmodule
 78
 79  module zbt_out(reset, clock, pixel_count, line_count, pixel_z, rgb, noop,
 80                 ram_we_b, ram_address, ram_select,
 81                 ram0_data, ram1_data, switch_buffer, next);
 82
 83
 84      parameter DEFAULT_VALUE = {12'b111111111111, 24'b0};
 85      parameter MAX_ADDRESS = `SCREEN_HEIGHT * `SCREEN_WIDTH;
 86
 87      input reset;
 88      input clock;
 89      input [9:0] pixel_count;
 90      input [9:0] line_count;
 91      input [23:0] rgb;
 92      input [11:0] pixel_z;
 93      input noop;
 94      output next;
 95
 96      output ram_we_b;     // physical line to ram we_b
 97      output [18:0] ram_address;  // physical line to ram address
 98      input ram_select;
 99
100      inout  [35:0] ram0_data;    // physical line to ram data
101      inout  [35:0] ram1_data; // physical line to ram data
102
103      input switch_buffer;
104
105      // buffer the outputs
106      reg ram_we_b;
107      reg [18:0] ram_address;
108      reg [35:0] ram_data_out;
109      reg next;
110
111      // internal state
112      reg oe;
113      reg [1:0] state;
114
115      localparam CLEARING = 0;
116      localparam WRITING = 1;
117      localparam READING = 2;
118
119      // additional input
120      wire [35:0] ram_data_in = (ram_select) ? ram1_data: ram0_data;
121      assign ram1_data = (oe && ram_select) ? ram_data_out : {36{1'bZ}};
122      assign ram0_data = (oe && !ram_select) ? ram_data_out : {36{1'bZ}};
123
124      // shift registers
125      reg [35:0] ram_data_in0, ram_data_in1, ram_data_in2;
126      reg [18:0] ram_address0, ram_address1, ram_address2;
 _____

 Page: 2
```

```
      screen_buffer.v
 _____

127      reg [23:0] rgb0, rgb1, rgb2;
128      reg [11:0] pixel_z0, pixel_z1, pixel_z2;
129      reg noop0, noop1, noop2;
130
131      always @ (posedge clock) begin
132          {ram_data_in0, ram_data_in1, ram_data_in2}
133              <= {ram_data_in, ram_data_in0, ram_data_in1};
134          {ram_address0, ram_address1, ram_address2}
135              <= {ram_address, ram_address0, ram_address1};
136          {rgb0, rgb1, rgb2} <= {rgb, rgb0, rgb1};
137          {pixel_z0, pixel_z1, pixel_z2} <= {pixel_z, pixel_z0, pixel_z1};
138          {noop0, noop1, noop2} <= {noop, noop0, noop1};
139      end
140
141      // next state wires
142      reg next_we;
143      reg [18:0] next_address;
144      reg [35:0] next_data_out;
145      reg next_oe;
146      reg [1:0] next_state;
147
148      always @ (posedge clock)
149          if (reset) begin
150              ram_we_b <= 1'b1;
151              ram_address <= 19'b0;
152              ram_data_out <= 36'b0;
153              oe <= 1'b0;
154              state <= READING;
155          end else begin
156              ram_we_b <= !next_we;
157              ram_address <= next_address;
158              ram_data_out <= next_data_out;
159              oe <= next_oe;
160              state <= next_state;
161          end
162
163      always @ (state, ram_address, switch_buffer, line_count, pixel_count, oe,
164                ram_we_b, ram_data_out, noop, pixel_z2, ram_data_in0, rgb2,
165                noop1, ram_data_in) begin
166          next_state = state;
         next_state  next_address = ram_address;
167          next_oe = oe;
     next_data_out
168          next_we = !ram_we_b;
     next_oe
169          next_data_out = ram_data_out;
     next_we
170          next = 1'b0;
171          case (state)
     next_address      CLEARING:
172          next_data_out    begin
173          next_oe              next = 1'b0;
174          next_we              if (ram_address == MAX_ADDRESS+1) begin
175                                   = READING;
176                                       = DEFAULT_VALUE;
177                                   = 1'b1;
178                                   = 1'b0;
179                               end else begin
180                                       = ram_address + 1;
181                                       = DEFAULT_VALUE;
182                                   = 1'b1;
183                                   = 1'b1;
184                               end
185                           end
186                   WRITING:
187                   begin
 _____

 Page: 3
```

screen_buffer.v

```
190            next = 1'b1;
191            if (switch_buffer) begin
192                  = 1'b0;
193                     = CLEARING;
194                  = 19'b0;
195              next_oe = 1'b0;
196              next_we = 1'b1;
197            end else begin
198                  =
199                (line_count>=`SCREEN_HEIGHT ||
200                 pixel_count>=`SCREEN_WIDTH)
201                   ? -1 : line_count * `SCREEN_WIDTH + pixel_count;
202              next_oe = 1'b0;
203              next_we = 1'b0;
204                     = READING;
205          end
206        end
207        READING:
208        begin
209          next = 1'b0;
210          if (switch_buffer) begin
211                  = CLEARING;
212                  = 19'b0;
213              next_oe = 1'b0;
214              next_we = 1'b1;
215          end else begin
216              next_we = !noop0;//!noop;//1'b1;
217              next_oe = !noop2;//!noop1;//1'b1;
218                     = (pixel_z2 > ram_data_in[35:24])
219                 ? ram_data_in : {pixel_z2, rgb2};
220                     = WRITING;
221          end
222        end
223      endcase
224    end
225  endmodule
226
227  module zbt_reader(
228    reset, clock, pixel_count, line_count, read_rgb,
229    ram_we_b, ram_address, ram_data);
230    input clock;              // system clock
231    input reset;
232    input [9:0] pixel_count;
233    input [9:0] line_count;
234    output [23:0] read_rgb; // data read from memory
235
236    output ram_we_b;     // physical line to ram we_b
237    output [18:0] ram_address;  // physical line to ram address
238    input  [35:0] ram_data; // physical line to ram data
239
240    assign ram_we_b = 1'b1;
241    reg [18:0] ram_address;
242    reg [23:0] read_rgb, read_rgb0, read_rgb1;
243
244    wire [18:0] next_address =
245        (line_count>=`SCREEN_HEIGHT ||
246         pixel_count>=`SCREEN_WIDTH)
247        ? -1 : line_count * `SCREEN_WIDTH + pixel_count;
248    always @ (posedge clock)
249        if (reset) begin
250            ram_address <= 36'b0;
251            read_rgb <= 36'b0;
252            read_rgb0 <= 36'b0;
```

screen_buffer.v

```
253            read_rgb1 <= 36'b0;
254        end else begin
255            ram_address <= next_address;
256            {read_rgb, read_rgb0, read_rgb1}
257                <= {read_rgb0, read_rgb1, ram_data[23:0]};
258        end
259  endmodule
```

triangle_shader.v

```verilog
1    `timescale lns / 1ps
2    `include "defines.v"
3    module triangle_shader(
4        reset, clock, triangle_data, triangle_empty,
5        triangle_noop, triangle_next,
6        pixel_data, pixel_empty, pixel_noop, pixel_next);
7        // input
8        input reset;
9        input clock;
10       input [`TRIANGLE_BITS-1:0] triangle_data;
11       input triangle_empty;
12       input triangle_noop;
13       output triangle_next;
14
15       // output
16       output [`PIXEL_BITS-1:0] pixel_data;
17       output pixel_empty;
18       output pixel_noop;
19       input pixel_next;
20
21       // register some outputs
22       reg [`PIXEL_BITS-1:0] pixel_data;
23       reg pixel_empty;
24       reg triangle_next;
25
26       // register state
27       reg [1:0] state;
28       localparam RHOMBUS_A = 0;
29       localparam RHOMBUS_B = 1;
30       localparam NEXT = 2;
31
32       // inputs to rhombus shader
33       reg [`RHOMBUS_BITS-1:0] rhombus_data;
34       reg rhombus_noop;
35       wire rhombus_next;
36
37       // next state values
38       reg [1:0] next_state;
39       wire [`PIXEL_BITS-1:0] next_pixel_data;
40       wire [`POINT_BITS-1:0] next_point_data;
41       reg next_pixel_empty;
42       reg next_rhombus_noop;
43       reg [`RHOMBUS_BITS-1:0] next_rhombus_data;
44
45       rhombus_shader rs (reset, clock, rhombus_data, rhombus_noop, rhombus_next,
46                   next_point_data, pixel_noop, pixel_next);
47
48       // split up rhombus_painter inputs
49       wire [`RHOMBUS_BITS-1:0] rhombus_a, rhombus_b;
50       wire [`COORD_BITS-1:0] a_min_x, a_min_y, a_max_x, a_max_y,
51                        b_min_x, b_min_y, b_max_x, b_max_y,
52                        a_z, b_z;
53       wire [`LINE_BITS-1:0] a_left_line, a_right_line, b_left_line, b_right_line;
54       assign rhombus_a = {a_left_line, a_right_line,
55                       a_min_x, a_min_y, a_max_x, a_max_y, a_z};
56       assign rhombus_b = {b_left_line, b_right_line,
57                       b_min_x, b_min_y, b_max_x, b_max_y, b_z};
58
59       // split up triangle_data inputs
60       wire [`POINT_BITS-1:0] top_point, left_point,
61                        right_point, bottom_point, mid_point;
62       wire [`COORD_BITS-1:0]  top_x, top_y, top_z,
63                        left_x, left_y, left_z, right_x, right_y, right_z,
```

triangle_shader.v

```verilog
64                        bottom_x, bottom_y, bottom_z, mid_x, mid_y, mid_z;
65       wire [`COLOR_BITS-1:0] rgb;
66       assign {top_point, left_point, right_point, rgb} = triangle_data;
67       assign {top_x, top_y, top_z} = top_point;
68       assign {left_x, left_y, left_z} = left_point;
69       assign {right_x, right_y, right_z} = right_point;
70       reg [`COLOR_BITS-1:0] rgb0;
71
72       // find mid/bottom point
73       wire left_bellow_right;
74       signed_lt #(`COORD_BITS) slt (left_y, right_y, left_bellow_right);
75       assign {bottom_point, mid_point} = left_bellow_right ?
76           {left_point, right_point} : {right_point, left_point};
77       assign {bottom_x, bottom_y, bottom_z} = bottom_point;
78       assign {mid_x, mid_y, mid_z} = mid_point;
79
80       //assign min/max
81       wire [`COORD_BITS-1:0] min_x, max_x;
82       signed_outer3 #(`COORD_BITS) so3 (left_x, top_x, right_x, min_x, max_x);
83
84       assign a_min_x = min_x;
85       assign b_min_x = min_x;
86       assign a_max_x = max_x;
87       assign b_max_x = max_x;
88
89       assign a_max_y = top_y;
90       assign a_min_y = mid_y;
91       assign b_max_y = mid_y;
92       assign b_min_y = bottom_y;
93
94       // calculate average z (kind of)
95       wire [`COORD_BITS+1:0] z_sum;
96       wire [`COORD_BITS-1:0] average_z;
97       signed_add3 #(`COORD_BITS) sa3 (top_z, left_z, right_z, z_sum);
98       assign average_z = z_sum[`COORD_BITS+1:2];
99       assign a_z = average_z;
100      assign b_z = average_z;
101
102      // assign lines
103      assign a_left_line = {top_x, top_y, left_x, left_y};
104      assign a_right_line = {top_x, top_y, right_x, right_y};
105      assign b_left_line = left_bellow_right ?
106          {top_x, top_y, left_x, left_y} : {left_x, left_y, right_x, right_y};
107      assign b_right_line = left_bellow_right ?
108          {right_x, right_y, left_x, left_y} : {top_x, top_y, right_x, right_y};
109
110      //
111      always @ (posedge clock)
112          if (reset) begin
113              state <= NEXT;
114              rhombus_data <= {(`RHOMBUS_BITS){1'b0}};
115              rhombus_noop <= 1'b1;
116              pixel_empty <= 1'b0;
117              pixel_data <= {(`PIXEL_BITS){1'b0}};
118              rgb0 <= 24'b0;
119          end else begin
120              state <= next_state;
121              rhombus_data <= next_rhombus_data;
122              rhombus_noop <= next_rhombus_noop;
123              pixel_empty <= next_pixel_empty;
124              pixel_data <= next_pixel_data;
125              rgb0 <= rgb;
126          end
```

Page: 1   Page: 2

triangle_shader.v

```
127
128      // calculate next values
129  next_pixel ptp (next_point_data, rgb0, next_pixel_data);
     Point_to_pixel
130      always @ (state, rhombus_noop, rhombus_data, rhombus_a, rhombus_b,
     next_state
131  next_rhombus_data  rhombus_next, triangle_empty, triangle_noop) begin
132          next_state = state;
133                          = 1'b0;
134          next_rhombus_noop = rhombus_noop;
135  next_state  next_rhombus_data = rhombus_data;
136          triangle_next = 1'b0;
137  next_rhombus_noop
     triangle_next
138          case (state)
139              RHOMBUS_A:
140                  if (rhombus_next) begin
141                          = RHOMBUS_B;
142  triangle_next                        = rhombus_b;
143  next_pixel_empty   end
144              RHOMBUS_B:
145  triangle_next   if (rhombus_next) begin
146                          = NEXT;
147  next_state                       = 1'b1;
148  next_rhombus_data        = 1'b1;
149  next_rhombus_noop   end
150              NEXT:
151              begin
152                  if (triangle_empty) begin
153                          = 1'b1;
154                              = 1'b1;
155                  end else if (triangle_noop) begin
156                          = 1'b1;
157                  end else if (rhombus_next) begin
158                          = RHOMBUS_A;
159                              = rhombus_a;
160                              = 1'b0;
161                  end
162              end
163          endcase
164      end
165  endmodule
166
167  module point_to_pixel(point_data, rgb, pixel_data/*, pixel_noop*/);
168      input [`POINT_BITS-1:0] point_data;
169      input [`COLOR_BITS-1:0] rgb;
170      output [`PIXEL_BITS-1:0] pixel_data;
171      //output pixel_noop;
172
173      wire [`COORD_BITS-1:0] unsigned_x, unsigned_z;
174      wire [`COORD_BITS-1:0] point_x, point_y, point_z;
175      wire [`COORD_BITS:0] neg_y, unsigned_y;
176
177      assign {point_x, point_y, point_z} = point_data;
178
179      signed_negate #(`COORD_BITS) sn (point_y, neg_y);
180
181      signed_to_unsigned #(`COORD_BITS, (`SCREEN_WIDTH/2))
182          stux (point_x, unsigned_x);
183      signed_to_unsigned #(`COORD_BITS+1, (`SCREEN_HEIGHT/2))
184          stuy (neg_y, unsigned_y);
185      signed_to_unsigned #(`COORD_BITS, {1'b1, {`COORD_BITS-1{1'b0}}})
186          stuz (point_z, unsigned_z);
187
188
189      assign pixel_data = {unsigned_x[9:0], unsigned_y[9:0],
```

triangle_shader.v

```
190                          unsigned_z[`COORD_BITS-1:`COORD_BITS-12], rgb};
191      //assign pixel_noop = (unsigned_x[`COORD_BITS-1:10] != 0) ||
192      //                    (unsigned_y[`COORD_BITS-1:10] != 0);
193  endmodule
194
195  `include "defines.v"
196  module rhombus_shader(
197      reset, clock, rhombus_data, rhombus_noop, rhombus_next,
198      next_point_data, point_noop, point_next);
199      input reset;
200      input clock;
201
202      input [`RHOMBUS_BITS-1:0] rhombus_data;
203      input rhombus_noop;
204      output rhombus_next;
205
206      output [`POINT_BITS-1:0] next_point_data;
207      output point_noop;
208      input point_next;
209
210      // split up rhombus data
211      wire [`LINE_BITS-1:0] left_line, right_line;
212      wire [`COORD_BITS-1:0] min_x, min_y, max_x, max_y, rhombus_z;
213      assign {left_line, right_line,
214          min_x, min_y,
215          max_x, max_y, rhombus_z} = rhombus_data;
216
217      // register point data output
218      reg [`COORD_BITS-1:0] point_x, point_y, point_z;
219      reg point_noop;
220
221      // state
222      reg state;
223      localparam IDLE = 0;
224      localparam PAINTING = 1;
225
226      // next state variables
227      reg [`COORD_BITS-1:0] next_point_x, next_point_y, next_point_z;
228      assign next_point_data = {next_point_x, next_point_y, next_point_z};
229      reg next_point_noop, next_state;
230      reg rhombus_next;
231
232      wire [`COORD_BITS:0] inc_x, inc_y;
233      signed_inc #(`COORD_BITS) six(point_x, inc_x);
234      signed_inc #(`COORD_BITS) siy(point_y, inc_y);
235
236      always @ (posedge clock)
237          if (reset) begin
238              point_x = {(`POINT_BITS){1'b0}};
239              point_y = {(`POINT_BITS){1'b0}};
240              point_z = {(`POINT_BITS){1'b0}};
241              state = IDLE;
242              point_noop = 1'b1;
243          end else begin
244              point_x = next_point_x;
245              point_y = next_point_y;
246              point_z = next_point_z;
247              state = next_state;
248              point_noop = next_point_noop;
249          end
250
251      wire before_left_line;
252      wire after_right_line;
```

Page: 3

Page: 4

triangle_shader.v

```
253
254      always @ (state, rhombus_noop, point_next, point_y, max_y, point_x,
255          min_x, min_y, rhombus_z, max_x, max_y, point_z, point_noop,
256          before_left_line, after_right_line, inc_y, inc_x) begin
257          next_state = state;
258          next_point_x = point_x;
259          next_point_y = point_y;
260          next_point_z = point_z;
261          next_point_noop = point_noop;
262          rhombus_next = 1'b0;
263          case (state)
264              IDLE:
265                  if (rhombus_noop) begin
266                      rhombus_next        = 1'b1;
267                      next_point_noop     = 1'b1;
268                  end else begin
269                      next_state          = PAINTING;
270                      next_point_x        = min_x;
271                      next_point_y        = min_y;
272                      next_point_z        = rhombus_z;
273                      next_point_noop     = before_left_line || after_right_line;
274                  end
275              PAINTING:
276                  if (point_next) begin
277                      if (point_y == max_y) begin
278                          next_state          = IDLE;
279                          rhombus_next        = 1'b1;
280                          next_point_noop     = 1'b1;
281                      end else if (point_x == max_x) begin
282                          next_point_x        = min_x;
283                          next_point_y        = inc_y[`COORD_BITS-1:0];
284                          next_point_noop     = before_left_line || after_right_line;
285                      end else begin
286                          next_point_x        = inc_x[`COORD_BITS-1:0];
287                          next_point_noop     = before_left_line || after_right_line;
288                      end
289                  end
290          endcase
291      end
292
293      //point_left_of_line
294      wire before_right_line;
295      point_left_of_line plol1
296          (next_point_x, next_point_y, left_line, before_left_line);
297
298      point_left_of_line plol2
299          (next_point_x, next_point_y, right_line, before_right_line);
300
301      assign after_right_line = ~before_right_line;
302  endmodule
303
304  module point_left_of_line(p_x, p_y, line, out);
305      input [`COORD_BITS-1:0] p_x, p_y;
306      input [`LINE_BITS-1:0] line;
307      output out;
308
309      wire [`COORD_BITS-1:0] a_x, a_y, b_x, b_y;
310      assign {a_x, a_y, b_x, b_y} = line;
311
312
313      wire [`COORD_BITS:0] pxmax, aymby, pymay, axmbx;
314
315      signed_subtract #(`COORD_BITS) ss1 (p_x, a_x, pxmax);
```

triangle_shader.v

```
316      signed_subtract #(`COORD_BITS) ss2 (a_y, b_y, aymby);
317      signed_subtract #(`COORD_BITS) ss3 (p_y, a_y, pymay);
318      signed_subtract #(`COORD_BITS) ss4 (a_x, b_x, axmbx);
319
320      wire [2*(`COORD_BITS)+1:0] left, right;
321      signed_multiply #(`COORD_BITS+1,`COORD_BITS+1) sm1 (pxmax, aymby, left);
322      signed_multiply #(`COORD_BITS+1,`COORD_BITS+1) sm2 (pymay, axmbx, right);
323
324      wire left_lt_right, ay_lt_by;
325      signed_lt #(2*(`COORD_BITS)+2) slt1 (left, right, left_lt_right);
326      signed_lt #(`COORD_BITS) slt2 (a_y, b_y, ay_lt_by);
327
328      assign out = left_lt_right ^ ay_lt_by;
329  endmodule
330
```

triangle_pipeline.v

```verilog
1    `include "defines.v"
2    module triangle_pipeline(
3        reset, clock,
4        in_triangle, in_noop, in_empty, in_next,
5        out_triangle, out_noop, out_empty, out_next,
6        translation_vector, rotation_matrix, light_vector, zoom,
7        rotor_matrix, ball_vector, paddle_vector);
8        input reset;
9        input clock;
10
11       // incomming triangles
12       input [`TRIANGLE_BITS+`NORMAL_BITS-1:0] in_triangle;
13       input in_empty;
14       input in_noop;
15       output in_next;
16
17       // outgoing triangles
18       output [`TRIANGLE_BITS-1:0] out_triangle;
19       output out_empty;
20       output out_noop;
21       input out_next;
22
23       // control signals
24       input [3*`COORD_BITS-1:0] translation_vector, ball_vector,
25                             paddle_vector , light_vector;
26       input [9*`TRIG_OUT_BITS-1:0] rotation_matrix, rotor_matrix;
27       input [`COORD_BITS-1:0] zoom;
28
29       // next propogation wiring
30       assign in_next = out_next;
31
32       // internal wiring
33
34       wire [`NORMAL_BITS-1:0] next_normal0, next_normal1, next_normal2, next_normal3,
35                             next_normal4, next_normal5, next_normal6,
36                             normal0, normal1, normal2, normal3, normal4,
37                             normal5, normal6;
38       wire [`TRIANGLE_BITS-1:0]
39           next_data0, next_data1, next_data2, next_data3, next_data4, next_data5,
40           next_data6, next_data7, next_data8, next_data9, next_data10, next_data11,
41           next_data12, next_data13,
42           data0, data1, data2,  data3, data4, data5, data6, data7, data8, data9,
43           data10, data11, data12, data13;
44       wire next_noop0, next_noop1, next_noop2, next_noop3, next_noop4, next_noop5,
45           next_noop6, next_noop7, next_noop8, next_noop9, next_noop10, next_noop11,
46           next_noop12, next_noop13,
47           noop0, noop1, noop2, noop3, noop4, noop5, noop6, noop7, noop8, noop9,
48           noop10, noop11, noop12, noop13,
49           empty0, empty1, empty2, empty3, empty4, empty5, empty6, empty7, empty8,
50           empty9, empty10, empty11, empty12, empty13;
51
52       assign {data0, normal0} = in_triangle;
53       assign noop0 = in_noop;
54       assign empty0 = in_empty;
55
56       pipeline_register #(`TRIANGLE_BITS+`NORMAL_BITS) pr0
57           (reset, clock, out_next, {data0,normal0}, noop0, empty0,
58               {data1, normal1}, noop1, empty1);
59
60       wire [`TRIANGLE_BITS-1:0] optdata2;
61       wire [`NORMAL_BITS-1:0] optnormal2;
62       wire optnoop2, optempty2;
```

Page: 1

E:\rogi\lab4\triangle_pipeline.v
rotation_block b1

```verilog
64           (data1, normal1, next_data1, next_normal1, next_noop1, rotor_matrix);
65       pipeline_register #(`TRIANGLE_BITS+`NORMAL_BITS) pr1
66           (reset, clock, out_next, {data1,normal1}, noop1, empty1,
67               {data2, normal2}, noop2, empty2);
68       pipeline_register #(`TRIANGLE_BITS+`NORMAL_BITS) pr1_opt
69           (reset, clock, out_next, {next_data1,next_normal1}, next_noop1, empty1,
70               {optdata2, optnormal2}, optnoop2, optempty2);
71
72       opt_block #(`ROTOR_COLOR) b2
73           (optdata2, optnormal2, optnoop2, data2, normal2,
74               next_data2, next_normal2, next_noop2);
75       pipeline_register #(`TRIANGLE_BITS+`NORMAL_BITS) pr2
76           (reset, clock, out_next, {next_data2,next_normal2}, next_noop2||noop2, empty2,
77               {data3, normal3}, noop3, empty3);
78
79       opt_translation_block #(`BALL_COLOR) b3 (data3, next_data3, next_noop3, ball_vector);
80       pipeline_register #(`TRIANGLE_BITS+`NORMAL_BITS) pr3
81           (reset, clock, out_next, {next_data3, normal3}, next_noop3||noop3, empty3,
82               {data4, normal4}, noop4, empty4);
83
84       opt_translation_block #(`PADDLE_COLOR) b4
85           (data4, next_data4, next_noop4, paddle_vector);
86       pipeline_register #(`TRIANGLE_BITS+`NORMAL_BITS) pr4
87           (reset, clock, out_next, {next_data4, normal4}, next_noop4||noop4, empty4,
88               {data5, normal5}, noop5, empty5);
89   rotation_block b5
90
91           (data5, normal5, next_data5, next_normal5, next_noop5, rotation_matrix);
92       pipeline_register #(`TRIANGLE_BITS+`NORMAL_BITS) pr5
93           (reset, clock, out_next, {next_data5,next_normal5}, next_noop5||noop5, empty5,
94               {data6, normal6}, noop6, empty6);
95
96       wire [9:0] next_change, change;
97       light_block0 b6 (data6, normal6, next_data6, next_change, next_noop6, light_vector);
98       pipeline_register #(`TRIANGLE_BITS+10) pr6
99           (reset, clock, out_next, {next_data6, next_change}, next_noop6||noop6, empty6,
100              {data7, change}, noop7, empty7);
101
102      light_block1 b7 (data7, change, next_data7, next_noop7);
103      pipeline_register #(`TRIANGLE_BITS) pr7
104          (reset, clock, out_next, next_data7, next_noop7||noop7, empty7,
105              data8, noop8, empty8);
106
107      normal_block b8 (data8, next_data8, next_noop8);
108      pipeline_register #(`TRIANGLE_BITS) pr8
109          (reset, clock, out_next, next_data8, next_noop8||noop8, empty8,
110              data9, noop9, empty9);
111
112      reorder_block b9 (data9, next_data9, next_noop9);
113      pipeline_register #(`TRIANGLE_BITS) pr9
114          (reset, clock, out_next, next_data9, next_noop9||noop9, empty9,
115              data10, noop10, empty10);
116
117      zoom_block b10 (data10, next_data10, next_noop10, zoom);
118      pipeline_register #(`TRIANGLE_BITS) pr10
119          (reset, clock, out_next, next_data10, next_noop10||noop10, empty10,
120              data11, noop11, empty11);
121
122      translation_block b11 (data11, next_data11, next_noop11, translation_vector);
123      pipeline_register #(`TRIANGLE_BITS) pr11
124          (reset, clock, out_next, next_data11, next_noop11||noop11, empty11,
```

Page: 2

```
      triangle_pipeline.v
_____
126              data12, noop12, empty12);
127
128      wire [4*(`COORD_BITS)-1:0] next_extrema, extrema;
129      keep_visible_block0 b12 (data12, next_data12, next_extrema, next_noop12);
130      pipeline_register #(`TRIANGLE_BITS+(4*(`COORD_BITS))) pr12
131          (reset, clock, out_next, {next_data12, next_extrema},
132              next_noop12||noop12, empty12,
133              {data13, extrema}, noop13, empty13);
134
135      keep_visible_block1 b13 (data13, extrema, next_data13, next_noop13);
136      pipeline_register #(`TRIANGLE_BITS) pr13
137          (reset, clock, out_next, next_data13, next_noop13||noop13, empty13,
138              out_triangle, out_noop, out_empty);
139
140  endmodule
141
142  module keep_visible_block0(in_triangle, out_triangle, out_extrema, out_noop);
143      input [`TRIANGLE_BITS-1:0] in_triangle;
144
145      // outgoing triangles
146      output [`TRIANGLE_BITS-1:0] out_triangle;
147      output [4*(`COORD_BITS)-1:0] out_extrema;
148      output out_noop;
149
150      // split up inputs
151      wire [`POINT_BITS-1:0] in_a, in_b, in_c;
152      wire [`COORD_BITS-1:0] a_x, a_y, a_z, b_x, b_y, b_z, c_x, c_y, c_z;
153      wire [`COLOR_BITS-1:0] in_rgb;
154
155      assign {in_a, in_b, in_c, in_rgb} = in_triangle;
156      assign {a_x, a_y, a_z} = in_a;
157      assign {b_x, b_y, b_z} = in_b;
158      assign {c_x, c_y, c_z} = in_c;
159      assign out_triangle = in_triangle;
160
161      // compare the y coordinates
162      wire [`COORD_BITS-1:0] min_x, max_x, min_y, max_y;
163
164      signed_outer3 #(`COORD_BITS) sox (a_x, b_x, c_x, min_x, max_x);
165      signed_outer3 #(`COORD_BITS) soy (a_y, b_y, c_y, min_y, max_y);
166
167      assign out_extrema = {min_x, max_x, min_y, max_y};
168
169      assign out_noop = 0;
170  endmodule
171
172  module keep_visible_block1(in_triangle, in_extrema, out_triangle, out_noop);
173      input [`TRIANGLE_BITS-1:0] in_triangle;
174      input [4*(`COORD_BITS)-1:0] in_extrema;
175
176      // outgoing triangles
177      output [`TRIANGLE_BITS-1:0] out_triangle;
178      output out_noop;
179
180      // split up inputs
181      assign out_triangle = in_triangle;
182
183      // compare the y coordinates
184      wire [`COORD_BITS-1:0] min_x, max_x, min_y, max_y;
185      assign {min_x, max_x, min_y, max_y} = in_extrema;
186
187      wire slt0, slt1, slt2, slt3;
188      signed_lt #(`COORD_BITS) slt_0 (max_x, -(`SCREEN_WIDTH/2), slt0);
```

```
      triangle_pipeline.v
_____
189      signed_lt #(`COORD_BITS) slt_1 ((`SCREEN_WIDTH/2), min_x, slt1);
190      signed_lt #(`COORD_BITS) slt_2 (max_y, -(`SCREEN_HEIGHT/2), slt2);
191      signed_lt #(`COORD_BITS) slt_3 ((`SCREEN_HEIGHT/2), min_y, slt3);
192
193      // assing noop output
194      assign out_noop = 0;//slt0||slt1||slt2||slt3;
195  endmodule
196
197  module light_block0(
198      in_triangle, in_normal, out_triangle, out_change, out_noop, light_vector);
199      input [`TRIANGLE_BITS-1:0] in_triangle;
200      input [`NORMAL_BITS-1:0] in_normal;
201
202      // outgoing triangles
203      output [`TRIANGLE_BITS-1:0] out_triangle;
204      output [9:0] out_change;
205      output out_noop;
206
207      input [3*`COORD_BITS-1:0] light_vector;
208
209      // split up inputs and outputs
210      assign out_triangle = in_triangle;
211      assign out_noop = 0;
212
213      wire [(2*`COORD_BITS+2)-1:0] correlation;
214
215      dot_product #(`COORD_BITS, `COORD_BITS) dp1 (in_normal, light_vector, correlation);
216
217      assign out_change = correlation[2*`COORD_BITS-1:2*`COORD_BITS-10];
218  endmodule
219
220  module light_block1(in_triangle, in_change, out_triangle, out_noop);
221      input [`TRIANGLE_BITS-1:0] in_triangle;
222      input [9:0] in_change;
223
224      // outgoing triangles
225      output [`TRIANGLE_BITS-1:0] out_triangle;
226      output out_noop;
227
228      // split up inputs and outputs
229      wire [7:0] out_r, out_g, out_b, in_r, in_g, in_b;
230      wire [3*`POINT_BITS-1:0] in_points;
231      assign {in_points, in_r, in_g, in_b} = in_triangle;
232      assign out_triangle = {in_points, out_r, out_g, out_b};
233      assign out_noop = 0;
234
235      wire [10:0] temp_r, temp_g, temp_b;
236
237      signed_add #(10) sar ({2'b0, in_r}, in_change, temp_r);
238      signed_add #(10) sag ({2'b0, in_g}, in_change, temp_g);
239      signed_add #(10) sab ({2'b0, in_b}, in_change, temp_b);
240
241      wire [10:0] bound_r, bound_g, bound_b;
242
243      signed_bound #(11, 255, 0) sbr (temp_r, bound_r);
244      signed_bound #(11, 255, 0) sbg (temp_g, bound_g);
245      signed_bound #(11, 255, 0) sbb (temp_b, bound_b);
246
247      assign out_r = bound_r[7:0];
248      assign out_g = bound_g[7:0];
249      assign out_b = bound_b[7:0];
250  endmodule
251
```

Page: 3

Page: 4

```
     triangle_pipeline.v
     ─────────────────────────────────────────────
252  module normal_block(in_triangle, out_triangle, out_noop);
253      input [`TRIANGLE_BITS-1:0] in_triangle;
254
255      // outgoing triangles
256      output [`TRIANGLE_BITS-1:0] out_triangle;
257      //output [3*(2*`COORD_BITS+3)-1:0] out_normal;
258
259      output out_noop;
260
261      // split up inputs and outputs
262      wire [`POINT_BITS-1:0] in_a, in_b, in_c;
263      wire [`COLOR_BITS-1:0] in_rgb;
264
265      assign {in_a, in_b, in_c, in_rgb} = in_triangle;
266      assign out_triangle = in_triangle;
267
268
269      wire [3*(`COORD_BITS+1)-1:0] a_to_b, a_to_c;
270      vector_subtract #(`COORD_BITS) vs_1 (in_b, in_a, a_to_b);
271      vector_subtract #(`COORD_BITS) vs_2 (in_c, in_a, a_to_c);
272
273      wire [(2*`COORD_BITS+3)-1:0] n_x, n_y, n_z;
274      cross_product #(`COORD_BITS+1, `COORD_BITS+1) cp (a_to_b, a_to_c, {n_x, n_y, n_z});
275
276      assign out_noop = n_z[(2*`COORD_BITS+3)-1];
277      //assign out_normal = {n_x, n_y, n_z};
278  endmodule
279
280  module opt_block(
281      opt_triangle, opt_normal, opt_noop, in_triangle, in_normal,
282      out_triangle, out_normal, out_noop);
283      parameter COLOR = 24'b1;
284
285      input [`TRIANGLE_BITS-1:0] in_triangle, opt_triangle;
286      input [`NORMAL_BITS-1:0] in_normal, opt_normal;
287      input opt_noop;
288
289      // outgoing triangles
290      output [`TRIANGLE_BITS-1:0] out_triangle;
291      output [`NORMAL_BITS-1:0] out_normal;
292      output out_noop;
293
294      assign {out_triangle, out_normal, out_noop} =
295              ((in_triangle[23:0] == COLOR) ||
296               (in_triangle[23:0] == `MIT_RED) ||
297               (in_triangle[23:0] == `MIT_GRAY))?
298              {opt_triangle, opt_normal, opt_noop}:
299              {in_triangle, in_normal, 1'b0};
300  endmodule
301
302  module opt_translation_block(in_triangle, out_triangle, out_noop, translation_vector);
303      parameter COLOR = 24'b1;
304
305      input [`TRIANGLE_BITS-1:0] in_triangle;
306
307      // outgoing triangles
308      output [`TRIANGLE_BITS-1:0] out_triangle;
309      output out_noop;
310
311      input [3*`COORD_BITS-1:0] translation_vector;
312
313      wire [`TRIANGLE_BITS-1:0] out_triangle0;
314      wire out_noop0;
```

```
     triangle_pipeline.v
     ─────────────────────────────────────────────
315
316      translation_block tb (in_triangle, out_triangle0, out_noop0, translation_vector);
317
318      assign {out_triangle, out_noop} =
319          in_triangle[23:0] == COLOR ?
320              {out_triangle0, out_noop0}:
321              {in_triangle, 1'b0};
322  endmodule
323
324  module pipeline_register(
325      reset, clock, next, in_data, in_noop, in_empty, out_data, out_noop, out_empty);
326
327      parameter WIDTH = 1;
328
329      input reset, clock, next, in_noop, in_empty;
330      output out_noop, out_empty;
331
332      input [WIDTH-1:0] in_data;
333      output [WIDTH-1:0] out_data;
334
335      reg out_noop, out_empty;
336      reg [WIDTH-1:0] out_data;
337
338      always @ (posedge clock)
339          if (reset) begin
340              out_data <= {WIDTH{1'b0}};
341              out_noop <= 1'b1;
342              out_empty <= 1'b0;
343          end else if (next) begin
344              out_data <= in_data;
345              out_noop <= in_noop;
346              out_empty <= in_empty;
347          end
348  endmodule
349
350  module reorder_block(in_triangle, out_triangle, out_noop);
351      input [`TRIANGLE_BITS-1:0] in_triangle;
352
353      // outgoing triangles
354      output [`TRIANGLE_BITS-1:0] out_triangle;
355      output out_noop;
356
357      // split up inputs
358      wire [`POINT_BITS-1:0] in_a, in_b, in_c;
359      wire [`COORD_BITS-1:0] a_x, a_y, a_z, b_x, b_y, b_z, c_x, c_y, c_z;
360      wire [`COLOR_BITS-1:0] in_rgb;
361
362      assign {in_a, in_b, in_c, in_rgb} = in_triangle;
363      assign {a_x, a_y, a_z} = in_a;
364      assign {b_x, b_y, b_z} = in_b;
365      assign {c_x, c_y, c_z} = in_c;
366
367      // compare the y coordinates
368      wire a_lt_b, a_lt_c, b_lt_c;
369      signed_lt #(`COORD_BITS) slt0 (a_y, b_y, a_lt_b);
370      signed_lt #(`COORD_BITS) slt1 (a_y, c_y, a_lt_c);
371      signed_lt #(`COORD_BITS) slt2 (b_y, c_y, b_lt_c);
372
373      // assign triangle output
374      reg [`TRIANGLE_BITS-1:0] out_triangle;
375      always @(a_lt_b, a_lt_c, b_lt_c, in_a, in_b, in_c, in_rgb)
376          if (!a_lt_b && ! a_lt_c)
377                              = {in_a, in_b, in_c, in_rgb};
```

```
    out_triangle

    out_triangle
   triangle_pipeline.v
```
_____

```
378          else if (!b_lt_c)
379                      = {in_b, in_c, in_a, in_rgb};
380          else
381                      = {in_c, in_a, in_b, in_rgb};
382
383      // assing noop output
384      assign out_noop = 1'b0;
385   endmodule
386
387   module rotation_block(
388      in_triangle, in_normal,
389      out_triangle, out_normal, out_noop,
390      rotation_matrix);
391
392      input [`TRIANGLE_BITS-1:0] in_triangle;
393      input [`NORMAL_BITS-1:0] in_normal;
394
395      // outgoing triangles
396      output [`TRIANGLE_BITS-1:0] out_triangle;
397      output [`NORMAL_BITS-1:0] out_normal;
398      output out_noop;
399
400      input [9*`TRIG_OUT_BITS-1:0] rotation_matrix;
401
402      // split up inputs and outputs
403      wire [9*`COORD_BITS-1:0] in_m, out_m;
404      wire [9*(`COORD_BITS+`TRIG_OUT_BITS+2)-1:0] temp_m;
405      wire [3*(`COORD_BITS+`TRIG_OUT_BITS+2)-1:0] temp_normal;
406      wire [`COLOR_BITS-1:0] in_rgb, out_rgb;
407
408      assign {in_m, in_rgb} = in_triangle;
409      assign out_triangle = {out_m, out_rgb};
410      assign out_rgb = in_rgb;
411
412      matrix_product #(`COORD_BITS, `TRIG_OUT_BITS)
413          mp (in_m, rotation_matrix, temp_m);
414      matrix_trim #(`COORD_BITS+`TRIG_OUT_BITS+2, `COORD_BITS, `TRIG_OUT_BITS-2)
415          mt (temp_m, out_m, out_noop);
416
417      vector_matrix_product #(`COORD_BITS, `TRIG_OUT_BITS)
418          vmp (in_normal, rotation_matrix, temp_normal);
419      vector_trim #(`COORD_BITS+`TRIG_OUT_BITS+2, `COORD_BITS, `TRIG_OUT_BITS-2)
420          vt (temp_normal, out_normal);
421   endmodule
422
423   module translation_block(
424      in_triangle, out_triangle, out_noop, translation_vector);
425      input [`TRIANGLE_BITS-1:0] in_triangle;
426
427      // outgoing triangles
428      output [`TRIANGLE_BITS-1:0] out_triangle;
429      output out_noop;
430
431      input [3*`COORD_BITS-1:0] translation_vector;
432
433      // split up inputs and outputs
434      wire [9*`COORD_BITS-1:0] in_m, out_m;
435      wire [9*(`COORD_BITS+1)-1:0] temp_m;
436      wire [`COLOR_BITS-1:0] in_rgb, out_rgb;
437
438      assign {in_m, in_rgb} = in_triangle;
439      assign out_triangle = {out_m, out_rgb};
440      assign out_rgb = in_rgb;
```
_____

Page: 7

```
   triangle_pipeline.v
```
_____

```
441
442      matrix_subtract #(`COORD_BITS)
443          ms (in_m, {3{translation_vector}}, temp_m);
444      matrix_trim #(`COORD_BITS+1, `COORD_BITS)
445          mt (temp_m, out_m, out_noop);
446   endmodule
447
448   module zoom_block(in_triangle, out_triangle, out_noop, zoom);
449      input [`TRIANGLE_BITS-1:0] in_triangle;
450      input [`COORD_BITS-1:0] zoom;
451
452      localparam TEMP_MAX = 2*`COORD_BITS-2;
453      localparam TEMP_MIN = `COORD_BITS-2;
454
455      // outgoing triangles
456      output [`TRIANGLE_BITS-1:0] out_triangle;
457      output out_noop;
458
459      // split up inputs
460      wire [`POINT_BITS-1:0] in_a, in_b, in_c;
461      wire [`COORD_BITS-1:0] in_a_x, in_a_y, in_a_z,
462                            in_b_x, in_b_y, in_b_z,
463                            in_c_x, in_c_y, in_c_z;
464      wire [`COLOR_BITS-1:0] in_rgb;
465
466      assign {in_a, in_b, in_c, in_rgb} = in_triangle;
467      assign {in_a_x, in_a_y, in_a_z} = in_a;
468      assign {in_b_x, in_b_y, in_b_z} = in_b;
469      assign {in_c_x, in_c_y, in_c_z} = in_c;
470
471      wire [`POINT_BITS-1:0] out_a, out_b, out_c;
472      wire [`COORD_BITS-1:0] out_a_x, out_a_y, out_a_z,
473                            out_b_x, out_b_y, out_b_z,
474                            out_c_x, out_c_y, out_c_z;
475      wire [`COLOR_BITS-1:0] out_rgb;
476
477      assign out_a = {out_a_x, out_a_y, out_a_z};
478      assign out_b = {out_b_x, out_b_y, out_b_z};
479      assign out_c = {out_c_x, out_c_y, out_c_z};
480      assign out_triangle = {out_a, out_b, out_c, out_rgb};
481
482      assign out_rgb = in_rgb;
483      assign out_noop = 0;
484
485      wire [2*`COORD_BITS-1:0] temp_a_x, temp_a_y, temp_a_z,
486                              temp_b_x, temp_b_y, temp_b_z,
487                              temp_c_x, temp_c_y, temp_c_z;
488
489      signed_multiply #(`COORD_BITS, `COORD_BITS)
490          sm_a_x (in_a_x, zoom, temp_a_x);
491      signed_multiply #(`COORD_BITS, `COORD_BITS)
492          sm_a_y (in_a_y, zoom, temp_a_y);
493      signed_multiply #(`COORD_BITS, `COORD_BITS)
494          sm_a_z (in_a_z, zoom, temp_a_z);
495
496      signed_multiply #(`COORD_BITS, `COORD_BITS)
497          sm_b_x (in_b_x, zoom, temp_b_x);
498      signed_multiply #(`COORD_BITS, `COORD_BITS)
499          sm_b_y (in_b_y, zoom, temp_b_y);
500      signed_multiply #(`COORD_BITS, `COORD_BITS)
501          sm_b_z (in_b_z, zoom, temp_b_z);
502
503      signed_multiply #(`COORD_BITS, `COORD_BITS)
```
_____

Page: 8

```
      triangle_pipeline.v
      _____
504          sm_c_x (in_c_x, zoom, temp_c_x);
505      signed_multiply #(`COORD_BITS, `COORD_BITS)
506          sm_c_y (in_c_y, zoom, temp_c_y);
507      signed_multiply #(`COORD_BITS, `COORD_BITS)
508          sm_c_z (in_c_z, zoom, temp_c_z);
509
510      assign out_a_x = temp_a_x[TEMP_MAX:TEMP_MIN];
511      assign out_a_y = temp_a_y[TEMP_MAX:TEMP_MIN];
512      assign out_a_z = temp_a_z[TEMP_MAX:TEMP_MIN];
513
514      assign out_b_x = temp_b_x[TEMP_MAX:TEMP_MIN];
515      assign out_b_y = temp_b_y[TEMP_MAX:TEMP_MIN];
516      assign out_b_z = temp_b_z[TEMP_MAX:TEMP_MIN];
517
518      assign out_c_x = temp_c_x[TEMP_MAX:TEMP_MIN];
519      assign out_c_y = temp_c_y[TEMP_MAX:TEMP_MIN];
520      assign out_c_z = temp_c_z[TEMP_MAX:TEMP_MIN];
521  endmodule
522
```

```
      triangle_source.v
      _____
1    `timescale 1ns / 1ps
2    `include "defines.v"
3    module triangle_source(
4          reset, clock, model_select, triangle_data,
5          triangle_empty, triangle_noop, next, next_frame);
6       parameter ADDRESS_BITS = 12;
7
8       input reset;
9       input clock;
10      output [`TRIANGLE_BITS + `NORMAL_BITS - 1:0] triangle_data;
11      output triangle_empty;
12      output triangle_noop;
13      input next;
14      input next_frame;
15      input [3:0] model_select;
16
17      wire [ADDRESS_BITS-1:0] address0, address1;
18      wire [`TRIANGLE_BITS + `NORMAL_BITS - 1:0] triangle_data0, triangle_data1;
19      wire triangle_empty0, triangle_empty1,
20          triangle_noop0, triangle_noop1;
21
22      assign {triangle_data, triangle_empty, triangle_noop} = model_select[0] ?
23          {triangle_data0, triangle_empty0, triangle_noop0} :
24          {triangle_data1, triangle_empty1, triangle_noop1};
25
26      rom_data_source #(`TRIANGLE_BITS + `NORMAL_BITS, ADDRESS_BITS) rds0 (
27          .clock(clock),
28          .reset(reset),
29          .next_address(address0),
30          .data(triangle_data0),
31          .empty(triangle_empty0),
32          .noop(triangle_noop0),
33          .next(next),
34          .next_frame(next_frame)
35      );
36
37      triangle_data td0 (.addr(address0), .clk(clock), .dout(triangle_data0));
38
39      rom_data_source #(`TRIANGLE_BITS + `NORMAL_BITS, ADDRESS_BITS) rds1 (
40          .clock(clock),
41          .reset(reset),
42          .next_address(address1),
43          .data(triangle_data1),
44          .empty(triangle_empty1),
45          .noop(triangle_noop1),
46          .next(next),
47          .next_frame(next_frame)
48      );
49
50      triangle_data1 td1 (.addr(address1), .clk(clock), .dout(triangle_data1));
51  endmodule
52
53  module rom_data_source(
54          reset, clock, next_address, data,
55          empty, noop, next, next_frame);
56      parameter DATA_BITS = 60;
57      parameter ADDRESS_BITS = 10;
58
59      input reset;
60      input clock;
61      output [ADDRESS_BITS-1:0] next_address;
62          // save a cycle by outputting next/address
63      input [DATA_BITS-1:0] data;
```

Page: 9   Page: 1

triangle_source.v

```
64        output empty;
65        output noop;
66        input next;
67        input next_frame;
68
69        //output [ADDRESS_BITS-1:0] max_address;
70
71
72        // register output
73        reg empty;
74        reg noop;
75        reg [ADDRESS_BITS-1:0] address;
76        reg [ADDRESS_BITS-1:0] max_address;
77
78        // wiring
79        wire next_empty;
80        wire next_noop;
81        wire [ADDRESS_BITS-1:0] next_address;
82        wire [ADDRESS_BITS-1:0] next_max_address;
83
84
85        always @ (posedge clock)
86            if (reset) begin
87                address <= 0;
88                empty <= 1'b0;
89                noop <= 1'b1;
90                max_address <= -1;
91            end else if (next_frame) begin
92                address <= 1;
93                empty <= 0;
94                noop <= 1;
95                max_address <= max_address;
96            end else begin
97                address <= next_address;
98                empty <= next_empty;
99                noop <= next_noop;
100               max_address <= next_max_address;
101           end
102
103       assign next_address =
104           ((address <= max_address) && next && !reset) ? address + 1: address;
105       assign next_empty =
106           next_address > max_address;
107       assign next_noop =
108           next_address == 0 || next_empty;
109       assign next_max_address =
110           (address == 0) ? data[ADDRESS_BITS-1:0] : max_address;
111   endmodule
112
```

orienter.v

```
1     `timescale 1ns / 1ps
2     `include "defines.v"
3     module orienter(reset, clock,
4                     mouse0_clock, mouse0_data,
5                     mouse1_clock, mouse1_data, buttons,
6                     enable, switch_buffer,
7                     paddle_y, ball_x, ball_y,
8                     translation_vector,
9                     rotation_matrix,
10                    light_vector,
11                    zoom,
12                    rotor_matrix,
13                    ball_vector,
14                    paddle_vector);
15
16        input reset;
17        input clock;
18
19        input mouse0_clock;
20        input mouse0_data;
21        input mouse1_clock;
22        input mouse1_data;
23        input [`NUM_BUTTONS-1:0] buttons;
24
25        input enable;
26        input switch_buffer;
27
28        input [9:0] paddle_y, ball_x, ball_y;
29
30        localparam MAX = {2'b01, {(`COORD_BITS-2){1'b0}}};
31        localparam ZERO = {(`COORD_BITS){1'b0}};
32
33        output [3*`COORD_BITS-1:0] translation_vector, ball_vector,
34                                   paddle_vector, light_vector;
35        output [9*`TRIG_OUT_BITS-1:0] rotation_matrix, rotor_matrix;
36        output [`COORD_BITS-1:0] zoom;
37
38        // calculate ball/paddle positions
39        wire [10:0] neg_paddle_y, neg_ball_x, neg_ball_y;
40        signed_subtract #(11) snpy (8, {1'b0, paddle_y}, neg_paddle_y);
41        signed_subtract #(11) snbx ((28+8+16), {1'b0, ball_x}, neg_ball_x);
42        signed_subtract #(11) snby (8, {1'b0, ball_y}, neg_ball_y);
43
44        wire [3*`COORD_BITS-1:0] next_ball_vector, next_paddle_vector;
45        assign next_paddle_vector =
46            {ZERO, {neg_paddle_y[10:0], 1'b0}, ZERO};
47        assign next_ball_vector =
48            {{neg_ball_x[10:0], 1'b0}, {neg_ball_y[10:0], 1'b0}, ZERO};
49
50        assign light_vector = {ZERO, {2'b11, {(`COORD_BITS-2){1'b0}}}, ZERO};
51        // maintain zoom
52        wire [`COORD_BITS-1:0] temp_zoom;
53        reg [`COORD_BITS-1:0] zoom;
54        slider #(`COORD_BITS, 1, MAX, MAX/2, 1) sl_zoom
55            (reset, clock, enable,
56            buttons[6]&&buttons[8], buttons[7]&&buttons[8], temp_zoom);
57
58        // maintain transpose
59        wire [`COORD_BITS-1:0] t_x, t_y, t_z;
60        wire [3*`COORD_BITS-1:0] temp_translation_vector;
61        reg [3*`COORD_BITS-1:0] translation_vector, paddle_vector, ball_vector;
62        assign temp_translation_vector = {t_x, t_y, t_z};
63        assign t_z = 0;
```

Page: 2

Page: 1

orienter.v

```verilog
 64
 65        slider #(`COORD_BITS, -13'd1000, 13'd1000, 0, 12'b1) slx
 66            (reset, clock, enable, buttons[1]&&buttons[8],
 67            buttons[0]&&buttons[8], t_x);
 68        slider #(`COORD_BITS, -13'd1000, 13'd1000, 0, 12'b1) sly
 69            (reset, clock, enable, buttons[2]&&buttons[8],
 70            buttons[3]&&buttons[8], t_y);
 71
 72
 73        // maintain rotation
 74        wire [`TRIG_IN_BITS-1:0] theta_x, theta_y, theta_z, theta_rotor;
 75        slider #(`TRIG_IN_BITS, -128, 127, 0, 1, 1) sl_theta_x
 76            (reset, clock, enable, buttons[1]&&(!buttons[8]),
 77            buttons[0]&&(!buttons[8]), theta_x);
 78        slider #(`TRIG_IN_BITS, -128, 127, -128, 1, 1) sl_theta_y
 79            (reset, clock, enable, buttons[2]&&(!buttons[8]),
 80            buttons[3]&&(!buttons[8]), theta_y);
 81        slider #(`TRIG_IN_BITS, -128, 127, -128, 1, 1) sl_theta_z
 82            (reset, clock, enable, buttons[6]&&(!buttons[8]),
 83            buttons[7]&&(!buttons[8]), theta_z);
 84        slider #(`TRIG_IN_BITS, -128, 127, 0, 3, 1) sl_theta_rotor
 85            (reset, clock, enable, 1'b0, 1'b1, theta_rotor);
 86
 87        wire [9*`TRIG_OUT_BITS-1:0] temp_rotation_matrix, temp_rotor_matrix;
    thetas_to_matrix tm
 88        reg [9*`TRIG_OUT_BITS-1:0] rotation_matrix, rotor_matrix;
 89
 90            (reset, clock, enable, theta_x, theta_y, theta_z,
 91             theta_rotor, temp_rotation_matrix, temp_rotor_matrix);
 92
 93        always @ (posedge clock)
 94            if (reset) begin
 95                rotation_matrix <= {(9*`TRIG_OUT_BITS){1'b0}};
 96                rotor_matrix <= {(9*`TRIG_OUT_BITS){1'b0}};
 97                translation_vector <= {(3*`COORD_BITS){1'b0}};
 98                paddle_vector <= {(3*`COORD_BITS){1'b0}};
 99                ball_vector <= {(3*`COORD_BITS){1'b0}};
100                zoom <= MAX/2;
101            end else if (switch_buffer) begin
102                zoom <= temp_zoom;
103                rotation_matrix <= temp_rotation_matrix;
104                rotor_matrix <= temp_rotor_matrix;
105                translation_vector <= temp_translation_vector;
106                paddle_vector <= next_paddle_vector;
107                ball_vector <= next_ball_vector;
108            end
109    endmodule
110
111    module slider(reset, clock, enable, up, down, value);
112        parameter WIDTH = 12;
113        parameter MIN_VALUE = -100;
114        parameter MAX_VALUE = 100;
115        parameter START_VALUE = 0;
116        parameter DELTA = 1;
117        parameter WRAP = 0;
118
119        input reset;
120        input clock;
121        input enable;
122        input up;
123        input down;
124        output [WIDTH-1:0] value;
125
126        // registered state
```

orienter.v

```verilog
127        reg [WIDTH-1:0] value;
128
129        // next holders
130        reg [WIDTH-1:0] next_value;
131
132        wire [WIDTH:0] sum, diff;
133        wire max_lt_sum, diff_lt_min;
134
135
136        always @ (posedge clock)
    enable
137            if (reset)
138                value <= START_VALUE;
139            else
140                value <= next_value;
141
142        always @(up or down or value or sum or diff or
143                    or max_lt_sum or diff_lt_min)
144            begin
145                // by default, don't change state
146                next_value = value;
147                if (enable && (up ^ down)) begin
148                    if (up)
149                        if (!max_lt_sum)
150                            next_value = sum[WIDTH-1:0];
151                        else
152                            next_value = WRAP ? MIN_VALUE : MAX_VALUE;
153                    else
154                        if (!diff_lt_min)
155                            next_value = diff[WIDTH-1:0];
156                        else
157                            next_value = WRAP ? MAX_VALUE : MIN_VALUE;
158                end
159            end
160
161        signed_add #(WIDTH)      sa (value, DELTA, sum);
162        signed_subtract #(WIDTH) ss (value, DELTA, diff);
163
164        signed_lt #(WIDTH+1) slt0 (MAX_VALUE, sum, max_lt_sum);
165        signed_lt #(WIDTH+1) slt1 (diff, MIN_VALUE, diff_lt_min);
166    endmodule
167
```

game_fsm.v

```verilog
1    `timescale lns / 1ps
2    module fsm (reset, clock, enable, speed, paddle_speed,
3                up_sync, down_sync, paddle_y, ball_x, ball_y);
4        input reset;
5        input clock;
6        input enable;
7        input [3:0] speed;
8        input [3:0] paddle_speed;
9        input up_sync;
10       input down_sync;
11       output [9:0] paddle_y;
12       output [9:0] ball_x;
13       output [9:0] ball_y;
14
15       // registered state
16       reg [0:0] state;
17       reg [6:0] speed_x;
18       reg [6:0] speed_y;
19       reg dir_x;
20       reg dir_y;
21
22       // Parameters
23       `include "params.v"
24       parameter SPEED_GRADIANT = 3;
25       parameter MAX_DIST = (PADDLE_HEIGHT + BALL_HEIGHT) / 2;
26
27       //  States
28       parameter PLAYING = 0;
29       parameter GAME_OVER = 1;
30
31       // next state holders
32       reg [0:0] next_state;
33       reg [6:0] next_speed_x;
34       reg [6:0] next_speed_y;
35       reg next_dir_x;
36       reg next_dir_y;
37
38       // Minor FSMs (Used mainly for abstraction)
39       paddle_fsm paddle_fsm1
40           (clock, reset, enable, up_sync, down_sync, paddle_speed, paddle_y);
41       ball_fsm  ball_fsm1
42           (reset, clock, enable, next_speed_x, next_speed_y,
43           next_dir_x, next_dir_y, ball_x, ball_y);
44
45       // state changes synchronously
46       always @ (posedge clock)
47           if (reset) begin
48               state <= PLAYING;
49               speed_x <= {4'b0, speed[1:0]};
50               speed_y <= {4'b0, speed[3:2]};
51               dir_x <= POS;
52               dir_y <= POS;
53           end else begin
54               state <= next_state;
55               speed_x <= next_speed_x;
56               speed_y <= next_speed_y;
57               dir_x <= next_dir_x;
58               dir_y <= next_dir_y;
59           end
60
61       // local wires used for calculation
62       wire [9:0] ball_center_y = ball_y + (BALL_HEIGHT / 2);
63       wire [9:0] paddle_center_y = paddle_y + (PADDLE_HEIGHT / 2);
```

game_fsm.v

```verilog
64       wire [9:0] dist = ball_center_y < paddle_center_y ?
65                               paddle_center_y - ball_center_y :
66                               ball_center_y - paddle_center_y;
67       wire [3:0] y_speed_change =
68   speed_x(SPEED_GRADIANT * dist) / (PADDLE_HEIGHT/2)/*MAX_DIST*/;
69       wire [3:0] x_speed_change = SPEED_GRADIANT - y_speed_change;
70       wire y_speed_change_dir =
71           ball_center_y < paddle_center_y ? NEG : POS;
72
73       // next state calculated asynchronously
74       always @ (enable or state or ball_x or ball_y or
75                       or speed_y or dir_x or dir_y
76               or dist or x_speed_change or y_speed_change
77   DISPLAY_WIDTHor y_speed_change_dir) begin
78               7/ by default, keep old state
79               next_state = state;
80               next_speed_x = speed_x;
81               next_speed_y = speed_y;
82   DISPLAYnextdir_x = dir_x;
83               next_dir_y = dir_y;
84
85           case (state)
86               PLAYING:
87                   if (enable) begin
88                       if (dir_x == POS &&
89                           ball_x + speed_x >=
90                                           - BORDER_WIDTH - BALL_WIDTH)
91                           next_dir_x = NEG; // Right Wall
92
93                       if (dir_y == POS &&
94                           ball_y + speed_y >=
95                                           - BORDER_HEIGHT - BALL_HEIGHT)
96                           next_dir_y = NEG; // Buttom Wall
97
98                       if (dir_y == NEG && ball_y <= BORDER_HEIGHT + speed_y)
99                           next_dir_y = POS; // Top Wall
100
101                      // see if we hit the paddle
102                      // (ignoring speed_y, but compensating for speed_x)
103                      if (dir_x == NEG
104                          && ball_x <= PADDLE_X + PADDLE_WIDTH + speed_x
105                          && ball_x + BALL_WIDTH + speed_x >= PADDLE_X
106                          && dist < MAX_DIST) begin
107
108  next_speed_x          next_dir_x = POS;
109                          next_speed_x = speed_x + x_speed_change;
110  next_speed_y
111                          // since we're using unsigned arithmetic,
112                          // speed_y is tricky
113                          if (dir_y == y_speed_change_dir)
114                              next_speed_y = speed_y + y_speed_change;
115                          else if (speed_y < y_speed_change) begin
116                              next_speed_y = y_speed_change - speed_y;
117                              next_dir_y = ~next_dir_y;
118                          end else
119                              next_speed_y = speed_y - y_speed_change;
120
121                          // Make sure we don't get too close to overflow
122                          if (next_speed_x > MAX_SPEED)
123                                          = MAX_SPEED;
124                          if (next_speed_y > MAX_SPEED)
125                                          = MAX_SPEED;
126                      end
```

```
game_fsm.v
 next_state
127
128                    if (dir_x == NEG && ball_x <= speed_x) begin
129                        // Hit left screen
130                        next_state = GAME_OVER;
131                        next_speed_x = 0;
132                        next_speed_y = 0;
133                    end
134                end
135            GAME_OVER:
136                if (up_sync && down_sync) begin
137                        = PLAYING;
138                    next_speed_x = {4'b0, speed[1:0]};
139                    next_speed_y = {4'b0, speed[3:2]};
140                    next_dir_x = POS;
141                    next_dir_y = POS;
142                end
143        endcase
144    end
145 endmodule
146
147 module ball_fsm(reset, clock, enable, speed_x,
148                speed_y, dir_x, dir_y, ball_x, ball_y);
149    input reset;
150    input clock;
151    input enable;
152    input [6:0] speed_x;
153    input [6:0] speed_y;
154    input dir_x;
155    input dir_y;
156    output [9:0] ball_x;
157    output [9:0] ball_y;
158
159    // Parameters
160    `include "params.v"
161
162    // State
163    reg [9:0] ball_x;
164    reg [9:0] ball_y;
165
166    // Next State
167    wire [9:0] next_x;
168    wire [9:0] next_y;
169
170    always @ (posedge clock)
171        if (reset) begin
172            ball_x <= PADDLE_X + PADDLE_WIDTH;
173            ball_y <= DISPLAY_HEIGHT / 2;
174        end else begin
175            ball_x <= next_x;
176            ball_y <= next_y;
177        end
178
179    assign next_x = enable ?
180        (dir_x == POS ? ball_x + speed_x : ball_x - speed_x) :
181        ball_x;
182    assign next_y = enable ?
183        (dir_y == POS ? ball_y + speed_y : ball_y - speed_y) :
184        ball_y;
185 endmodule
186
187 module paddle_fsm(clock, reset, enable, up_sync,
188                down_sync, paddle_speed, paddle_y);
189    input clock;
```

Page: 3

```
game_fsm.v
190    input reset;
191    input enable;
192    input up_sync;
193    input down_sync;
194    input [3:0] paddle_speed;
195    output [9:0] paddle_y;
196
197    // parameters
198    `include "params.v"
199    parameter MIN_Y = BORDER_HEIGHT;
200    parameter MAX_Y = DISPLAY_HEIGHT - BORDER_HEIGHT - PADDLE_HEIGHT;
201    parameter START_Y = (MIN_Y + MAX_Y)/2;
202
203    // registered state
204    reg [9:0] paddle_y;
205
206    // next holders
207    reg [9:0] next_y;
208
209    always @ (posedge clock)
210        if (reset)
211            paddle_y <= START_Y;
212        else
213            paddle_y <= next_y;
214
215    always @(up_sync or down_sync or paddle_y or enable or paddle_speed)
216        begin
217            // by default, don't change state
218            next_y = paddle_y;
219            if (enable && (up_sync ^ down_sync)) begin
220                if (up_sync)
221                    if (paddle_y > MIN_Y + paddle_speed)
222                        next_y = paddle_y - paddle_speed;
223                    else
224                        next_y = MIN_Y;
225                else
226                    if (paddle_y + paddle_speed < MAX_Y)
227                        next_y = paddle_y + paddle_speed;
228                    else
229                        next_y = MAX_Y;
230            end
231        end
232 endmodule
233
```

Page: 4

display_field.v
```
1    `timescale 1ns / 1ps
2    module display_field(reset, pixel_count, line_count,
3                         paddle_y, ball_x, ball_y, in_rgb, out_rgb);
4        input reset; // ignored, since display_field has no state
5        input [9:0] pixel_count;
6        input [9:0] line_count;
7        input [9:0] paddle_y;
8        input [9:0] ball_x;
9        input [9:0] ball_y;
10       input [23:0] in_rgb;
11       output [23:0] out_rgb;
12
13       // Internal Wiring
14       wire [23:0] rgb0, rgb1, rgb2, rgb3, rgb4, rgb5, rgb6,
15                   rgb7, rgb8, rgb9, rgb10, rgb11, rgb12;
16
17       // Global Parameters
18       `include "params.v"
19
20       // Borders
21       parameter BACGROND    = 0;
22       parameter FOREGROND   = -1;
23       assign rgb0 = in_rgb;//BACGROND;
24
25       box border1(pixel_count, line_count, 10'b0, 10'b0, rgb0, rgb1);
26       defparam border1.RGB = FOREGROND  ;
27  DISPLAY defparam border1.WDTH  = DISPLAYWDTH  ;
28       defparam border1.HIGH   = BORDER_HIGH  ;
29
30       box border2(pixel_count, line_count,
31                   _WDTH   - BORDER_WDTH , 10'b0, rgb1, rgb2);
32  DISPLAY defparam border2.RGB = FOREGROND  ;
33       defparam border2.WDTH  = BORDER_WDTH ;
34       defparam border2.HIGH   = DISPLAYHIGH   ;
35
36       box border3(pixel_count, line_count, 10'b0,
37                   _HIGH    - BORDER_HIGH  , rgb2, rgb3);
38       defparam border3.RGB = FOREGROND  ;
39       defparam border3.WDTH  = DISPLAYWDTH  ;
40       defparam border3.HIGH   = BORDER_HIGH  ;
41
42       // Logo
43       parameter MIT_RED = {8'b0101_1111, 8'b0001_1111, 8'b0001_1111};
44       parameter MIT_GRAY = {8'b0100_1111, 8'b0100_1111, 8'b0011_1111};
45       parameter LOGO_WDTH  = 10'd39;
46       parameter LOGO_HIGH   = 10'd195;
47       parameter LOGO_X = 10'd133;
48       parameter LOGO_Y = 10'd134;
49       parameter LOGO_DX = LOGO_WDTH  + 10'd25;
50
51       box m1(pixel_count, line_count, LOGO_X, LOGO_Y, rgb3, rgb4);
52       defparam m1.RGB = MIT_RED;
53       defparam m1.WDTH  = LOGO_WDTH ;
54       defparam m1.HIGH   = LOGO_HIGH  ;
55
56       box m2(pixel_count, line_count, LOGO_X + LOGO_DX, LOGO_Y, rgb4, rgb5);
57       defparam m2.RGB = MIT_RED;
58       defparam m2.WDTH  = LOGO_WDTH ;
59       defparam m2.HIGH   = 137;
60
61       box m3(pixel_count, line_count,
62              (LOGO_X + 2'd2*LOGO_DX), LOGO_Y, rgb5, rgb6);
63       defparam m3.RGB = MIT_RED;
64       defparam m3.WDTH  = LOGO_WDTH ;
65       defparam m3.HIGH   = LOGO_HIGH  ;
66
67       box i1(pixel_count, line_count,
68              (LOGO_X + 2'd3*LOGO_DX), LOGO_Y, rgb6, rgb7);
69       defparam i1.RGB = MIT_RED;
70       defparam i1.WDTH  = LOGO_WDTH ;
71       defparam i1.HIGH   = LOGO_WDTH ;
72
73       box i2(pixel_count, line_count,
74              (LOGO_X + 2'd3*LOGO_DX),
75              LOGO_Y + 2'd2*LOGO_WDTH , rgb7, rgb8);
76       defparam i2.RGB = MIT_GRAY;
77       defparam i2.WDTH  = LOGO_WDTH ;
78       defparam i2.HIGH   = LOGO_HIGH   - 2*LOGO_WDTH ;
79
80       box t1(pixel_count, line_count,
81              (LOGO_X + 3'd4*LOGO_DX), LOGO_Y, rgb8, rgb9);
82       defparam t1.RGB = MIT_RED;
83       defparam t1.WDTH  = LOGO_HIGH   - 2*LOGO_WDTH ;
84       defparam t1.HIGH   = LOGO_WDTH ;
85
86       box t2(pixel_count, line_count,
87              (LOGO_X + 3'd4*LOGO_DX),
88              (LOGO_Y + 2'd2*LOGO_WDTH ), rgb9, rgb10);
89       defparam t2.RGB = MIT_RED;
90       defparam t2.WDTH  = LOGO_WDTH ;
91       defparam t2.HIGH   = LOGO_HIGH   - 2*LOGO_WDTH ;
92
93       // paddle
94       box paddle(pixel_count, line_count,
95                  PADDLE_X, paddle_y, rgb10, rgb11);
96       defparam paddle.RGB = FOREGROND  ;
97       defparam paddle.WDTH  = PADDLE_WDTH ;
98       defparam paddle.HIGH   = PADDLE_HIGH  ;
99
100      // ball
101      box ball(pixel_count, line_count, ball_x, ball_y, rgb11, rgb12);
102      defparam ball.RGB = FOREGROND  ;
103      defparam ball.WDTH  = BALL_WDTH ;
104      defparam ball.HIGH   = BALL_HIGH  ;
105
106      assign out_rgb = rgb12;
107  endmodule
108
109  module box(pixel_count, line_count, x, y, rgb_in, rgb_out);
110      input [9:0] pixel_count;
111      input [9:0] line_count;
112      input [9:0] x;
113      input [9:0] y;
114      input [23:0] rgb_in;
115      output [23:0] rgb_out;
116
117      parameter WDTH  = 1;
118      parameter HIGH   = 1;
119      parameter RGB = 24'b0;
120
121      assign rgb_out =
122          ((pixel_count >= x && pixel_count < x + WDTH )
123          && (line_count >= y && line_count < y + HIGH  ))
124          ? RGB : rgb_in;
125  endmodule
126
```

Page: 1

Page: 2

signed.v

```
1    module signed_add(a, b, c);
2        parameter WIDTH = 12;
3
4        input signed [WIDTH-1:0] a, b;
5        output signed [WIDTH:0] c;
6
7        assign c = a + b;
8    endmodule
9
10   module signed_add3(a, b, c, sum);
11       parameter WIDTH = 12;
12       localparam C_WIDTH = WIDTH + 2;
13
14       input signed [WIDTH-1:0] a, b, c;
15       output signed [C_WIDTH-1:0] sum;
16
17       assign sum = a + b + c;
18   endmodule
19
20   module signed_inc(a, c);
21       parameter WIDTH = 12;
22
23       input signed [WIDTH-1:0] a;
24       output signed [WIDTH:0] c;
25
26       assign c = a + 1;
27   endmodule
28
29   module signed_lt(a, b, c);
30       parameter WIDTH = 12;
31
32       input signed [WIDTH-1:0] a, b;
33       output c;
34
35       assign c = a < b;
36   endmodule
37
38   module signed_bound(a, b);
39       parameter WIDTH = 12;
40       parameter MAX = 100;
41       parameter MIN = -100;
42
43       input signed [WIDTH-1:0] a;
44       output signed [WIDTH-1:0]  b;
45       assign b = a < MIN ? MIN : (a > MAX ? MAX : a);
46   endmodule
47
48   module signed_multiply(a, b, c);
49       parameter A_WIDTH = 12;
50       parameter B_WIDTH = 12;
51       localparam C_WIDTH = A_WIDTH + B_WIDTH;
52
53       input signed [A_WIDTH-1:0] a;
54       input signed [B_WIDTH-1:0] b;
55       output signed [C_WIDTH-1:0] c;
56
57       assign c = a * b;
58   endmodule
59
60   module signed_negate(a, b);
61       parameter WIDTH = 12;
62
63       input signed [WIDTH-1:0] a;
```

signed.v

```
64       output signed [WIDTH:0] b;
65
66       assign b = -a;
67   endmodule
68
69   module signed_outer3(a, b, c, min, max);
70       parameter WIDTH = 12;
71
72       input signed [WIDTH-1:0] a, b, c;
73       output signed [WIDTH-1:0] min, max;
74
75       assign min = a < b ?
76               (a < c ? a : c):
77               (b < c ? b : c);
78
79       assign max = a > b ?
80               (a > c ? a : c):
81               (b > c ? b : c);
82   endmodule
83
84   module signed_subtract(a, b, c);
85       parameter WIDTH = 12;
86
87       input signed [WIDTH-1:0] a, b;
88       output signed [WIDTH:0] c;
89
90       assign c = a - b;
91   endmodule
92
93   module signed_to_unsigned(in_a, next_a);
94       parameter WIDTH = 10;
95       parameter ZERO = 320;
96       input [WIDTH-1:0] in_a;
97       output [WIDTH-1:0] next_a;
98
99       wire [WIDTH:0] temp;
100
101      signed_add #(WIDTH) sa1 (in_a, ZERO, temp);
102      assign next_a = temp[WIDTH-1:0];
103  endmodule
104
105  module signed_trim(in_a, next_a, next_noop);
106      parameter IN_WIDTH = 13;
107      parameter OUT_WIDTH = 12;
108      parameter SKIP_WIDTH = 0;
109
110      input [IN_WIDTH-1:0] in_a;
111      output [OUT_WIDTH-1:0] next_a;
112      output next_noop;
113
114      wire [OUT_WIDTH-1:0] next_a;
115      wire [IN_WIDTH-1:OUT_WIDTH-1] trimmed;
116      wire next_noop;
117
118      assign next_a = in_a[OUT_WIDTH-1+SKIP_WIDTH:0+SKIP_WIDTH];
119      assign trimmed = in_a[IN_WIDTH-1:OUT_WIDTH-1+SKIP_WIDTH];
120      assign next_noop =
121          !(trimmed == {(IN_WIDTH-OUT_WIDTH-SKIP_WIDTH){1'b0}}
122                      || {(IN_WIDTH-OUT_WIDTH-SKIP_WIDTH){1'b1}});
123  endmodule
124
```

Page: 1

Page: 2

vector.v

```verilog
1    module vector_subtract(a, b, c);
2        parameter WIDTH = 12;
3
4        input [3*WIDTH-1:0] a, b;
5        output [3*(WIDTH+1)-1:0] c;
6
7        wire [WIDTH-1:0] a_x, a_y, a_z;
8        wire [WIDTH-1:0] b_x, b_y, b_z;
9        wire [WIDTH:0] c_x, c_y, c_z;
10
11       assign {a_x, a_y, a_z} = a;
12       assign {b_x, b_y, b_z} = b;
13 signed_subtract #c_x, c_y, c_z};
14 signed_subtract #
15 signed_subtract #     (WIDTH) ss_x (a_x, b_x, c_x);
16                       (WIDTH) ss_y (a_y, b_y, c_y);
17                       (WIDTH) ss_z (a_z, b_z, c_z);
18   endmodule
19 b3_x
20 module vector_matrix_product(a, b, c);
21       parameter A_WIDTH = 12;
22       parameter B_WIDTH = 12;
23       localparam C_WIDTH = A_WIDTH + B_WIDTH + 2;
24
25       input [3*A_WIDTH-1:0] a;          vector
26       input [9*B_WIDTH-1:0] b;          matrix
27       output [3*C_WIDTH-1:0] c;
28
29       wire [3*B_WIDTH-1:0] b1, b2, b3;
30       wire [B_WIDTH-1:0] b1_x, b1_y, b1_z,
31                           , b2_y, b2_z,
32                           , b3_y, b3_z;
33       wire [C_WIDTH-1:0] c_x, c_y, c_z;
34
35       assign {b1, b2, b3} = b;
36       assign {b1_x, b1_y, b1_z} = b1;
37       assign {b2_x, b2_y, b2_z} = b2;
38       assign {b3_x, b3_y, b3_z} = b3;
39
40 dot_product #c = {c_x, c_y, c_z};
41 dot_product #
42 dot_product #     (A_WIDTH, B_WIDTH) dp1 (a, {b1_x, b2_x, b3_x}, c_x);
43                   (A_WIDTH, B_WIDTH) dp2 (a, {b1_y, b2_y, b3_y}, c_y);
44                   (A_WIDTH, B_WIDTH) dp3 (a, {b1_z, b2_z, b3_z}, c_z);
45   endmodule
46
47 module vector_trim(in_v, next_v, next_noop);
48       parameter IN_WIDTH = 12;
49       parameter OUT_WIDTH = 11;
50       parameter SKIP_WIDTH = 0;
51
52       input [3*IN_WIDTH-1:0] in_v;
53       output [3*OUT_WIDTH-1:0] next_v;
54       output next_noop;
55
56       wire [IN_WIDTH-1:0] in_x, in_y, in_z;
57       wire [OUT_WIDTH-1:0] next_x, next_y, next_z;
58       wire x_noop, y_noop, znoop ;
59
60       assign {in_x, in_y, in_z} = in_v;
61       assign next_v = {next_x, next_y, next_z};
62       assign next_noop = x_noop||y_noop||znoop ;
63
```

Page: 1

vector.v
```verilog
     signed_trim #
     signed_trim #
64   signed_trim #     (IN_WIDTH, OUT_WIDTH, SKIP_WIDTH) st_x (in_x, next_x, x_noop);
65                     (IN_WIDTH, OUT_WIDTH, SKIP_WIDTH) st_y (in_y, next_y, y_noop);
66                     (IN_WIDTH, OUT_WIDTH, SKIP_WIDTH) st_z (in_z, next_z, znoop );
67   endmodule
68
69 module dot_product(a, b, c);
70       parameter A_WIDTH = 12;
71       parameter B_WIDTH = 12;
72       localparam C_WIDTH = A_WIDTH + B_WIDTH + 2;
73
74       input [3*A_WIDTH-1:0] a;
75       input [3*B_WIDTH-1:0] b;
76       output [C_WIDTH-1:0] c;
77
78       wire [A_WIDTH-1:0] a_x, a_y, a_z;
79       wire [B_WIDTH-1:0] b_x, b_y, b_z;
80       wire [(A_WIDTH+B_WIDTH)-1:0] c_x, c_y, c_z;
81
82       assign {a_x, a_y, a_z} = a;
83       assign {b_x, b_y, b_z} = b;
84 signed_multiply #
85 signed_multiply #
86 signed_multiply #     (A_WIDTH, B_WIDTH) sm_x (a_x, b_x, c_x);
87                       (A_WIDTH, B_WIDTH) sm_y (a_y, b_y, c_y);
88 signed_add3 #         (A_WIDTH, B_WIDTH) sm_z (a_z, b_z, c_z);
89                       (A_WIDTH+B_WIDTH) sa(c_x, c_y, c_z, c);
90   endmodule
91
92 module cross_product(a, b, c);
93       parameter A_WIDTH = 12;
94       parameter B_WIDTH = 12;
95
96       input [3*A_WIDTH-1:0] a;
97       input [3*B_WIDTH-1:0] b;
98       output [3*(A_WIDTH+B_WIDTH+1)-1:0] c;
99
100      wire [A_WIDTH-1:0] a_x, a_y, a_z;
101      wire [B_WIDTH-1:0] b_x, b_y, b_z;
102      wire [A_WIDTH+B_WIDTH-1:0] c_x1, c_y1, c_z1 , c_x2, c_y2, c_z2 ;
103
104      assign {a_x, a_y, a_z} = a;
105      assign {b_x, b_y, b_z} = b;
106 signed_multiply #
107                       (A_WIDTH, B_WIDTH) sm_x1 (a_y, b_z, c_x1);
108 signed_multiply #     (A_WIDTH, B_WIDTH) sm_x2 (a_z, b_y, c_x2);
109 signed_multiply #
110                       (A_WIDTH, B_WIDTH) sm_y1 (a_z, b_x, c_y1);
111 signed_multiply #     (A_WIDTH, B_WIDTH) sm_y2 (a_x, b_z, c_y2);
112 signed_multiply #
113                       (A_WIDTH, B_WIDTH) sm_z1  (a_x, b_y, c_z1 );
114 vector_subtract #     (A_WIDTH, B_WIDTH) sm_z2  (a_y, b_x, c_z2 );
115
116                       (A_WIDTH + B_WIDTH) vs ({c_x1, c_y1, c_z1 }, {c_x2, c_y2, c_z2 }, c);
117  endmodule
118
```

Page: 2

matrix.v
_____

```
1    module matrix_product(a, b, c);
2        parameter A_WIDTH = 12;
3        parameter B_WIDTH = 12;
4        localparam C_WIDTH = A_WIDTH+B_WIDTH+2;
5
6        input [9*A_WIDTH-1:0] a; // matrix
7        input [9*B_WIDTH-1:0] b; // matrix
8        output [9*C_WIDTH-1:0] c; // matrix
9
10       wire [3*B_WIDTH-1:0] b1, b2, b3;
11       wire [B_WIDTH-1:0] b1_x, b1_y, b1_z,
12                          b2_x, b2_y, b2_z,
13                          b3_x, b3_y, b3_z;
14
15       assign {b1, b2, b3} = b;
16       assign {b1_x, b1_y, b1_z} = b1;
17       assign {b2_x, b2_y, b2_z} = b2;
18       assign {b3_x, b3_y, b3_z} = b3;
19
20       wire [3*C_WIDTH-1:0] c1, c2, c3;
21       wire [C_WIDTH-1:0] c1_x, c1_y, c1_z,
22                          c2_x, c2_y, c2_z,
23                          c3_x, c3_y, c3_z;
24
25       assign c = {c1, c2, c3};
26       assign c1 = {c1_x, c1_y, c1_z};
27       assign c2 = {c2_x, c2_y, c2_z};
28       assign c3 = {c3_x, c3_y, c3_z};
29
30       matrix_vector_product #(A_WIDTH, B_WIDTH) mvpx
31           (a, {b1_x, b2_x, b3_x}, {c1_x, c2_x, c3_x});
32       matrix_vector_product #(A_WIDTH, B_WIDTH) mvpy
33           (a, {b1_y, b2_y, b3_y}, {c1_y, c2_y, c3_y});
34       matrix_vector_product #(A_WIDTH, B_WIDTH) mvpz
35           (a, {b1_z, b2_z, b3_z}, {c1_z, c2_z, c3_z});
36   endmodule
37
38   module matrix_subtract(a, b, c);
39       parameter WIDTH = 12;
40
41       input [9*WIDTH-1:0] a, b;
42       output [9*(WIDTH+1)-1:0] c;
43
44       wire [3*WIDTH-1:0] a1, a2, a3, b1, b2, b3;
45       wire [3*(WIDTH+1)-1:0] c1, c2, c3;
46
47       assign {a1, a2, a3} = a;
48       assign {b1, b2, b3} = b;
49       assign c = {c1, c2, c3};
50
51       vector_subtract #(WIDTH) vs_1 (a1, b1, c1);
52       vector_subtract #(WIDTH) vs_2 (a2, b2, c2);
53       vector_subtract #(WIDTH) vs_3 (a3, b3, c3);
54   endmodule
55
56   module matrix_trim(in_m, next_m, next_noop);
57       parameter IN_WIDTH = 12;
58       parameter OUT_WIDTH = 11;
59       parameter SKIP_WIDTH = 0;
60
61       input [9*IN_WIDTH-1:0] in_m;
62       output [9*OUT_WIDTH-1:0] next_m;
63       output next_noop;
```

matrix.v
_____

```
64
65       wire [3*IN_WIDTH-1:0] in_a, in_b, in_c;
66       wire [3*OUT_WIDTH-1:0] next_a, next_b, next_c;
67       wire a_noop, b_noop, c_noop;
68
69       assign {in_a, in_b, in_c} = in_m;
70       assign next_m = {next_a, next_b, next_c};
71       assign next_noop = a_noop||b_noop||c_noop;
72
73       vector_trim #(IN_WIDTH, OUT_WIDTH, SKIP_WIDTH) st_a (in_a, next_a, a_noop);
74       vector_trim #(IN_WIDTH, OUT_WIDTH, SKIP_WIDTH) st_b (in_b, next_b, b_noop);
75       vector_trim #(IN_WIDTH, OUT_WIDTH, SKIP_WIDTH) st_c (in_c, next_c, c_noop);
76   endmodule
77
78   module matrix_vector_product(a, b, c);
79       parameter A_WIDTH = 12;
80       parameter B_WIDTH = 12;
81       localparam C_WIDTH = A_WIDTH + B_WIDTH + 2;
82
83       input [9*A_WIDTH-1:0] a; // matrix
84       input [3*B_WIDTH-1:0] b; // vector
85       output [3*C_WIDTH-1:0] c;
86
87       wire [3*A_WIDTH-1:0] a1, a2, a3;
88       wire [C_WIDTH-1:0] c_x, c_y, c_z;
89
90       assign {a1, a2, a3} = a;
91       assign c = {c_x, c_y, c_z};
92
93       dot_product #(A_WIDTH, B_WIDTH) dp1 (a1, b, c_x);
94       dot_product #(A_WIDTH, B_WIDTH) dp2 (a2, b, c_y);
95       dot_product #(A_WIDTH, B_WIDTH) dp3 (a3, b, c_z);
96   endmodule
97
```

Left column — `thetas_to_matrix.v`

```verilog
thetas_to_matrix.v

1    `timescale 1ns / 1ps
2    `include "defines.v"
3    module thetas_to_matrix(
4            reset, clock, enable, theta_x, theta_y, theta_z,
5            theta_rotor, matrix, rotor_matrix);
6        input reset, clock, enable;
7
8        input [`TRIG_IN_BITS-1:0] theta_x, theta_y, theta_z, theta_rotor;
9        output [9*`TRIG_OUT_BITS-1:0] matrix, rotor_matrix;
10
11       localparam IDLE = 0;
12       localparam X_ROTATION = 1;
13       localparam YROTATION  = 2;
14       localparam Z_ROTATION = 3;
15       localparam ROTOR = 4;
16       localparam FINAL  = 5;
17
18       reg [2:0] state, next_state;
19       reg [`TRIG_IN_BITS-1:0] theta, next_theta;
20       reg [1:0] select, next_select;
21       reg [9*`TRIG_OUT_BITS-1:0] matrix, next_matrix,
22                              rotor_matrix, next_rotor_matrix,
23                              temp0, next_temp0,
24                              temp1, next_temp1;
25
26       wire [9*`TRIG_OUT_BITS-1:0] ttm_matrix, mt_matrix;
27       wire [9*(2*`TRIG_OUT_BITS+2)-1:0] mp_matrix;
28
29       theta_to_matrix ttm (theta, select, ttm_matrix);
30
31       matrix_product #(`TRIG_OUT_BITS, `TRIG_OUT_BITS)
32           mp (temp0, temp1, mp_matrix);
33
34           #(`TRIG_OUT_BITS+`TRIG_OUT_BITS+2, `TRIG_OUT_BITS, `TRIG_OUT_BITS-2)
35           mt (mp_matrix, mt_matrix, out_noop);
36
37       always @ (posedge clock)
38           if (reset) begin
39               state <= IDLE;
40               theta <=  {(`TRIG_IN_BITS){1'b0}};
41               select <= 2'b0;
42               temp0 <= {(9*`TRIG_OUT_BITS){1'b0}};
43               temp1 <= {(9*`TRIG_OUT_BITS){1'b0}};
44               matrix <= {(9*`TRIG_OUT_BITS){1'b0}};
45               rotor_matrix <= {(9*`TRIG_OUT_BITS){1'b0}};
46           end else begin
47               state <= next_state;
48               theta <= next_theta;
49               select <= next_select;
50               temp0 <= next_temp0;
51               temp1 <= next_temp1;
52               matrix <= next_matrix;
53               rotor_matrix <= next_rotor_matrix;
54           end
55
56       wire timer;
57       divider #(5) div0 (reset, clock, timer);
58
59       always @ (state, theta, select, temp0, temp1, matrix, enable,
60               theta_x, theta_y, ttm_matrix, theta_z, mt_matrix,
61               rotor_matrix, theta_rotor)  begin
62           next_state = state;
63           next_theta = theta;
```

Right column — `thetas_to_matrix.v`

```verilog
thetas_to_matrix.v

64           next_select = select;
65           next_temp0 = temp0;
66           next_temp1 = temp1;
67           next_matrix = matrix;
68           next_rotor_matrix = rotor_matrix;
69           if (timer || enable) begin
70               case (state)
71                   IDLE:
72                       if (enable) begin
73                                       = X_ROTATION;
74                                       = theta_x;
75                                       = 2'b0;
76                       end
77                   X_ROTATION:
78                       begin
79                                       = YROTATION ;
80                                       = theta_y;
81                                       = 2'b1;
82                                       = ttm_matrix;
83                       end
84                   _ROTATION :
85                       begin
86                                       = ttm_matrix;
87                                       = Z_ROTATION;
88                                       = theta_z;
89                                       = 2'b10;
90                       end
91                   Z_ROTATION:
92                       begin
93                                       = mt_matrix;
94                                       = ttm_matrix;
95                                       = theta_rotor;
96                                       = 2'b10;
97                                       = FINAL ;
98                       end
99                   FINAL :
100                      begin
101                                      = mt_matrix;
102                                      = ttm_matrix;
103                                      = IDLE;
104                      end
105              endcase
106          end
107      end
108  endmodule
109
110  module theta_to_matrix(theta, select, matrix);
111
112      input [`TRIG_IN_BITS-1:0] theta;
113      input [1:0] select;
114      output [9*`TRIG_OUT_BITS-1:0] matrix;
115
116      localparam MAX = {2'b01, {(`TRIG_OUT_BITS-2){1'b0}}};
117
118      reg [`TRIG_OUT_BITS-1:0]
119          a_x, a_y, a_z, b_x, b_y, b_z, c_x, c_y, c_z;
120      assign matrix =
121          {a_x, a_y, a_z, b_x, b_y, b_z, c_x, c_y, c_z};
122
123      wire [`TRIG_OUT_BITS-1:0] cos, sin, nsin;
124      wire [`TRIG_OUT_BITS:0] neg_sin;
125
126      trig t (theta, sin, cos);
```

Page: 1   Page: 2

thetas_to_matrix.v

```
127     signed_negate #(`TRIG_OUT_BITS) sn (sin, neg_sin);
128
129     assign nsin = neg_sin[`TRIG_OUT_BITS-1:0];
130
131     always @(theta, select, cos, sin, nsin) begin
132         {a_x, a_y, a_z, b_x, b_y, b_z, c_x, c_y, c_z} =
133             {(9*`TRIG_OUT_BITS){1'b0}};
134         case (select)
135             2'd0:
136             begin
137                 a_x = MAX;
138                 b_y = cos;
139                 b_z = sin;
140                 c_y = nsin;
141                 c_z = cos;
142             end
143             2'd1:
144             begin
145                 a_x = cos;
146                 a_z = nsin;
147                 b_y = MAX;
148                 c_x = sin;
149                 c_z = cos;
150             end
151             2'd2:
152             begin
153                 a_x = cos;
154                 a_y = sin;
155                 b_x = nsin;
156                 b_y = cos;
157                 c_z = MAX;
158             end
159         endcase
160     end
161 endmodule
162
163 module divider(reset, clock, enable);
164     parameter COUNT = 10'd10;
165     input clock;
166     input reset;
167     output enable;
168
169     reg [9:0]counter;
170     reg enable;
171
172     always @(posedge clock)
173         begin
174             enable <= 1'b0;
175             if (reset)
176                 counter <= 10'b0;
177             else if (counter == COUNT)
178                 begin
179                     counter <= 10'b0;
180                     enable <= 1'b1;
181                 end
182             else
183                 counter <= counter + 1;
184         end
185 endmodule
186
```

Page: 3

labkit.v

```
1  ////////////////////////////////////////////////////////////////////////////////
2  //
3  // 6.111 FPGA Labkit -- Template Toplevel Module for Lab 4 (Spring 2006)
4  //
5  //
6  // Created: March 13, 2006
7  // Author: Nathan Ickes
8  //
9  ////////////////////////////////////////////////////////////////////////////////
10
11 module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
12         ac97_bit_clock,
13
14         vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
15         vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
16         vga_out_vsync,
17
18         tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
19         tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
20         tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
21
22         tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
23         tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
24         tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
25         tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
26
27         ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
28         ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
29
30         ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
31         ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
32
33         clock_feedback_out, clock_feedback_in,
34
35         flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
36         flash_reset_b, flash_sts, flash_byte_b,
37
38         rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
39
40         mouse_clock, mouse_data, keyboard_clock, keyboard_data,
41
42         clock_27mhz, clock1, clock2,
43
44         disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
45         disp_reset_b, disp_data_in,
46
47         button0, button1, button2, button3, button_enter, button_right,
48         button_left, button_down, button_up,
49
50         switch,
51
52         led,
53
54         user1, user2, user3, user4,
55
56         daughtercard,
57
58         systemace_data, systemace_address, systemace_ce_b,
59         systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
60
61         analyzer1_data, analyzer1_clock,
62         analyzer2_data, analyzer2_clock,
63         analyzer3_data, analyzer3_clock,
```

Page: 1

```
      labkit.v                                                                    labkit.v

64              analyzer4_data, analyzer4_clock);           127
65                                                          128      output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
66      output beep, audio_reset_b, ac97_synch, ac97_sdata_out;  129          analyzer4_data;
67      input  ac97_bit_clock, ac97_sdata_in;               130      output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;
68                                                          131
69      output [7:0] vga_out_red, vga_out_green, vga_out_blue;  132      ////////////////////////////////////////////////////////////////////////////
70      output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,  133      //
71        vga_out_hsync, vga_out_vsync;                     134      // I/O Assignments
72                                                          135      //
73      output [9:0] tv_out_ycrcb;                          136      ////////////////////////////////////////////////////////////////////////////
74      output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,  137
75        tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,  138      // Audio Input and Output
76        tv_out_subcar_reset;                              139      assign beep= 1'b0;
77                                                          140      assign audio_reset_b = 1'b0;
78      input  [19:0] tv_in_ycrcb;                          141      assign ac97_synch = 1'b0;
79      input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,  142      assign ac97_sdata_out = 1'b0;
80        tv_in_hff, tv_in_aff;                             143
81      output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,  144      // Video Output
82        tv_in_reset_b, tv_in_clock;                       145      assign tv_out_ycrcb = 10'h0;
83      inout  tv_in_i2c_data;                              146      assign tv_out_reset_b = 1'b0;
84                                                          147      assign tv_out_clock = 1'b0;
85      inout  [35:0] ram0_data;                            148      assign tv_out_i2c_clock = 1'b0;
86      output [18:0] ram0_address;                         149      assign tv_out_i2c_data = 1'b0;
87      output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;  150      assign tv_out_pal_ntsc = 1'b0;
88      output [3:0] ram0_bwe_b;                            151      assign tv_out_hsync_b = 1'b1;
89                                                          152      assign tv_out_vsync_b = 1'b1;
90      inout  [35:0] ram1_data;                            153      assign tv_out_blank_b = 1'b1;
91      output [18:0] ram1_address;                         154      assign tv_out_subcar_reset = 1'b0;
92      output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;  155
93      output [3:0] ram1_bwe_b;                            156      // Video Input
94                                                          157      assign tv_in_i2c_clock = 1'b0;
95      input  clock_feedback_in;                           158      assign tv_in_fifo_read = 1'b0;
96      output clock_feedback_out;                          159      assign tv_in_fifo_clock = 1'b0;
97                                                          160      assign tv_in_iso = 1'b0;
98      inout  [15:0] flash_data;                           161      assign tv_in_reset_b = 1'b0;
99      output [23:0] flash_address;                        162      assign tv_in_clock = 1'b0;
100     output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;  163      assign tv_in_i2c_data = 1'bZ;
101     input  flash_sts;                                   164
102                                                         165      // Flash ROM
103     output rs232_txd, rs232_rts;                        166      assign flash_data = 16'hZ;
104     input  rs232_rxd, rs232_cts;                        167      assign flash_address = 24'h0;
105                                                         168      assign flash_ce_b = 1'b1;
106     input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;  169      assign flash_oe_b = 1'b1;
107                                                         170      assign flash_we_b = 1'b1;
108     input  clock_27mhz, clock1, clock2;                 171      assign flash_reset_b = 1'b0;
109                                                         172      assign flash_byte_b = 1'b1;
110     output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;  173
111     input  disp_data_in;                                174      // RS-232 Interface
112     output  disp_data_out;                              175      assign rs232_txd = 1'b1;
113                                                         176      assign rs232_rts = 1'b1;
114     input  button0, button1, button2, button3, button_enter, button_right,  177
115       button_left, button_down, button_up;              178      // LED Displays
116     input  [7:0] switch;                                179      assign disp_blank = 1'b1;
117     output [7:0] led;                                   180      assign disp_clock = 1'b0;
118                                                         181      assign disp_rs = 1'b0;
119     inout  [31:0] user1, user2, user3, user4;           182      assign disp_ce_b = 1'b1;
120                                                         183      assign disp_reset_b = 1'b0;
121     inout  [43:0] daughtercard;                         184      assign disp_data_out = 1'b0;
122                                                         185
123     inout  [15:0] systemace_data;                       186      // Buttons, Switches, and Individual LEDs
124     output [6:0]  systemace_address;                    187      assign led = 8'hFF;
125     output systemace_ce_b, systemace_we_b, systemace_oe_b;  188
126     input  systemace_irq, systemace_mpbrdy;             189      // User I/Os
```

Page: 2   Page: 3

labkit.v

```
190      assign user1 = 32'hZ;
191      assign user2 = 32'hZ;
192      assign user3 = 32'hZ;
193      assign user4 = 32'hZ;
194
195      // Daughtercard Connectors
196      assign daughtercard = 44'hZ;
197
198      // SystemACE Microprocessor Port
199      assign systemace_data = 16'hZ;
200      assign systemace_address = 7'h0;
201      assign systemace_ce_b = 1'b1;
202      assign systemace_we_b = 1'b1;
203      assign systemace_oe_b = 1'b1;
204
205      ////////////////////////////////////////////////////////////////////////
206      //
207      // Lab 4 Components
208      //
209      ////////////////////////////////////////////////////////////////////////
210
211      wire pclk, pixel_clock, locked;
212
213      assign pixel_clock = clock_27mhz;
214      assign ram0_clk = pixel_clock;
215      assign ram1_clk = pixel_clock;
216      assign clock_feedback_out = 1'b0;
217
218      DCM pixel_clock_dcm1 (.CLKIN(pixel_clock), .CLKFX(vga_out_pixel_clock));
219      // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm1 is 2
220      // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm1 is 2
221      // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm1  is "NONE"
222      // synthesis attribute CLKOUT_PHASE_SHIFT of pixel_clock_dcm1 is "NONE"
223      // synthesis attribute PHASE_SHIFT of pixel_clock_dcm1 is 0
224
225      // The composite sync signal is used to encode sync data in the green
226      // channel analog voltage for older monitors.  It does not need to be
227      // implemented for the monitors in the 6.111 lab, and can be left at 1'b1.
228      assign vga_out_sync_b = 1'b1;
229
230       wire reset_sync, up_sync, down_sync;
231       wire [8:0] buttons;
232
233       debounce db1 (1'b0, pixel_clock, ~button0, reset_sync);
234       debounce db2 (1'b0, pixel_clock, ~button1, up_sync);
235       debounce db3 (1'b0, pixel_clock, ~button2, down_sync);
236       debounce db4 (1'b0, pixel_clock, ~button_right, buttons[0]);
237       debounce db5 (1'b0, pixel_clock, ~button_left, buttons[1]);
238       debounce db6 (1'b0, pixel_clock, ~button_down, buttons[2]);
239       debounce db7 (1'b0, pixel_clock, ~button_up, buttons[3]);
240       debounce db8 (1'b0, pixel_clock, ~button1, buttons[4]);
241       debounce db9 (1'b0, pixel_clock, ~button2, buttons[5]);
242       debounce db10 (1'b0, pixel_clock, ~button3, buttons[6]);
243       debounce db11 (1'b0, pixel_clock, ~button_enter, buttons[7]);
244
245       assign buttons[8] = switch[7];
246
247       internal internal1(
248           .reset(reset_sync), .clock(pixel_clock),
249
250           .model_select({2'b0, switch[0], switch[6]}),
251
252           .paddle_up(up_sync),
```

labkit.v

```
253          .paddle_down(down_sync),
254          .paddle_speed({switch[5:3], 1'b1}),
255          .ball_initial_speed({switch[2:0], 1'b1}), // game inputs
256
257          .ram0_data(ram0_data),
258          .ram0_address(ram0_address),
259          .ram0_we_b(ram0_we_b), // ram 0 buss
260          .ram1_data(ram1_data),
261          .ram1_address(ram1_address),
262          .ram1_we_b(ram1_we_b), // ram 1 buss
263
264          .mouse0_clock(mouse_clock),
265          .mouse0_data(mouse_data), // ps/2 0 buss
266          .mouse1_clock(keyboard_clock),
267          .mouse1_data(keyboard_data), // ps/2 1 buss
268
269          .buttons(buttons),
270
271          .vga_rgb({vga_out_red, vga_out_green, vga_out_blue}),
272          .vga_blank_b(vga_out_blank_b),
273          .vga_hsync(vga_out_hsync),
274          .vga_vsync(vga_out_vsync) // vga outputs
275          );
276
277      // SRAMs
278      assign ram0_adv_ld = 1'b0;
279      assign ram0_cen_b = 1'b0;
280      assign ram0_ce_b = 1'b0;
281      assign ram0_oe_b = 1'b0;
282      assign ram0_bwe_b = 4'h0;
283      assign ram1_adv_ld = 1'b0;
284      assign ram1_cen_b = 1'b0;
285      assign ram1_ce_b = 1'b0;
286      assign ram1_oe_b = 1'b0;
287      assign ram1_bwe_b = 4'h0;
288
289      // Logic Analyzer
290      assign analyzer1_data = 16'h0;
291      assign analyzer1_clock = 1'b1;
292      assign analyzer2_data =  {ram0_we_b, ram0_address[14:0]};
293      assign analyzer2_clock = ram0_clk;
294      assign analyzer3_data = 16'h0;
295      assign analyzer3_clock = 1'b1;
296      assign analyzer4_data = {ram0_data[15:0]};
297      assign analyzer4_clock = ram0_clk;
298  endmodule
299
300  // Switch Debounce Module
301  // use your system clock for the clock input
302  // to produce a synchronous, debounced output
303  module debounce (reset, clock, noisy, clean);
304      parameter DELAY = 270000;   // .01 sec with a 27Mhz clock
305      input reset, clock, noisy;
306      output clean;
307
308      reg [18:0] count;
309      reg new, clean;
310
311      always @(posedge clock)
312        if (reset)
313          begin
314          count <= 0;
315          new <= noisy;
```

labkit.v
```
316        clean <= noisy;
317         end
318      else if (noisy != new)
319        begin
320      new <= noisy;
321      count <= 0;
322         end
323      else if (count == DELAY)
324        clean <= new;
325      else
326        count <= count+1;
327
328  endmodule
329
```

internal_tf.v
```
1    `timescale 1ns / 1ps
2    module internal_tf_v;
3        // Inputs
4        reg reset;
5        reg clock;
6        reg [3:0] model_select;
7        reg paddle_up;
8        reg paddle_down;
9        reg [3:0] paddle_speed;
10       reg [3:0] ball_initial_speed;
11       reg mouse0_clock;
12       reg mouse0_data;
13       reg mouse1_clock;
14       reg mouse1_data;
15       reg [7:0] buttons;
16       reg [35:0] write_data;
17       reg tristate;
18
19       // Outputs
20       wire [18:0] ram0_address;
21       wire ram0_we_b;
22       wire [18:0] ram1_address;
23       wire ram1_we_b;
24       wire [23:0] vga_rgb;
25       wire vga_blank_b;
26       wire vga_hsync;
27       wire vga_vsync;
28
29       // Bidirs
30       wire [35:0] ram0_data;
31       wire [35:0] ram1_data;
32
33       assign ram0_data = (tristate) ? {36{1'bZ}} : write_data;
34
35       // Instantiate the Unit Under Test (UUT)
36       internal uut (
37           .reset(reset),
38           .clock(clock),
39           .model_select(model_select),
40           .paddle_up(paddle_up),
41           .paddle_down(paddle_down),
42           .paddle_speed(paddle_speed),
43           .ball_initial_speed(ball_initial_speed),
44           .ram0_data(ram0_data),
45           .ram0_address(ram0_address),
46           .ram0_we_b(ram0_we_b),
47           .ram1_data(ram1_data),
48           .ram1_address(ram1_address),
49           .ram1_we_b(ram1_we_b),
50           .mouse0_clock(mouse0_clock),
51           .mouse0_data(mouse0_data),
52           .mouse1_clock(mouse1_clock),
53           .mouse1_data(mouse1_data),
54           .vga_rgb(vga_rgb),
55           .vga_blank_b(vga_blank_b),
56           .vga_hsync(vga_hsync),
57           .vga_vsync(vga_vsync),
58           .buttons(buttons)
59       );
60       always #5 clock <= ~clock;
61
62       initial begin
63           // Initialize Inputs
```

internal_tf.v

```
64          reset = 0;
65          clock = 0;
66          model_select = 0;
67          paddle_up = 0;
68          paddle_down = 0;
69          paddle_speed = 0;
70          ball_initial_speed = 0;
71          mouse0_clock = 0;
72          mouse0_data = 0;
73          mouse1_clock = 0;
74          mouse1_data = 0;
75          tristate = 1;
76          write_data = 0;
77          buttons = 0;
78          // Wait 100 ns for global reset to finish
79          #100;
80          reset = 1;
81          @(posedge clock);
82          @(posedge clock);
83          @(posedge clock);
84          reset = 0;
85          @(posedge clock);
86          @(posedge clock);
87          @(posedge clock);
88          tristate = 0;
89          @(posedge clock);
90          tristate = 1;
91          @(posedge clock);
92          tristate = 0;
93          @(posedge clock);
94          tristate = 1;
95          @(posedge clock);
96          tristate = 0;
97          @(posedge clock);
98          tristate = 1;
99          @(posedge clock);
100         tristate = 0;
101         @(posedge clock);
102         tristate = 1;
103         @(posedge clock);
104         tristate = 0;
105         @(posedge clock);
106         tristate = 1;
107     end
108 endmodule
109
110
```

defines.v

```
1   `define COORD_BITS (12)
2   `define COLOR_BITS (24)
3   `define POINT_BITS (3 * `COORD_BITS)
4   `define TRIANGLE_BITS ((3 * `POINT_BITS) + `COLOR_BITS)
5   `define NORMAL_BITS (3 * `COORD_BITS)
6
7   // For Trig Functions
8   `define TRIG_IN_BITS (8)
9   `define TRIG_OUT_BITS (13)
10
11  // SVGA Output
12  `define PIXEL_BITS (10*2+12+24)
13  `define SCREEN_WIDTH (640)
14  `define SCREEN_HEIGHT (480)
15
16  `define P2D_BITS (2 * `COORD_BITS)
17  `define LINE_BITS (2 * `P2D_BITS)
18  `define RHOMBUS_BITS (2 * `LINE_BITS + 5*`COORD_BITS)
19
20  `define NUM_BUTTONS (9)
21
22  `define ROTOR_COLOR (24'h3F3FBE)
23  `define BALL_COLOR (24'hBEA33F)
24  `define PADDLE_COLOR (24'hBE3FBE)
25
26
27  `define MIT_RED  (24'h5B1F1F) //{8'b0101_1111, 8'b0001_1111, 8'b0001_1111})
28  `define MIT_GRAY  ({8'b0100_1111, 8'b0100_1111, 8'b0011_1111})
29
```

Page: 2

Page: 1

```
params.v
_____

1    // params.v
2    // Global Constants (Used in more than one module)
3    parameter DISPLAYWIDTH   = 10d640 ;
4    parameter DISPLAYHEIGHT     = 10d480 ;
5    parameter PADDLE_WIDTH   = 10d16 ;
6    parameter PADDLE_HEIGHT   = 10d128 ;
7    parameter PADDLE_X   = 10d28 ;
8    parameter BALLWIDTH    = 10d64 ;
9    parameter BALLHEIGHT      = 10d64 ;
10   parameter BORDERWIDTH    = 10d8 ;
11   parameter BORDERHEIGHT     = 10d8 ;
12   parameter MAXSPEED    = 5b11111 ;
13
14   parameter POS = 0;
15   parameter NEG = 1;
16
17
```