Lab 6

Friday, August 19, 2022 12:49 PM

Date: August 19, 2022 DE1_SoC.sv Project: DE1_SoC

```
1   module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, SW, LEDR, GPIO_1, CLOCK_50);
2       output logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
3       output logic [9:0]  LEDR;
4       input  logic [3:0]  KEY;
5       input  logic [9:0]  SW;
6       output logic [35:0] GPIO_1; // Used for LED board
7       input logic CLOCK_50;
8
9       // Turn off HEX displays
10      assign HEX3 = '1;
11      assign HEX4 = '1;
12      assign HEX5 = '1;
13
14      // Reset
15      logic U;
16      int position;
17      int shift;
18      int shift2;
19      int counter;
20      logic rst;  // reset - toggle this on startup
21      assign rst = SW[9];
22      wire addPoint;
23      logic gameover;
24      logic lp0, lp1, lp2;
25      //assign gameover = SW[0];
26
27
28      logic [31:0] div_clk;
29      clock_divider cdiv (.clk(CLOCK_50),
30                          .rst,
31                          .divided_clocks(div_clk));
32
33      // Clock selection; allows for easy switching between simulation and board clock
34      logic clkSelect;
35
36      // Uncomment ONE of the following two lines depending on intention
37      assign clkSelect = CLOCK_50;  // for simulation
38      //assign clkSelect = div_clk[14]; // 1526 Hz clock for board
39
40
41      logic [15:0][15:0]RedPixels; // 16 x 16 array representing red LEDs (row x col)
42      logic [15:0][15:0]GrnPixels; // 16 x 16 array representing green LEDs (row x col)
43
44
45      LEDDriver Driver (.clk(clkSelect), .rst, .EnableCount(1'b1), .RedPixels, .
    GrnPixels, .GPIO_1);
46
47
48      Press UP(.clk(clkSelect), .reset(rst), .inputButton(KEY[0]), .out(U));
49      crash col(.clk(clkSelect), .gA(GrnPixels), .rA(RedPixels), .gameover, .reset(rst
    ), .addPoint);
50      bird flap(.clk(clkSelect), .rst, .button(U), .gameover, .position);
51      pipeShift move(.clk(clkSelect), .rst, .gameover, .shift, .counter);
52      single_hex d0(.clk(clkSelect), .reset(rst), .dv(7'b1000000), .increment(addPoint
    ), .display(HEX0), .cycle(lp0));
53      single_hex d1(.clk(clkSelect), .reset(rst), .dv(7'b1000000), .increment(lp0), .
    display(HEX1), .cycle(lp1));
54      single_hex d2(.clk(clkSelect), .reset(rst), .dv(7'b1000000), .increment(lp1), .
    display(HEX2), .cycle(lp2));
55
56
57      always_comb begin
58          RedPixels = '0;
59          RedPixels[position][13] = 1'b1;
60          RedPixels[position + 1][13] = 1'b1;
61          RedPixels[position][12] = 1'b1;
62          RedPixels[position + 1][12] = 1'b1;
63      end
```


Date: August 19, 2022 DE1_SoC.sv Project: DE1_SoC

```
64
65      always_comb begin
66          GrnPixels = '0;
67          for(int i = 0; i < 16; i++) begin
68              if(counter % 17 == 0) begin
69                  GrnPixels[15][shift] = 1'b1;
70                  GrnPixels[14][shift] = 1'b1;
71                  GrnPixels[13][shift] = 1'b1;
72                  GrnPixels[12][shift] = 1'b1;
73                  GrnPixels[11][shift] = 1'b1;
74                  GrnPixels[10][shift] = 1'b1;
75                  GrnPixels[9][shift] = 1'b1;
76                  GrnPixels[0][shift] = 1'b1;
77              end
78              else if(counter % 13 == 0) begin
```

```
79                GrnPixels[15][shift] = 1'b1;
80                GrnPixels[14][shift] = 1'b1;
81                GrnPixels[13][shift] = 1'b1;
82                GrnPixels[12][shift] = 1'b1;
83                GrnPixels[11][shift] = 1'b1;
84                GrnPixels[10][shift] = 1'b1;
85                GrnPixels[1][shift] = 1'b1;
86                GrnPixels[0][shift] = 1'b1;
87             end
88             else if(counter % 11 == 0) begin
89                GrnPixels[13][shift] = 1'b1;
90                GrnPixels[14][shift] = 1'b1;
91                GrnPixels[15][shift] = 1'b1;
92                GrnPixels[0][shift] = 1'b1;
93                GrnPixels[1][shift] = 1'b1;
94                GrnPixels[2][shift] = 1'b1;
95                GrnPixels[12][shift] = 1'b1;
96                GrnPixels[11][shift] = 1'b1;
97
98             end
99             else if(counter % 7 == 0) begin
100               GrnPixels[0][shift] = 1'b1;
101               GrnPixels[1][shift] = 1'b1;
102               GrnPixels[2][shift] = 1'b1;
103               GrnPixels[3][shift] = 1'b1;
104               GrnPixels[4][shift] = 1'b1;
105               GrnPixels[5][shift] = 1'b1;
106               GrnPixels[6][shift] = 1'b1;
107               GrnPixels[7][shift] = 1'b1;
108            end
109            else if(counter % 5 == 0) begin
110               GrnPixels[15][shift] = 1'b1;
111               GrnPixels[0][shift] = 1'b1;
112               GrnPixels[1][shift] = 1'b1;
113               GrnPixels[2][shift] = 1'b1;
114               GrnPixels[3][shift] = 1'b1;
115               GrnPixels[4][shift] = 1'b1;
116               GrnPixels[5][shift] = 1'b1;
117               GrnPixels[6][shift] = 1'b1;
118
119            end
120            else if(counter % 3 == 0) begin
121               GrnPixels[14][shift] = 1'b1;
122               GrnPixels[15][shift] = 1'b1;
123               GrnPixels[0][shift] = 1'b1;
124               GrnPixels[1][shift] = 1'b1;
125               GrnPixels[2][shift] = 1'b1;
126               GrnPixels[3][shift] = 1'b1;
127               GrnPixels[4][shift] = 1'b1;
128               GrnPixels[5][shift] = 1'b1;
129
130            end
131            else if(counter % 2 == 0) begin
```



```
132               GrnPixels[13][shift] = 1'b1;
133               GrnPixels[14][shift] = 1'b1;
134               GrnPixels[15][shift] = 1'b1;
135               GrnPixels[0][shift] = 1'b1;
136               GrnPixels[1][shift] = 1'b1;
137               GrnPixels[2][shift] = 1'b1;
138               GrnPixels[3][shift] = 1'b1;
139               GrnPixels[4][shift] = 1'b1;
140
141            end
142            else begin
143               GrnPixels[12][shift] = 1'b1;
144               GrnPixels[13][shift] = 1'b1;
145               GrnPixels[14][shift] = 1'b1;
146               GrnPixels[15][shift] = 1'b1;
147               GrnPixels[0][shift] = 1'b1;
148               GrnPixels[1][shift] = 1'b1;
149               GrnPixels[2][shift] = 1'b1;
150               GrnPixels[3][shift] = 1'b1;
151            end
152            if(shift > 7) begin
153                  GrnPixels[15][shift-8] = 1'b1;
154                  GrnPixels[14][shift-8] = 1'b1;
155                  GrnPixels[13][shift-8] = 1'b1;
156                  GrnPixels[12][shift-8] = 1'b1;
157                  GrnPixels[11][shift-8] = 1'b1;
158                  GrnPixels[10][shift-8] = 1'b1;
159                  GrnPixels[9][shift-8] = 1'b1;
160                  GrnPixels[0][shift-8] = 1'b1;
161               end
162            if(shift < 8 & counter > 1) begin
163                  GrnPixels[15][shift+8] = 1'b1;
164                  GrnPixels[14][shift+8] = 1'b1;
165                  GrnPixels[13][shift+8] = 1'b1;
166                  GrnPixels[12][shift+8] = 1'b1;
167                  GrnPixels[11][shift+8] = 1'b1;
168                  GrnPixels[10][shift+8] = 1'b1;
169                  GrnPixels[9][shift+8] = 1'b1;
170                  GrnPixels[0][shift+8] = 1'b1;
171            end
172         end
173      end
174
175   endmodule
```

```
176
177  module DE1_SoC_testbench ();
178     logic CLOCK_50;
179     logic [3:0] KEY;
180     logic [9:0] SW;
181     logic [9:0] LEDR;
182     logic [6:0] HEX0;
183     logic [6:0] HEX1;
184     logic [6:0] HEX2;
185     logic [35:0] GPIO_1;
186
187     DE1_SoC dut (CLOCK_50, GPIO_1, HEX0, HEX1, HEX2, LEDR, KEY, SW);
188
189     // Set up the clock
190     parameter CLOCK_PERIOD=100;
191
192    initial CLOCK_50=1;
193
194    always begin
195
196       #(CLOCK_PERIOD/2); CLOCK_50 = ~CLOCK_50;
197
198    end
199
```



```
200     // Set up the inputs to the design (each line is a clock cycle)
201
202     initial begin
203
204         SW[9] <= 1;                    @(posedge CLOCK_50);
205         SW[9] <= 0;                    @(posedge CLOCK_50);
206         SW[8:0] <= 9'b100000000;       @(posedge CLOCK_50);
207                                        @(posedge CLOCK_50);
208         KEY[0] <= 1;                   @(posedge CLOCK_50);
209         KEY[0]<= 0;                    @(posedge CLOCK_50);
210         KEY[0] <= 1;                   @(posedge CLOCK_50);
211         KEY[0] <= 0;                   @(posedge CLOCK_50);
212         KEY[0] <= 1;                   @(posedge CLOCK_50);
213         KEY[0] <= 0;                   @(posedge CLOCK_50);
214         KEY[0] <= 1;                   @(posedge CLOCK_50);
215         KEY[0] <= 0;                   @(posedge CLOCK_50);
216         KEY[0] <= 1;                   @(posedge CLOCK_50);
217         KEY[0] <= 0;                   @(posedge CLOCK_50);
218         KEY[0] <= 1;                   @(posedge CLOCK_50);
219         KEY[0] <= 0;                   @(posedge CLOCK_50);
220         KEY[0] <= 1;                   @(posedge CLOCK_50);
221         SW[9] <= 1;                    @(posedge CLOCK_50);
222         SW[9] <= 0;                    @(posedge CLOCK_50);
223         KEY[0] <= 0;                   @(posedge CLOCK_50);
224         KEY[0] <= 1;                   @(posedge CLOCK_50);
225         KEY[0] <= 0;                   @(posedge CLOCK_50);
226         KEY[0] <= 1;                   @(posedge CLOCK_50);
227         KEY[0] <= 0;                   @(posedge CLOCK_50);
228         KEY[0] <= 1;                   @(posedge CLOCK_50);
229         KEY[0] <= 0;                   @(posedge CLOCK_50);
230         KEY[0] <= 1;                   @(posedge CLOCK_50);
231         KEY[0] <= 0;                   @(posedge CLOCK_50);
232         KEY[0] <= 1;                   @(posedge CLOCK_50);
233         KEY[0] <= 0;                   @(posedge CLOCK_50);
234         KEY[0] <= 1;                   @(posedge CLOCK_50);
235         KEY[0] <= 0;                   @(posedge CLOCK_50);
236         KEY[0] <= 1;                   @(posedge CLOCK_50);
237         KEY[0] <= 0;                   @(posedge CLOCK_50);
238         KEY[0] <= 1;                   @(posedge CLOCK_50);
239         KEY[0] <= 0;                   @(posedge CLOCK_50);
240         KEY[0] <= 1;                   @(posedge CLOCK_50);
241         KEY[0] <= 0;                   @(posedge CLOCK_50);
242         SW[9] <= 1;
243       $stop;          //End the simulation
244
245     end
246
247
248  endmodule
249
250  module DFFs( clk, reset, D, Q);
251     output logic Q;
252     input logic clk, reset, D;
253
254     parameter NumOfDff = 2;
255     logic [ NumOfDff - 1 : 0 ] q;
256
257     always_ff @( posedge clk ) begin
258        if ( reset == 1'b1 ) begin
259           q <= '0;
260        end else begin
261           q <= { q[ NumOfDff - 2 : 0], D };
262        end
263     end
264
265     assign Q = q[ NumOfDff - 1 ];
266
267  endmodule
```

```
268
269   module Input (clk, reset, KEYS, D);
270      input logic clk, reset, KEYS;
271      output logic D;
272      Press up (.clk, .reset, .inputButton(KEYS) , .out(D));
273
274
275   endmodule
276
277   module Input_testbench();
278      logic clk, reset, KEYS;
279      logic D;
280
281      parameter CLOCK_PERIOD=100;
282      initial begin
283         clk <= 0;
284         forever #(CLOCK_PERIOD/2) clk <= ~clk;
285      end
286
287      Input dut (clk, reset, KEYS, D);
288
289
290
291      initial begin
292            reset <= 1;  KEYS <= 0;       @(posedge clk);
293                                          @(posedge clk);
294            reset <= 0;                 @(posedge clk);
295                                        @(posedge clk);
296                                        @(posedge clk);
297            KEYS <= 1;                  @(posedge clk);
298                                          @(posedge clk);
299                                        @(posedge clk);
300                                       @(posedge clk);
301            reset <= 1;  KEYS<= 0;      @(posedge clk);
302                                        @(posedge clk);
303                                       @(posedge clk);
304                                       @(posedge clk);
305            KEYS <= 1;                 @(posedge clk);
306                                    @(posedge clk);
307            reset<=1;               @(posedge clk);
308                                    @(posedge clk);
309
310
311      $stop;
312      end
313
314   endmodule
315
316   module LEDDriver #(parameter FREQDIV = 0) (GPIO_1, RedPixels, GrnPixels, EnableCount
      , clk, rst);
317       output logic [35:0] GPIO_1;
318       input logic [15:0][15:0] RedPixels ;
319       input logic [15:0][15:0] GrnPixels ;
320       input logic EnableCount, clk, rst;
321
322       logic [(FREQDIV + 3):0] Counter;
323       logic [3:0] RowSelect;
324       assign RowSelect = Counter[(FREQDIV + 3):FREQDIV];
325
326       always_ff @(posedge clk)
327       begin
328           if(rst) Counter <= 'b0;
329           if(EnableCount) Counter <= Counter + 1'b1;
330       end
331
332       assign GPIO_1[35:32] = RowSelect;
333       assign GPIO_1[31:16] = { GrnPixels[RowSelect][0], GrnPixels[RowSelect][1],
      GrnPixels[RowSelect][2], GrnPixels[RowSelect][3], GrnPixels[RowSelect][4], GrnPixels
```



```
      [RowSelect][5], GrnPixels[RowSelect][6], GrnPixels[RowSelect][7], GrnPixels[
      RowSelect][8], GrnPixels[RowSelect][9], GrnPixels[RowSelect][10], GrnPixels[
      RowSelect][11], GrnPixels[RowSelect][12], GrnPixels[RowSelect][13], GrnPixels[
      RowSelect][14], GrnPixels[RowSelect][15] };
334       assign GPIO_1[15:0] = { RedPixels[RowSelect][0], RedPixels[RowSelect][1],
      RedPixels[RowSelect][2], RedPixels[RowSelect][3], RedPixels[RowSelect][4], RedPixels
      [RowSelect][5], RedPixels[RowSelect][6], RedPixels[RowSelect][7], RedPixels[
      RowSelect][8], RedPixels[RowSelect][9], RedPixels[RowSelect][10], RedPixels[
      RowSelect][11], RedPixels[RowSelect][12], RedPixels[RowSelect][13], RedPixels[
      RowSelect][14], RedPixels[RowSelect][15] };
335   endmodule
336
337   module LEDDriver_Test();
338       logic clk, rst, EnableCount;
339       logic [15:0][15:0]RedPixels;
340       logic [15:0][15:0]GrnPixels;
341       logic [35:0] GPIO_1;
342
343       LEDDriver #(.FREQDIV(2)) Driver(.GPIO_1, .RedPixels, .GrnPixels, .EnableCount, .
      clk, .rst);
344
345       initial
```

```
346         begin
347             clk <= 1'b0;
348             forever #50 clk <= ~clk;
349         end
350
351         initial
352         begin
353             EnableCount <= 1'b0;
354             RedPixels <= '{default:0};
355             GrnPixels <= '{default:0};
356             @(posedge clk);
357
358             rst <= 1; @(posedge clk);
359             rst <= 0; @(posedge clk);
360             @(posedge clk); @(posedge clk); @(posedge clk);
361
362             GrnPixels[1][1] <= 1'b1; @(posedge clk);
363             EnableCount <= 1'b1; @(posedge clk); #1000;
364             RedPixels[2][2] <= 1'b1;
365             RedPixels[2][3] <= 1'b1;
366             GrnPixels[2][3] <= 1'b1; @(posedge clk); #1000;
367             EnableCount <= 1'b0; @(posedge clk); #1000;
368             GrnPixels[1][1] <= 1'b0; @(posedge clk);
369             $stop;
370
371         end
372     endmodule
373
374     module LEDDriver_TestPhysical(CLOCK_50, rst, Speed, GPIO_1);
375         input logic CLOCK_50, rst;
376         input logic [9:0] Speed;
377         output logic [35:0] GPIO_1;
378         logic [15:0][15:0]RedPixels;
379         logic [15:0][15:0]GrnPixels;
380         logic [31:0] Counter;
381         logic EnableCount;
382
383         LEDDriver #(.FREQDIV(15)) Driver (.clk(CLOCK_50), .rst, .EnableCount, .RedPixels
    , .GrnPixels, .GPIO_1);
384
385         //                          F E D C B A 9 8 7 6 5 4 3 2 1 0
386         assign RedPixels[00] = '{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
387         assign RedPixels[01] = '{1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1};
388         assign RedPixels[02] = '{1,0,1,1,1,1,1,1,1,1,1,1,1,0,1};
389         assign RedPixels[03] = '{1,0,1,1,0,0,0,0,0,0,0,1,1,0,1};
390         assign RedPixels[04] = '{1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1};
```



```
391         assign RedPixels[05] = '{1,0,1,0,1,1,0,0,0,0,1,1,0,1,0,1};
392         assign RedPixels[06] = '{1,0,1,0,1,0,1,1,1,1,0,1,0,1,0,1};
393         assign RedPixels[07] = '{1,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1};
394         assign RedPixels[08] = '{1,0,1,0,1,0,1,1,0,1,0,1,0,1,0,1};
395         assign RedPixels[09] = '{1,0,1,0,1,0,1,1,1,1,0,1,0,1,0,1};
396         assign RedPixels[10] = '{1,0,1,0,1,1,0,0,0,0,1,1,0,1,0,1};
397         assign RedPixels[11] = '{1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1};
398         assign RedPixels[12] = '{1,0,1,1,0,0,0,0,0,0,0,0,1,1,0,1};
399         assign RedPixels[13] = '{1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1};
400         assign RedPixels[14] = '{1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1};
401         assign RedPixels[15] = '{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
402
403         assign GrnPixels[00] = '{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
404         assign GrnPixels[01] = '{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0};
405         assign GrnPixels[02] = '{0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0};
406         assign GrnPixels[03] = '{0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0};
407         assign GrnPixels[04] = '{0,1,0,1,1,0,0,0,0,0,0,1,1,0,1,0};
408         assign GrnPixels[05] = '{0,1,0,1,0,1,1,1,1,1,1,0,1,0,1,0};
409         assign GrnPixels[06] = '{0,1,0,1,0,1,1,0,0,1,1,0,1,0,1,0};
410         assign GrnPixels[07] = '{0,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0};
411         assign GrnPixels[08] = '{0,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0};
412         assign GrnPixels[09] = '{0,1,0,1,0,1,1,0,0,1,1,0,1,0,1,0};
413         assign GrnPixels[10] = '{0,1,0,1,0,1,1,1,1,1,1,0,1,0,1,0};
414         assign GrnPixels[11] = '{0,1,0,1,1,0,0,0,0,0,0,1,1,0,1,0};
415         assign GrnPixels[12] = '{0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0};
416         assign GrnPixels[13] = '{0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0};
417         assign GrnPixels[14] = '{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0};
418         assign GrnPixels[15] = '{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
419
420         always_ff @(posedge CLOCK_50)
421         begin
422             if(rst) Counter <= 'b0;
423             else
424             begin
425                 Counter <= Counter + 1'b1;
426                 if(Counter >= Speed)
427                 begin
428                     EnableCount <= 1'b1;
429                     Counter <= 'b0;
430                 end
431                 else EnableCount <= 1'b0;
432             end
433         end
434     endmodule
435
436     module Press (clk, reset, inputButton, out);
437        input logic clk, reset;
438        input logic inputButton;
439        output logic out;
440        logic tempOut;
441
```

```
442     DFFs d0 (.clk, .reset, .D(~inputButton), .Q(tempOut));
443
444
445     enum logic [1:0]{off = 2'b00, semi= 2'b01, on = 2'b11} ps, ns;
446
447     always_comb begin
448        case (ps)
449           off:
450              if(tempOut) begin                                    ns = semi;
451              end else begin                                       ns = off;
452              end
453
454           semi:
455              if(tempOut) begin                                    ns = semi;
456              end else begin                                       ns = on;
457              end
458           on:
```



```
459              begin                                                ns = off;
460              end
461
462           //default: ns = off;
463        endcase
464     end
465
466     assign out = (ps == on);
467
468     always_ff @(posedge clk) begin
469        if (reset)           ps <= off;
470        else                 ps <= ns;
471     end
472  endmodule
473
474  module bird (clk, rst, button, gameover, position);
475
476           input logic clk, rst, button, gameover;
477           output int position;
478
479           int gravity;
480
481           always_ff @ (posedge clk) begin
482              if (rst) begin
483                 position <= 5;
484                 gravity <= 0;
485              end
486              else begin
487                 if (gameover) begin
488                    position <= position;
489                 end
490                 else if (position == 14 & ~button) begin
491                    position <= 14;
492                 end
493                 else if (position == 0 & button) begin
494                    position <= 0;
495                 end
496                 else if (button) begin
497                    position <=  position - 1;
498                 end
499                 else if (~button) begin
500                    gravity <= gravity + 1;
501                    if(gravity % 500 == 0) begin
502                       position <= position + 1;
503                    end
504                 end
505              end
506           end
507  endmodule
508
509  module bird_testbench();
510     logic clk, rst, button, gameover;
511     int position;
512
513     bird dut (.clk, .rst, .button, .position, .gameover);
514
515        parameter CLOCK_PERIOD=100;
516        initial begin
517           clk <= 0;
518           forever #(CLOCK_PERIOD/2) clk <= ~clk;
519        end
520
521        initial begin
522           rst <= 1;  @(posedge clk);
523           rst <= 0; @(posedge clk);
524           repeat (16) begin
525              button <= 1; @(posedge clk);
526              button<= 0; @(posedge clk);
```

```
527             button<= 0; @(posedge clk);
528             button<= 0; @(posedge clk);
529             button <= 1; @(posedge clk);
530             button<= 0; @(posedge clk);
531             button <= 1; @(posedge clk);
532          end
533          gameover <= 1; @(posedge clk);
534          button <= 1; @(posedge clk);
535          button<= 0; @(posedge clk);
536          button <= 1; @(posedge clk);
537          button <= 0;  @(posedge clk);
538          rst <= 1;  @(posedge clk);
539          rst <= 0; @(posedge clk);
540          gameover <= 0; @(posedge clk);
541       $stop;
542       end
543   endmodule
544
545   /* divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ...
546     [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz, ... */
547   module clock_divider (clk, rst, divided_clocks);
548    input logic clk, rst;
549    output logic [31:0] divided_clocks = 0;
550
551    always_ff @(posedge clk) begin
552      divided_clocks <= divided_clocks + 1;
553    end
554
555   endmodule
556
557   module crash (clk, reset, gA, rA, gameover, addPoint);
558      input logic clk, reset;
559      input logic [15:0][15:0] gA, rA;
560      output logic gameover, addPoint;
561
562      reg psc, nsc;
563      reg [15:0] psp, nsp;
564
565      always @(*) begin
566         nsc =  (((gA[0] & rA[0])  != 16'b0000000000000000) | ((gA[1] & rA[1]) !=
     16'b0000000000000000)  | ((gA[2] & rA[2]) != 16'b0000000000000000)
567            |((gA[3] & rA[3])  != 16'b0000000000000000) | ((gA[4] & rA[4]) !=
     16'b0000000000000000)  | ((gA[5] & rA[5]) != 16'b0000000000000000)
568            |((gA[6] & rA[6])  != 16'b0000000000000000) | ((gA[7] & rA[7]) !=
     16'b0000000000000000)  | ((gA[8] & rA[8]) != 16'b0000000000000000)
569            |((gA[9] & rA[9])  != 16'b0000000000000000) | ((gA[10] & rA[10]) !=
     16'b0000000000000000)| ((gA[11] & rA[11]) != 16'b0000000000000000)
570            |((gA[12] & rA[12]) != 16'b0000000000000000)| ((gA[13] & rA[13]) !=
     16'b0000000000000000)| ((gA[14] & rA[14]) != 16'b0000000000000000)
571            |((gA[15] & rA[15]) != 16'b0000000000000000) | rA[15] !=
     16'b0000000000000000);
572            nsp = gA[0][14];
573      end
574      assign gameover = psc;
575      assign addPoint = ~(psp == 1'b0) & (gA[0][14] != 1'b1);
576
577      always @(posedge clk)
578         if(reset) begin
579            psc <= 1'b0;
580            psp <= 1'b0;
581         end
582         else begin
583            psc <= nsc;
584            psp <= nsp;
585         end
586   endmodule
587
588
```



```
589   module crash_testbench ();
590      logic clk, reset;
591      logic [15:0][15:0] gA, rA;
592      logic resetgame, addPoint;
593
594      crash dut (clk, reset, gA, rA, resetgame, addPoint);
595
596
597      // Set up the clock
598      parameter CLOCK_PERIOD = 100;
599      initial begin
600         clk <= 0;
601         forever #(CLOCK_PERIOD/2) clk <= ~clk;
602      end
603
604      // Set up the inputs to the design. Each line is a clock cycle.
605      initial begin
606         rA <= {     {16'b0000000000000000},
607                     {16'b0000000000000000},
608                     {16'b0000000000000000},
609                     {16'b0000000000000000},
610                     {16'b0000000000000000},
611                     {16'b0000000000000000},
612                     {16'b0000000000000000},
613                     {16'b0000000000000000}};
614         gA <= {     {16'b0000000000000000},
615                     {16'b0000000000000000},
616                     {16'b0000000000000000},
617                     {16'b0000000000000000},
```

```
618                     {16'b0000000000000000},
619                     {16'b0000000000000000},
620                     {16'b0000000000000000},
621                     {16'b0000000000000000}};
622         reset <= 1; @(posedge clk);
623                     @(posedge clk);
624         reset <= 0; @(posedge clk);
625         rA[0] <= 16'b0100000001010101; gA[0] <= 16'b0100000000000000;   @(posedge clk
    );  //crash
626         rA[0] <= 16'b0100000010110101; gA[0] <= 16'b0010000000000000;   @(posedge clk);
627         rA[1] <= 16'b1000000011111111; gA[1] <= 16'b1000000000000000;   @(posedge clk
    );  // collosion
628         rA[1] <= 16'b0100000010100011; gA[1] <= 16'b0000010000000000;   @(posedge clk);
629         rA[2] <= 16'b0010000011010101; gA[2] <= 16'b0010000000000000;   @(posedge clk
    );  //crash
630         rA[2] <= 16'b0100000010101111; gA[2] <= 16'b1000000000000000;   @(posedge clk);
631         rA[3] <= 16'b0001000010101111; gA[3] <= 16'b0001000000000000;   @(posedge clk
    );  // crash
632         rA[3] <= 16'b0100000011010101; gA[3] <= 16'b1000000000000000;   @(posedge clk);
633         rA[4] <= 16'b0000100011010101; gA[4] <= 16'b0000100000000000;   @(posedge clk
    );  //crash
634         rA[4] <= 16'b0100000010101010; gA[4] <= 16'b1000000000000000;   @(posedge clk);
635         rA[5] <= 16'b0000010010101011; gA[5] <= 16'b0000010000000000;   @(posedge clk
    );  // crash
636         rA[5] <= 16'b0100000010110101; gA[5] <= 16'b1000000000000000;   @(posedge clk);
637         rA[6] <= 16'b0000001010110101; gA[6] <= 16'b0000001000000000;   @(posedge clk
    );  // crash
638         rA[6] <= 16'b0100000000000000; gA[6] <= 16'b1000000000000000;   @(posedge clk);
639         rA[7] <= 16'b0000000100000000; gA[7] <= 16'b0000000100000000;   @(posedge clk
    );  // crash
640         rA[7] <= 16'b0100000000000000; gA[2] <= 16'b1000000000000000;   @(posedge clk);
641         repeat (16) begin
642               ; @(posedge clk);
643               rA[5][12] <= 1'b1; @(posedge clk);
644               rA[5][13] <= 1'b1; @(posedge clk);
645               rA[6][12] <= 1'b1; @(posedge clk);
646               rA[6][13] <= 1'b1; @(posedge clk);
647               gA[0] <= 16'b1010110101100010; @(posedge clk);
648            end
```



```
649       $stop; // End the simulation
650       end
651
652   endmodule
653
654   module pipeShift (clk, rst, gameover, shift, counter);
655
656            input logic clk, rst, gameover;
657            output int shift;
658            output int counter;
659
660            int count;
661
662            always_ff @ (posedge clk) begin
663               if (rst) begin
664                  shift <= 0;
665                  count <= 0;
666                  counter <= 0;
667               end
668               else begin
669                  if (gameover) begin
670                     shift <= shift;
671                  end
672                  else if (shift == 16) begin
673                     shift <= 0;
674                     counter <= counter + 1;
675                  end
676                  else begin
677                     count <= count + 1;
678                     if(count % 200 == 0) begin
679                        shift <= shift + 1;
680                     end
681                  end
682               end
683            end
684   endmodule
685
686   module pipeShift_testbench();
687      logic clk, rst, gameover;
688      int shift, counter;
689
690      pipeShift dut (.clk, .rst, .gameover);
691
692         parameter CLOCK_PERIOD=100;
693         initial begin
694            clk <= 0;
695            forever #(CLOCK_PERIOD/2) clk <= ~clk;
696         end
697
698         initial begin
699            rst <= 1;  @(posedge clk);
700            rst <= 0; @(posedge clk);
701            repeat (16) @(posedge clk);
702            gameover <= 1; @(posedge clk);
703            rst <= 1;  @(posedge clk);
704            rst <= 0; @(posedge clk);
705            gameover <= 0; @(posedge clk);
```

```
706           repeat (16) @(posedge clk);
707        $stop;
708        end
709  endmodule
710
711  module single_hex(clk, reset, dv, increment, display, cycle);
712     input clk, reset, increment;
713     input [6:0] dv;
714     output cycle;
715     output [6:0] display;
716
```



```
717      parameter   zero  = 7'b1000000,
718                  one   = 7'b1111001,
719                  two   = 7'b0100100,
720                  three = 7'b0110000,
721                  four  = 7'b0011001,
722                  five  = 7'b0010010,
723                  six   = 7'b0000010,
724                  seven = 7'b1111000,
725                  eight = 7'b0000000,
726                  nine  = 7'b0010000;
727
728      reg [6:0] ps, ns;
729
730      always @(*)
731         if(increment)
732            case(ps)
733               zero:    ns = one;
734               one:     ns = two;
735               two:     ns = three;
736               three:   ns = four;
737               four:    ns = five;
738               five:    ns = six;
739               six:     ns = seven;
740               seven:   ns = eight;
741               eight:   ns = nine;
742               nine:    ns = zero;
743               default: ns = one;
744            endcase
745         else
746            ns = ps;
747
748      assign display[6:0] = ps[6:0];
749      assign cycle = (ps[6:0] == nine) & (increment);
750
751      always @(posedge clk)
752         if(reset)
753            ps <= dv;
754         else
755            ps <= ns;
756  endmodule
757
758  module single_hex_testbench();
759     reg clk, reset, increment;
760     reg [6:0] dv;
761     wire [6:0] display;
762     wire cycle;
763
764     single_hex dut(clk, reset, dv, increment, display, cycle);
765
766     // Set up the clock.
767     parameter CLOCK_PERIOD=100;
768     initial clk=1;
769     always begin
770        #(CLOCK_PERIOD/2);
771        clk = ~clk;
772     end
773
774     initial begin
775
776        reset <= 1;
777              dv <= 7'b1111111; increment <= 1;   @(posedge clk);
778              dv <= 7'b0000000;                   @(posedge clk);
779        reset <= 0;                               @(posedge clk);
780                                                  @(posedge clk);
781                                  increment <= 0;   @(posedge clk);
782                                  increment <= 1;   @(posedge clk);
783                                                  @(posedge clk);
784                                                  @(posedge clk);
```



```
785                                                  @(posedge clk);
786                                                  @(posedge clk);
787                                                  @(posedge clk);
788                                                  @(posedge clk);
789                                                  @(posedge clk);
790                                  increment <= 0;   @(posedge clk);
791                                  increment <= 1;   @(posedge clk);
```

```
791                                 increment <= 1;   @(posedge clk);
792                                         @(posedge clk);
793         $stop;
794     end
795 endmodule
796
797
798
```