

EE 271 Design Project Report: Flappy Bird

Diego Cervantes

Demo TA: Ryan Wang

June 6, 2022 @2:30 pm

Department of Computer & Electrical Engineering, University of Washington

EE 271: Digital Circuits and Systems

Prof. Denise Wilson

June 6, 2022

Design Problem:

The popular mobile phenomenon, Flappy Bird, was created using modules from previous labs and newly created ones. Flappy bird allows the user to play as a bird called Olly Oriole and try to achieve the highest possible score by “flapping” through the sky, avoiding the pipes. Olly is a single red LEDR, and the pipes floating from right to left are green. The user controls Olly by pressing key zero (KEY [0]). If the user does not press the key, then Olly will begin to descend downwards. Descending all the way to the “floor” will not cause Olly to die (end game). To begin playing, the user must use switch 9 (SW [9]); and this switch will be used as the “restart game” input. Three Hex displays are used to keep track of the gaps (points) cleared with a max score of 999. If a user surpasses this figure, the game will continue to be playable, but the score will no longer be trackable. Fifteen different pipe configurations have been created and will be outputted pseudo-randomly. The design of this lab uses many of the relevant concepts found in a digital and systems curriculum. Meta-stability, combinational logic, flip-flop implementation, and state diagrams are all concepts that are useful to complete a project like this.

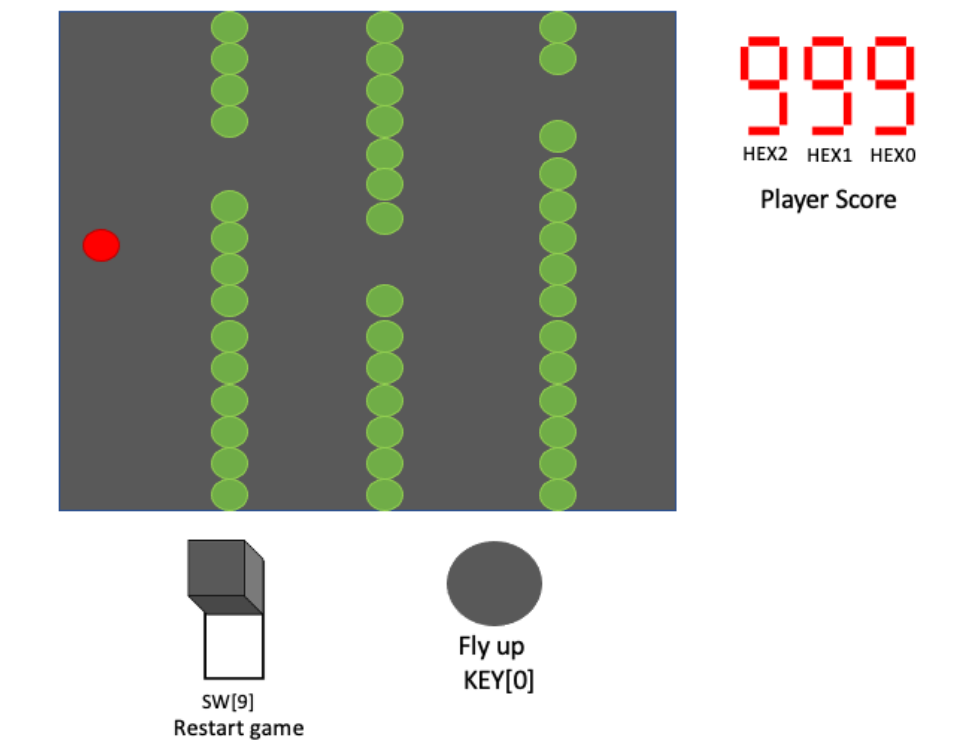


Figure 1. User-Level Block Diagram

Methods and Procedures:

Many of the modules from previous labs were used and modified to create the logic flappy bird required. For example, the modules from lab 5 controlling the center and normal lights met most of the logic required to control Olly. Quick modifications to these modules were made to account for "gravity," and slight changes to the leftmost and rightmost lights were made. The modules accounting for meta-stability and the user input from lab 7 were also recycled for the same reasons as mentioned above. A new module to generate the green pipes is used, while another newly created module serves to move the pipes across the LED array. To avoid the pipes from appearing in the same order all the time, a new LFSR module is implemented with a waiting gap of 4 columns between pipes to make the game playable. A module serving as a collision detector would then call the game over module to end the game. Whenever Olly surpassed a gap, a scoring module was called to track the player's score. The DE1_SoC was modified to contain a wide range of clocks, including one that controlled the entire system and another clock to simulate gravity.

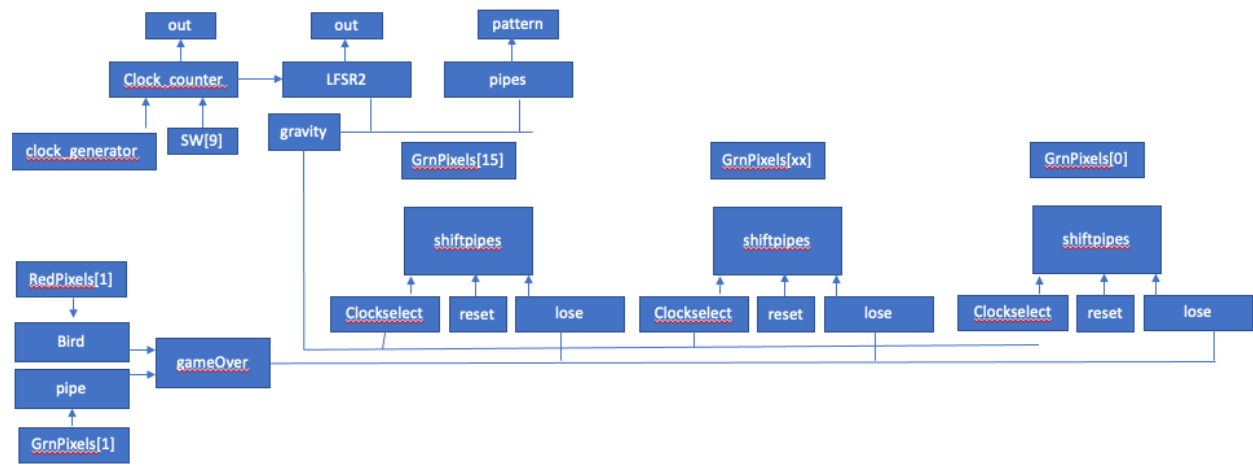


Figure 2. Designer-Level Block Diagram

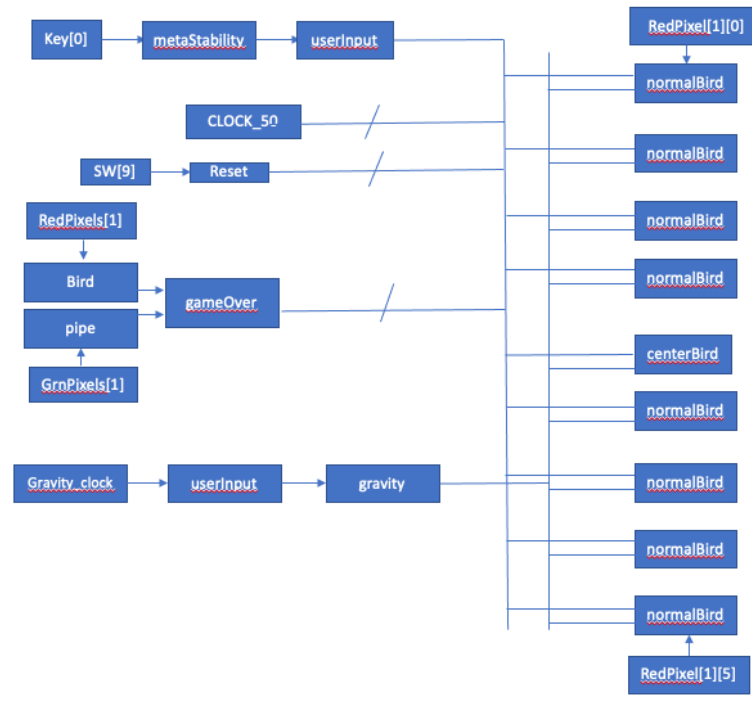


Figure 3. Designer-Level Block Diagram continued

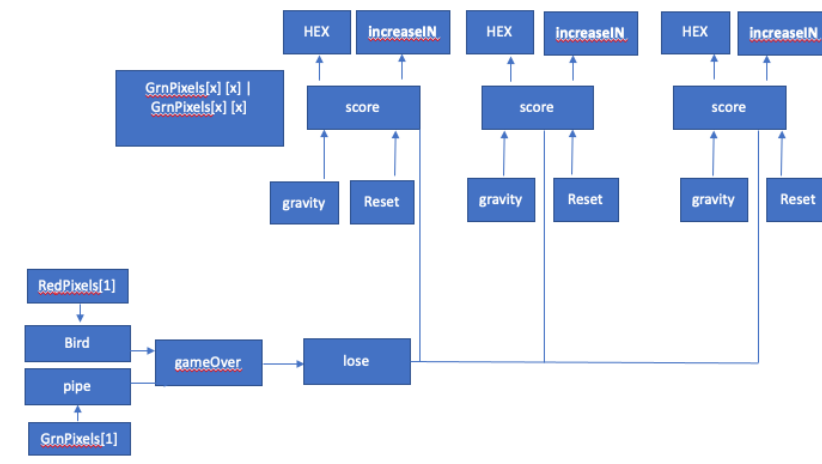


Figure 4. Designer-Level Block Diagram continued

Results:

Unfortunately, a simulation of the top-level module was unsuccessful. Reasons were unknown but the actual project on the board works as it should.

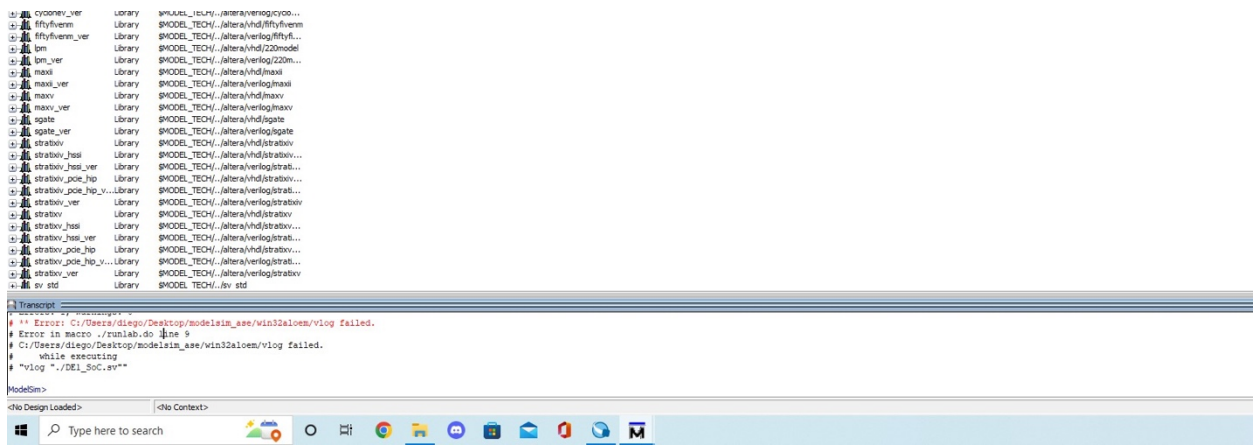


Figure 7. Simulation for top level module does not simulate successfully

```

nrunlab - Notepad
File Edit Format View Help
▶ Create work library
vlib work

# Compile Verilog
# All Verilog files that are part of this design should have
# their own "vlog" line below.
vlog "./mux2_1.v"
vlog "./mux4_1.v"
vlog "./DE1_SoC.v"
vlog "./fred_display.v"
vlog "./callseg7.v"
vlog "./seg7.v"
vlog "./simple.v"
vlog "./clock_divider.v"
vlog "./wind_lights.v"
vlog "./metaStable.v"
vlog "./playerInput.v"
vlog "./centerLight.v"
vlog "./normallight.v"
vlog "./victory.v"
vlog "./TugOfWar.v"
vlog "./counters.v"
vlog "./comparator.v"
vlog "./LFSR.v"
vlog "./computer.v"
vlog "./normalBird.v"
vlog "./shift_pipes.v"
vlog "./clock_counter.v"
vlog "./pipes.v"
vlog "./centerBird.v"
vlog "./gameOver.v"
vlog "./score.v"
vlog "./LEDBriver.v"
vlog "./LFSR2.v"

# Call vsyn to invoke simulator
# Make sure the last item on the line is the name of the
# testbench module you want to execute.
vsyn -voptargs="+acc" -t 1ps -lib work DE1_SoC_testbench

# Source the wave do file
# This should be the file that sets up the signal window for
# the module you are testing.
do DE1_SoC_wave.do

# Set the window types
view wave
view structure
view signals

# Run the simulation
run -all

# End

```

Reflection:

During the whole EE271 course labs, there were many skills learned that can be taken away from them and used for future classes in embedded systems. The knowledge that is likely to be retained from labs is the importance of state machines, combinational and sequential circuits, logic design, and the different techniques for modeling systems with Verilog. The state machines are used to map out every state of a process and look at the behavior of a machine. For combinational circuits, there is no feedback among the inputs and outputs. Therefore, combinational circuits are mainly used to do arithmetic (boolean algebra). On the other hand, sequential circuits are used to store important data. However, many systems consist of a mix of both combinational and sequential circuits which I could see being helpful for future embedded systems classes. Logic design in general is useful for building complex electronic components that are then used for digital systems. Creating the testbenches and analyzing the results on Modelsim helps to verify whether the machine is behaving the way it should. This will be very useful in future embedded systems classes, especially to analyze where something may be going wrong.

Throughout the quarter, I have learned many new topics about digital circuits and systems. Some of those included, logic design, boolean algebra, kmaps, flip flops, state machines, muxes and decoders. I was able to understand all of these topics pretty well and are important to know as fundamental knowledge for higher level embedded systems classes. Although there were still things I struggled with, I was able to apply each one of these topics into my labs.

Throughout the progress of the labs, there were things I became good at but also things I could continue to work on to improve the quality of each lab results. As for my strengths in terms of laboratory skills, I became good at debugging my code, loading and testing my code on the FPGA board, and implementing my code based off of the state machines I created. Whenever I ran the analysis on my code and encountered issues, I was able to find the errors fairly quickly and saved me some time. Uploading and testing my code was also a great strength of mine as I was able to execute different test cases on the board. In terms of the actual Verilog language, I was able to learn it quickly as I had never worked with it before. In terms of my weaknesses, I struggled with Modelsim and creating enough test cases for the modules. Sometimes I would not be able to get my Modelsim results to show up correctly so I couldn't analyze the results on Modelsim very well. I would mainly analyze the results on the FPGA board. Another one of my weaknesses was creating the state machines for the code that I needed to implement. In general, I struggled visualizing what each state needed to look like.

Creating complex designs that involve more than one finite state machine or combinational logic block were time consuming to create and require careful analysis of the expected behavior of each state machine or combinational logic block. In the final project, there were many state machines and combinational logic blocks. The lab took a long time to create because it was important to have the correct states implemented in the code. It was important to be precise with the inputs and outputs of a machine throughout all the labs. The flappy bird project required a counter for the score, random patterns for the pipe, and different states for the bird. There was a close relationship between each state machine for the flappy bird project which shows how careful I needed to be with this complex design that involved more than one finite

state machine/combinational logic block. All of this knowledge will help moving forward as I continue to create more complex designs.

Appendix

README

- DE1_SoC.sv is the top-level module and calls on most modules
- metaStable.sv prevents metastability from affecting the system
- playerInput.sv Prevents Olly (the bird) from flying by holding the key
- clock_devider.sv generates a clock that is easy to work with
- LEDDriver.sv is the file provided to control the 16x16 LED array
- centerBird.sv is the module responsible for when Olly is in the center
- normalBird.sv is the module that looks for the position of Olly besides the center light
- LFSR2.sv is a 4-bit linear feedback register that is used as a random generator every 4 clock cycles
- shift_pipes.sv controls the movement of pipes across all rows
- score.sv controls hex0-2 to display the users score
- clock_counter.sv is a clock that checks LFSR2 to see when it is time to create a new pattern and make sure they don't repeat
- gameOver.sv serves as a collision detector, ending the game once it hits
- pipes.sv creates the pattern for the pipes

```

1  module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, LEDR, SW, KEY, GPIO_1);
2  //The LED outputs and the 7segment display HEX outputs
3  output logic [9:0] LEDR;
4  output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
5  output logic [35:0] GPIO_1;
6
7  //CLOCK_50,SW, and KEY are inputs
8  input logic CLOCK_50;
9  input logic [9:0] SW;
10 input logic [3:0] KEY;
11
12 // Turn off unused HEX displays
13 assign HEX3 = '1;
14 assign HEX4 = '1;
15 assign HEX5 = '1;
16 //assigns sw9 as reset signal
17 logic Reset;
18 assign Reset = SW[9];
19
20 logic [31:0] div_clk;
21
22 parameter whichClock = 25; //768 Hz Clock
23 parameter gravityClock = 23; //3 Hz Clock
24 parameter systemClock = 14; // 1526 Hz clock
25
26 clock_divider cdiv(.clock(CLOCK_50), .reset(Reset), .divided_clocks(div_clk));
27
28 // Clock selection;
29 // allows for easy switching between simulation and board clocks
30 logic clkSelect;
31 //Uncomment ONE of the following two lines depending on intention
32 assign clkSelect = CLOCK_50; // for simulation
33 //assign clkSelect = div_clk[whichClock]; // for synthesis on DE1_SoC board
34
35 logic gravitySelect;
36 assign gravitySelect = div_clk[gravityClock]; //simulates gravity
37 logic systemSelect;
38 assign systemSelect = div_clk[systemClock];
39
40 /* If you notice flickering, set SYSTEM_CLOCK faster.
41 However, this may reduce the brightness of the LED board. */
42
43
44 /* Set up LED board driver
45 ===== */
46 logic [15:0][15:0] RedPixels; // 16 x 16 array representing red LEDs
47 logic [15:0][15:0] GrnPixels; // 16 x 16 array representing green LEDs
48
49 /* Standard LED Driver instantiation - set once and 'forget it'.
50 See LEDDriver.sv for more info. Do not modify unless you know what you are doing! */
51 LEDDriver.Driver (.CLK(systemSelect), .RST(Reset), .EnableCount(1'b1), .RedPixels, .
GrnPixels, .GPIO_1);
52
53 /* LED board test submodule - paints the boards with a static pattern.
54 Replace with your own code driving RedPixles and GrnPixels.
55
56 KEY0 :Reset
57 ===== */
58 //button to controll 01ly
59 logic flap;
60 assign flap = ~KEY[0];
61
62 //prevents metastability from happening by passing the input of the button into a dff
63 logic input_player;
64 metaStable met1(.Clock(CLOCK_50), .Reset, .player_in(flap), .player_out(input_player));
65 //goes through metastable again
66 logic stable_player;
67 metaStable met2 (.Clock(CLOCK_50), .Reset, .player_in(input_player), .player_out(
stable_player));
68
69 logic flappy;
70 playerInput player_1(.Clock(CLOCK_50), .Reset, .final_in(stable_player), .final_out(
flappy)); //player input
71
72 logic gravityyyyyy;
73 playerInput player_2(.Clock(CLOCK_50), .Reset, .final_in(gravitySelect), .final_out(

```

```

74 gravityyyyyy));
75 logic gmeover;
76 gameOver over(.bird(RedPixels[1]), .pipe(GrnPixels[1]), .lose(gmeover));
77
78
79
80 //logic [9:0] random_num;//random generator
81 //logic restart_game;//reset game at center light after there is a winner
82
83
84 //LFSR generator(.Clock(clkSelect), .Reset(restart_game), .out(random_num));
85
86 //logic computer_push;//computer push input
87 //computer cyberPlayer(.Clock(clkSelect), .Reset, .SW(SW[8:0]), .out(computer_push));
88
89 // player1 and com1 holds result from the first DFF for both players
90 //to prevent metastability
91 //logic player1, com1;
92
93 //metaStable met1 (.Clock(clkSelect), .Reset, .player_in(~KEY[0]),
94 .player_out(player1));
95 //metaStable met2 (.Clock(clkSelect), .Reset, .player_in(computer_push),
96 .player_out(com1));
97
98 // stable_p1 and stable_p2 holds result from the second DFF for both players
99 //to prevent metastability
100 //logic player1_2nd, com1_2nd;
101
102 //Prevents holding of a key using the stable player inputs
103 //playerInput player_1 (.Clock(clkSelect), .Reset, .final_in(player1),
104 .final_out(player1_2nd));//player input
105 //playerInput player_2 (.Clock(clkSelect), .Reset, .final_in(com1),
106 .final_out(com1_2nd));//computer input
107
108 //logic RESTART1, restart1, restart2;
109 //assign RESTART1 = Reset | restart1 | restart2;//restart game condition
110 //Temporary values sent to the normal and center light modules to get the output for
111 the LEDs
112 //logic hold1, hold2, hold3, hold4, hold5, hold6, hold7, hold8, hold9;
113
114 logic lose;
115
116 // set up the lights for the bird
117 normalBird zero (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
118 (RedPixels[1][1]), .NR(1'b0), .high(1'b1), .low(1'b0), .lose, .lightOn(
119 RedPixels[1][0]));
120 normalBird one (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
121 (RedPixels[1][2]), .NR(RedPixels[1][0]), .high(1'b0), .low(1'b0), .lose, .lightOn(
122 RedPixels[1][1]));
123 normalBird two (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
124 (RedPixels[1][3]), .NR(RedPixels[1][1]), .high(1'b0), .low(1'b0), .lose, .lightOn(
125 RedPixels[1][2]));
126 normalBird three (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
127 (RedPixels[1][4]), .NR(RedPixels[1][2]), .high(1'b0), .low(1'b0), .lose, .lightOn(
128 RedPixels[1][3]));
129 normalBird four (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
130 (RedPixels[1][5]), .NR(RedPixels[1][3]), .high(1'b0), .low(1'b0), .lose, .lightOn(
131 RedPixels[1][4]));
132 normalBird five (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
133 (RedPixels[1][6]), .NR(RedPixels[1][4]), .high(1'b0), .low(1'b0), .lose, .lightOn(
134 RedPixels[1][5]));
135 normalBird six (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
136 (RedPixels[1][7]), .NR(RedPixels[1][5]), .high(1'b0), .low(1'b0), .lose, .lightOn(
137 RedPixels[1][6]));
138 centerBird seven (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
139 (RedPixels[1][8]), .NR(RedPixels[1][6]), .lose, .lightOn(
140 RedPixels[1][7]));
141 normalBird eight (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
142 (RedPixels[1][9]), .NR(RedPixels[1][7]), .high(1'b0), .low(1'b0), .lose, .lightOn(
143 RedPixels[1][8]));
144 normalBird nine (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
145 (RedPixels[1][10]), .NR(RedPixels[1][8]), .high(1'b0), .low(1'b0), .lose, .lightOn(
146 RedPixels[1][9]));
147 normalBird ten (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL

```

```

(RedPixels[1][11]), .NR(RedPixels[1][9]), .high(1'b0), .low(1'b0), .lose, .lightOn(
RedPixels[1][10]));
124   normalBird eleven (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
(RedPixels[1][12]), .NR(RedPixels[1][10]), .high(1'b0), .low(1'b0), .lose, .lightOn(
RedPixels[1][11]));
125   normalBird twelve (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
(RedPixels[1][13]), .NR(RedPixels[1][11]), .high(1'b0), .low(1'b0), .lose, .lightOn(
RedPixels[1][12]));
126   normalBird thirteen (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
(RedPixels[1][14]), .NR(RedPixels[1][12]), .high(1'b0), .low(1'b0), .lose, .lightOn(
RedPixels[1][13]));
127   normalBird fourteen (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
(RedPixels[1][15]), .NR(RedPixels[1][13]), .high(1'b0), .low(1'b0), .lose, .lightOn(
RedPixels[1][14]));
128   normalBird fifteen (.Clock(CLOCK_50), .Reset, .gravity(gravityyyyyy), .R(flappy), .NL
(1'b0), .NR(RedPixels[1][14]), .high(1'b0), .low(1'b1), .lose, .lightOn(
RedPixels[1][15]));
129
130   shift_pipes rows15 (.Clock(clkSelect), .Reset, .newpattern(bar), .lose, .out(
GrnPixels[15]));
131
132   shift_pipes rows0 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[1]), .lose, .
out(GrnPixels[0]));
133   shift_pipes rows1 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[2]), .lose, .
out(GrnPixels[1]));
134   shift_pipes rows2 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[3]), .lose, .
out(GrnPixels[2]));
135   shift_pipes rows3 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[4]), .lose, .
out(GrnPixels[3]));
136   shift_pipes rows4 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[5]), .lose, .
out(GrnPixels[4]));
137   shift_pipes rows5 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[6]), .lose, .
out(GrnPixels[5]));
138   shift_pipes rows6 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[7]), .lose, .
out(GrnPixels[6]));
139   shift_pipes rows7 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[8]), .lose, .
out(GrnPixels[7]));
140   shift_pipes rows8 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[9]), .lose, .
out(GrnPixels[8]));
141   shift_pipes rows9 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[10]), .lose, .
out(GrnPixels[9]));
142   shift_pipes rows10 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[11]), .lose, .
out(GrnPixels[10]));
143   shift_pipes rows11 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[12]), .lose, .
out(GrnPixels[11]));
144   shift_pipes rows12 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[13]), .lose, .
out(GrnPixels[12]));
145   shift_pipes rows13 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[14]), .lose, .
out(GrnPixels[13]));
146   shift_pipes rows14 (.Clock(clkSelect), .Reset, .newpattern(GrnPixels[15]), .lose, .
out(GrnPixels[14]));
147
148
149   logic generate_clock;
150   clock_counter count(.Clock(clkSelect), .Reset, .out(generate_clock));
151
152   logic [3:0] random_num; // random generator
153   LFSR2 random(.Clock(generate_clock), .Reset, .out(random_num));
154
155   logic [15:0] bar;
156   pipes pipe(.LFSR2_out(random_num), .pattern(bar), .generate_clock);
157
158   //logic [15:0] actual_pattern;
159   //random_pattern repetition (.Clock(clkSelect), .Reset, .pattern(bar),
.actualpattern(actual_pattern), .clockgenerator(generate_clock));
160
161   // set up pipes
162   //shift_pipes rows15 (.Clock(clkSelect), .Reset, .newpattern(actual_pattern), .lose,
.out(GrnPixels[15]));
163
164   gameOver endgame(.bird(RedPixels[1]), .pipe(GrnPixels[1]), .lose);
165
166   //passes the temp. signal into the score module
167   logic [6:0] fakesignal1, fakesignal2, fakesignal3;
168   logic up1, up2, up3;
169   score first_digit(.Clock(clkSelect), .Reset, .increaseIN(GrnPixels[0][5] | GrnPixels[0

```

```

170 ][1]), .lose, .HEX(fakesignal1), .increaseout(up1));
171 score_second_digit(.Clock(clkSelect), .Reset, .increaseIN(up1), .lose, .HEX(
fakesignal2), .increaseout(up2));
172 score_third_digit(.Clock(clkSelect), .Reset, .increaseIN(up2), .lose, .HEX(fakesignal3
), .increaseout(up3));
173 //temporary signal to get information for hex displays
174 assign HEX0 = ~fakesignal1;
175 assign HEX1 = ~fakesignal2;
176 assign HEX2 = ~fakesignal3;
177
178 endmodule
179
180 /*//Setting the temporary outputs from the center and normal light modules to the
actual LEDs
181 assign LEDR[1] = hold1;
182 assign LEDR[2] = hold2;
183 assign LEDR[3] = hold3;
184 assign LEDR[4] = hold4;
185 assign LEDR[5] = hold5;
186 assign LEDR[6] = hold6;
187 assign LEDR[7] = hold7;
188 assign LEDR[8] = hold8;
189 assign LEDR[9] = hold9;*/
190
191 //Temporary value to send to counters module to display the winner in the 7segment
HEX0 and HEX1
192 //logic [6:0] hold00;
193 //logic [6:0] hold11;
194
195 //Assign the temporary value to HEX0 based on the result from the victory module
196 //assign HEX0 = ~hold00;
197 //assign HEX1 = ~hold11;
198
199 //Turns off all (2-5) 7segment HEX displays
200 /*assign HEX2 = 7'b1111111;
201 assign HEX3 = 7'b1111111;
202 assign HEX4 = 7'b1111111;
203 assign HEX5 = 7'b1111111;*/
204
205
206 //controls display of HEX0 and HEX1 for the scores of the player and computer
207 //counters player_win(.Clock(clkSelect), .Reset, .LR(player1_2nd & ~com1_2nd),
.LRM(hold1), .HEX(hold00), .restart(restart1));
208 //counters com_win(.Clock(clkSelect), .Reset, .LR(~player1_2nd & com1_2nd),
.LRM(hold9), .HEX(hold11), .restart(restart2));
209
210
211 module DE1_SoC_testbench();
212 logic [9:0] LEDR;
213 logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
214
215 logic CLOCK_50; //50MHZ clock
216 logic [9:0] SW;
217 logic [3:0] KEY;
218 logic [35:0] GPIO_1;
219
220 DE1_SoC dut(CLOCK_50, LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, SW, KEY, GPIO_1);
221
222 // Set up a simulated clock to toggle (from low to high or high to low)
223 // every 50 time steps
224 parameter CLOCK_PERIOD=100;
225 initial begin
226     CLOCK_50 <= 0;
227     forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; //toggle the clock indefinitely
228 end
229 initial begin
230
231
232     SW[9] <= 1;
233     SW[9] <= 0;
234     KEY[0] <= 1'b0;
235     KEY[0] <= 1'b1;
236     KEY[0] <= 1'b0;
237     KEY[0] <= 1'b1;
238
239     repeat(2) @ (posedge CLOCK_50); //player wins
240     repeat(2) @ (posedge CLOCK_50);
241     repeat(2) @ (posedge CLOCK_50);
242     repeat(2) @ (posedge CLOCK_50);
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

237 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50);
238 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
239 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50);
240 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
241 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50); //reset after
    player wins
242 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
243 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50);
244 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
245 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50);
246 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
247 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50);
248 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
249 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50);
250 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
251 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
252 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50); //hold test
253 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50);
254 KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50);
255 SW[9] <= 1; repeat(2) @(posedge CLOCK_50); //reset
256 SW[9] <= 0; KEY[0] <= 1'b0; repeat(2) @(posedge CLOCK_50); //moves right
257 SW[9] <= 1; repeat(2) @(posedge CLOCK_50); //reset
258 SW[9] <= 0; KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50); //moves left
259 SW[9] <= 1; repeat(2) @(posedge CLOCK_50); //reset
260 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50); //computer wins
261 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
262 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
263 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
264 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
265 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
266 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
267 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
268 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50);
269 SW[9] <= 1; repeat(2) @(posedge CLOCK_50); //reset
270 KEY[0] <= 1'b1; repeat(2) @(posedge CLOCK_50); //both plerys
    push at the same time
271 $stop; // End of simulation
272 end
273 endmodule
274
275

```

```

1  module centerBird (Clock, Reset, gravity, R, NL, NR, lose, lightOn);
2      input logic Clock, Reset;
3
4      input logic gravity, R, NL, NR, lose;
5
6      // when lightOn is true, the center light should be on.
7      //if lightOn is true, center light is also true
8      output logic lightOn;
9
10     enum {on, off} ps, ns; // on and off states
11
12     // logic for the on and off states for the center light
13     always_comb begin
14         case(ps)
15             on: if((gravity & ~R) | (~gravity & R)) begin
16                 ns = off;
17             end
18             else begin
19                 ns = on;
20             end
21             off: if((gravity & NR & ~R) | (R & NL & ~gravity)) begin
22                 ns = on;
23             end
24             else begin
25                 ns = off;
26             end
27         endcase
28     end
29
30     always_comb begin
31         case (ps)
32             on : lightOn = 1'b1; //true if ON state
33             off: lightOn = 1'b0; //false if OFF state
34         endcase
35     end
36
37     always_ff @(posedge Clock) begin
38         if(Reset) begin // centerlight stays on if on reset
39             ps <= on;
40         end
41         else if(lose) begin
42             ps <= ps;
43         end
44         else begin
45             ps <= ns;
46         end
47     end
48
49 endmodule
50
51 module centerBird_testbench();
52     logic Clock, Reset, gravity, R, NL, NR, lose;
53     logic lightOn;
54
55     centerBird dut (Clock, Reset, gravity, R, NL, NR, lose, lightOn);
56
57     // Set up a simulated clock.
58     parameter CLOCK_PERIOD=100;
59     initial begin
60         Clock <= 0;
61         forever #(CLOCK_PERIOD/2) Clock <= ~Clock; // Forever toggle the clock
62     end
63
64     // Set up the inputs to the design. Each line is a clock cycle.
65     initial begin
66
67         @(posedge Clock);
68         Reset <= 1;
69
70         Clock);
71         Reset <= 0; gravity <= 1'b0; R <= 1'b1; NL <= 1'b0; NR <= 1'b1; lose <= 1'b0; repeat(3
72     ) @(posedge Clock);
73         Reset <= 0; gravity <= 1'b1; R <= 1'b0; NL <= 1'b0; NR <= 1'b1; lose <= 1'b0; repeat(3
74     ) @(posedge Clock);
75         Reset <= 1;
76
77         @(posedge

```



```
71 Clock);
72   Reset <= 0; gravity <= 1'b1; R <= 1'b1; NL <= 1'b0; NR <= 1'b1; lose <= 1'b1; repeat(3
73 )   @(posedge Clock);
74   Reset <= 0; gravity <= 1'b1; R <= 1'b0; NL <= 1'b0; NR <= 1'b1; lose <= 1'b0; repeat(3
75 )   @(posedge Clock);
76   @(posedge Clock);
77   @(posedge Clock);
78   Reset <= 1;
79   @(posedge
80 Clock);
81   Reset <= 0; gravity <= 1'b0; R <= 1'b1; NL <= 1'b1; NR <= 1'b0; lose <= 1'b0; repeat(3
82 )   @(posedge Clock);
83   Reset <= 0; gravity <= 1'b1; R <= 1'b0; NL <= 1'b0; NR <= 1'b1; lose <= 1'b1; repeat(3
84 )   @(posedge Clock);
85   @(posedge Clock);
86   Reset <= 1;
87   @(posedge
88 Clock);
89   Reset <= 0; gravity <= 1'b1; R <= 1'b1; NL <= 1'b0; NR <= 1'b1; lose <= 1'b1; repeat(3
90 )   @(posedge Clock);
91   Reset <= 0; gravity <= 1'b0; R <= 1'b0; NL <= 1'b0; NR <= 1'b1; lose <= 1'b1; repeat(3
92 )   @(posedge Clock);
93   @(posedge Clock);
94   Reset <= 1;
95   @(posedge Clock);
96   $stop; // End the simulation.
97   end
endmodule
```



```

1  module normalBird (Clock, Reset, gravity, R, NL, NR, high, low, lose, lightOn);
2      input logic Clock, Reset;
3
4      //gravity is true when there is no input from key
5      input logic gravity, R, NL, NR, high, low, lose;
6
7      output logic lightOn;
8
9      enum {on, off} ps, ns; // two possible state: on and off states
10
11     //cominational logic for the ON and OFF states
12     always_comb begin
13         case(ps)
14             on: if((~low & gravity & ~R) | (~gravity & R & ~high)) begin //on when not low
and the right key is pressed
15                 ns = off;
16                 end
17                 else if(gravity & ~R & ~low) begin
18                     ns = off;
19                 end
20                 else begin
21                     ns = on;
22                 end
23             off: if((gravity & NR & ~R) | (R & NL & ~gravity)) begin
24                 ns = on;
25                 end
26                 else begin
27                     ns = off;
28                 end
29             endcase
30         end
31
32     always_comb begin
33         case (ps)
34             on : lightOn = 1'b1; //Outputs a light turned on if it's in the on state
35             off: lightOn = 1'b0; //Outputs a light turned off if it's in the on state
36         endcase
37     end
38
39     always_ff @(posedge Clock) begin
40         if(Reset) begin // if reset, normal light turns off
41             ps <= off;
42         end
43         else if(lose) begin
44             ps <= ps;
45         end
46         else begin
47             ps <= ns;
48         end
49     end
50 endmodule
51
52 module normalBird_testbench ();
53     logic Clock, Reset, gravity, R, NL, NR, high, low, lose;
54     logic lightOn;
55
56     normalBird dut (Clock, Reset, gravity, R, NL, NR, high, low, lightOn, lose);
57
58     // Set up a simulated clock.
59     parameter CLOCK_PERIOD=100;
60     initial begin
61         Clock <= 0;
62         forever #(CLOCK_PERIOD/2) Clock <= ~Clock; // Forever toggle the clock
63     end
64
65     // Set up the inputs to the design. Each line is a clock cycle.
66     initial begin
67
68         @(posedge Clock);
69         Reset <= 1;
70
71         @(posedge Clock);
69         Reset <= 1;
72
73         @(posedge Clock);
70         Reset <= 1;

```

```

71         @(posedge Clock);
72         Reset <= 0; gravity <= 1'b0; R <= 1'b1; NL <= 1'b0; NR <= 1'b1; high <= 1'b0; low <=
1'b1; lose <= 1'b0; repeat(3) @(posedge Clock);
73         Reset <= 0; gravity <= 1'b1; R <= 1'b0; NL <= 1'b0; NR <= 1'b1; high <= 1'b1; low <=
1'b0; lose <= 1'b0; repeat(3) @(posedge Clock);
74         Reset <= 0; gravity <= 1'b0; R <= 1'b1; NL <= 1'b1; NR <= 1'b0; high <= 1'b1; low <=
1'b0; lose <= 1'b1; repeat(3) @(posedge Clock);
75         gravity <= 1'b0; R <= 1'b1; NL <= 1'b0; NR <= 1'b0; high <= 1'b0; low <=
1'b1; lose <= 1'b0; repeat(3) @(posedge Clock);
76         Reset <= 1;

77         @(posedge Clock);
78         Reset <= 0; gravity <= 1'b0; R <= 1'b1; NL <= 1'b1; NR <= 1'b1; high <= 1'b1; low <=
1'b0; lose <= 1'b1; repeat(3) @(posedge Clock);
79         gravity <= 1'b1; R <= 1'b0; NL <= 1'b1; NR <= 1'b0; high <= 1'b0; low <=
1'b1; lose <= 1'b1; repeat(3) @(posedge Clock);
80         gravity <= 1'b0; R <= 1'b1; NL <= 1'b0; NR <= 1'b1; high <= 1'b0; low <=
1'b1; lose <= 1'b1; repeat(3) @(posedge Clock);
81         gravity <= 1'b1; R <= 1'b0; NL <= 1'b0; NR <= 1'b1; high <= 1'b1; low <=
1'b0; lose <= 1'b0; repeat(3) @(posedge Clock);
82
83         Reset <= 1;
84         Reset <= 0; gravity <= 1'b0; R <= 1'b1; NL <= 1'b1; NR <= 1'b0; high <= 1'b1; low
<= 1'b0; lose <= 1'b0; repeat(3) @(posedge Clock);
85         Reset <= 0; gravity <= 1'b1; R <= 1'b0; NL <= 1'b1; NR <= 1'b0; high <= 1'b1; low
<= 1'b0; lose <= 1'b1; repeat(4) @(posedge Clock);
86
87         @(posedge Clock);
88         $stop;// End of simulation
endmodule

```

```

1  module playerInput (Clock, Reset, final_in, final_out);
2  //The output for the player's input
3  output logic final_out;
4  //The inputs for the clock, the reset and the player input
5  input logic Clock;
6  input logic Reset;
7  input logic final_in;
8
9  enum {np, p} ps, ns; //state variables for pressed(p) and not pressed(np)
10
11  always_comb begin
12      case(ps)
13          np: if(final_in) begin //If it's not pressed, assign new state
14                  final_out = 1'b1;
15                  ns = p;
16              end else begin
17                  final_out = 1'b0; //Else it will remain not pressed
18                  ns = np; //Remains in the current state
19              end
20
21          p: if(final_in) begin //If it's pressed, output is 0 to prevent holding
22                  final_out = 1'b0;
23                  ns = p; //Remains in the same state if it's held
24              end else begin
25                  final_out = 1'b0;
26                  ns = np; //Goes to not pressed state if player lets go of key
27              end
28      endcase
29  end
30
31  always_ff @(posedge Clock) begin
32      if(Reset) begin
33          ps <= np; //Start with not pressed when reset starts the game
34      end else begin
35          ps <= ns; //New state becomes present state
36      end
37  end
38  endmodule
39
40  module playerInput_testbench ();
41      logic final_out;
42
43      logic Clock;
44      logic Reset;
45      logic final_in;
46
47      playerInput dut (Clock, Reset, final_in, final_out);
48
49      // Set up a simulated clock to toggle (from low to high or high to low)
50      // every 50 time steps
51      parameter CLOCK_PERIOD=100;
52      initial begin
53          Clock <= 0;
54          forever #(CLOCK_PERIOD/2) Clock <= ~Clock; //toggle the clock indefinitely
55      end
56      //Sets up possible combinations for simulations
57      //Each line represents a clock cycle.
58      // Simulation sends the state machine into both possible states
59      initial begin
60          Reset <= 1;
61          Reset <= 1; final_in <= 16'b1; repeat(4)
62          Reset <= 1; final_in <= 16'b0; repeat(4)
63          Reset <= 0; final_in <= 16'b0; repeat(4)
64          presses key final_in <= 16'b1; repeat(4)
65          final_in <= 16'b0; repeat(4)
66          final_in <= 16'b0; repeat(4)
67          final_in <= 16'b1; repeat(4)
68          final_in <= 16'b0; repeat(4)
69          final_in <= 16'b1; repeat(4)
70          final_in <= 16'b1; repeat(4)
71          final_in <= 16'b1; repeat(4)
72          final_in <= 16'b1; repeat(4)
73          final_in <= 16'b1; repeat(4)
74          $stop; //end the simulation
75      end

```

76
77

endmodule

```

1  module metaStable (Clock, Reset, player_in, player_out);
2      //The output for the player's input
3      output logic player_out;
4      //The inputs for the clock, the reset and the player input
5      input logic Clock;
6      input logic Reset;
7      input logic player_in;
8
9      //Assigns the player input as the output based on the clock, unless it's reset it
10     assigns 0
11     always_ff @(posedge Clock) begin
12         if(Reset) begin
13             player_out <= 1'b0; //Outputs 0 if it's in reset
14         end else begin
15             player_out <= player_in; //Outputs the player's input when it's time
16         end
17     end
18 endmodule
19
20 module metaStable_testbench ();
21     logic player_out;
22
23     logic Clock;
24     logic Reset;
25     logic player_in;
26
27     metaStable dut(Clock, Reset, player_in, player_out);
28
29     // Set up a simulated clock to toggle (from low to high or high to low)
30     // every 50 time steps
31     parameter CLOCK_PERIOD=100;
32     initial begin
33         Clock <= 0;
34         forever #(CLOCK_PERIOD/2) Clock <= ~Clock; //toggle the clock indefinitely
35     end
36
37     //Sets up possible combinations for simulations
38     //Each line represents a clock cycle.
39     //Simulation sends the state machine into both possible states
40     initial begin
41
42         Repeat(1) @ (posedge Clock);
43         Repeat(1) @ (posedge Clock); //reset on
44         Repeat(4) @ (posedge Clock); //Test different
45         player inputs
46         Reset <= 0;
47         player_in <= 1'b0;
48         player_in <= 1'b1;
49         player_in <= 1'b0;
50         player_in <= 1'b1;
51         Repeat(4) @ (posedge Clock);
52         Repeat(4) @ (posedge Clock);
53         Repeat(4) @ (posedge Clock);
54         Repeat(4) @ (posedge Clock); //Test player
55         inputs with reset
56         player_in <= 1'b0;
57         player_in <= 1'b1;
58         player_in <= 1'b0;
59         Repeat(4) @ (posedge Clock);
60         Repeat(4) @ (posedge Clock);
61         Repeat(4) @ (posedge Clock);
62         Repeat(4) @ (posedge Clock);
63         $stop; // End the simulation.
64     end
65 endmodule
66

```

```
1  //This module divides the on-board FPGA clock at 50Mhz to
2  //divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz,
3  //[24] = 1.5Hz, [25] = 0.75Hz, ...and so on.
4  module clock_divider (clock, reset, divided_clocks);
5      input logic reset, clock;
6      output logic [31:0] divided_clocks = 0;
7      always_ff @(posedge clock) begin
8          divided_clocks <= divided_clocks + 1;
9      end
10 endmodule
```

```

1  // A driver for the 16x16x2 LED display expansion board.
2  // Read below for an overview of the ports.
3  // IMPORTANT: You do not need to necessarily modify this file. But if you do, be sure you
   know what you are doing.
4
5  // FREQDIV: (Parameter) Sets the scanning speed (how often the display cycles through rows)
6  //           The CLK input divided by 2^A(FREQDIV) is the interval at which the driver
   switches rows.
7  // GPIO_1: (Output) The 36-pin GPIO1 header, as on the DE1-SoC board.
8  // RedPixels: (Input) A 16x16 array of logic items corresponding to the red pixels you'd
   like to have lit on the display.
9  // GrnPixels: (Input) A 16x16 array of logic items corresponding to the green pixels you'd
   like to have lit on the display.
10 // EnableCount: (Input) Whether to continue moving through the rows.
11 // CLK: (Input) The system clock.
12 // RST: (Input) Resets the display driver. Required during startup before use.
13 module LEDDriver #(parameter FREQDIV = 0) (GPIO_1, RedPixels, GrnPixels, EnableCount, CLK,
   RST);
14     output logic [35:0] GPIO_1;
15     input logic [15:0][15:0] RedPixels ;
16     input logic [15:0][15:0] GrnPixels ;
17     input logic EnableCount, CLK, RST;
18
19     reg [(FREQDIV + 3):0] Counter;
20     logic [3:0] RowSelect;
21     assign RowSelect = Counter[(FREQDIV + 3):FREQDIV];
22
23     always_ff @(posedge CLK)
24     begin
25         if(RST) Counter <= 'b0;
26         if(EnableCount) Counter <= Counter + 1'b1;
27     end
28
29     assign GPIO_1[35:32] = RowSelect;
30     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] };
31     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] };
32 endmodule
33
34 module LEDDriver_Test();
35     logic CLK, RST, EnableCount;
36     logic [15:0][15:0] RedPixels;
37     logic [15:0][15:0] GrnPixels;
38     logic [35:0] GPIO_1;
39
40     LEDDriver #(.FREQDIV(2)) Driver(.GPIO_1, .RedPixels, .GrnPixels, .EnableCount, .CLK, .
   RST);
41
42     initial
43     begin
44         CLK <= 1'b0;
45         forever #50 CLK <= ~CLK;
46     end
47
48     initial
49     begin
50         EnableCount <= 1'b0;
51         RedPixels <= '{default:0};
52         GrnPixels <= '{default:0};
53         @(posedge CLK);
54
55         RST <= 1; @(posedge CLK);
56         RST <= 0; @(posedge CLK);
57         @(posedge CLK); @(posedge CLK); @(posedge CLK);
58
59         GrnPixels[1][1] <= 1'b1; @(posedge CLK);
60         EnableCount <= 1'b1; @(posedge CLK); #1000;
61         RedPixels[2][2] <= 1'b1;
62         RedPixels[2][3] <= 1'b1;

```

```

63     GrnPixels[2][3] <= 1'b1; @(posedge CLK); #1000;
64     EnableCount <= 1'b0; @(posedge CLK); #1000;
65     GrnPixels[1][1] <= 1'b0; @(posedge CLK);
66     $stop;
67
68     end
69 endmodule
70
71 module LEDDriver_TestPhysical(CLOCK_50, RST, Speed, GPIO_1);
72     input logic CLOCK_50, RST;
73     input logic [9:0] Speed;
74     output logic [35:0] GPIO_1;
75     logic [15:0][15:0] RedPixels;
76     logic [15:0][15:0] GrnPixels;
77     logic [31:0] Counter;
78     logic EnableCount;
79
80     LEDDriver #(.FREQDIV(15)) Driver (.CLK(CLOCK_50), .RST, .EnableCount, .RedPixels, .
GrnPixels, .GPIO_1);
81
82     //
83     assign RedPixels[00] = '{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
84     assign RedPixels[01] = '{1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1};
85     assign RedPixels[02] = '{1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1};
86     assign RedPixels[03] = '{1,0,1,1,0,0,0,0,0,0,0,0,0,1,1,0};
87     assign RedPixels[04] = '{1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0};
88     assign RedPixels[05] = '{1,0,1,0,1,1,0,0,0,0,1,1,0,1,0,1};
89     assign RedPixels[06] = '{1,0,1,0,1,0,1,1,1,1,0,1,0,1,0,1};
90     assign RedPixels[07] = '{1,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1};
91     assign RedPixels[08] = '{1,0,1,0,1,0,1,1,0,1,0,1,0,1,0,1};
92     assign RedPixels[09] = '{1,0,1,0,1,0,1,1,1,1,0,1,0,1,0,1};
93     assign RedPixels[10] = '{1,0,1,0,1,1,0,0,0,0,1,1,0,1,0,1};
94     assign RedPixels[11] = '{1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1};
95     assign RedPixels[12] = '{1,0,1,1,0,0,0,0,0,0,0,0,1,1,0,1};
96     assign RedPixels[13] = '{1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1};
97     assign RedPixels[14] = '{1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1};
98     assign RedPixels[15] = '{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
99
100    assign GrnPixels[00] = '{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
101    assign GrnPixels[01] = '{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0};
102    assign GrnPixels[02] = '{0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0};
103    assign GrnPixels[03] = '{0,1,0,1,1,1,1,1,1,1,1,1,1,1,0,1,0};
104    assign GrnPixels[04] = '{0,1,0,1,1,0,0,0,0,0,0,0,1,1,0,1,0};
105    assign GrnPixels[05] = '{0,1,0,1,0,1,1,1,1,1,1,1,0,1,0,1,0};
106    assign GrnPixels[06] = '{0,1,0,1,0,1,1,0,0,1,1,0,1,0,1,0,1};
107    assign GrnPixels[07] = '{0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0};
108    assign GrnPixels[08] = '{0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,0,1};
109    assign GrnPixels[09] = '{0,1,0,1,0,1,1,0,0,1,1,0,1,0,1,0,1};
110    assign GrnPixels[10] = '{0,1,0,1,0,1,1,1,1,1,1,1,0,1,0,1,0};
111    assign GrnPixels[11] = '{0,1,0,1,1,0,0,0,0,0,0,0,1,1,0,1,0};
112    assign GrnPixels[12] = '{0,1,0,1,1,1,1,1,1,1,1,1,1,1,0,1,0};
113    assign GrnPixels[13] = '{0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0};
114    assign GrnPixels[14] = '{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0};
115    assign GrnPixels[15] = '{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
116
117    always_ff @(posedge CLOCK_50)
118    begin
119        if(RST) Counter <= 'b0;
120        else
121        begin
122            Counter <= Counter + 1'b1;
123            if(Counter >= Speed)
124            begin
125                EnableCount <= 1'b1;
126                Counter <= 'b0;
127            end
128            else EnableCount <= 1'b0;
129        end
130    end
131 endmodule
132

```



```

1  module shift_pipes(Clock, Reset, newpattern, lose, out);
2      input logic Clock, Reset, lose;
3      input logic [15:0] newpattern;
4      output logic [15:0] out;
5
6      always_ff @(posedge Clock) begin
7          if(Reset) begin
8              out <= 16'b0; //16 is a 16 bit result
9          end
10         else if (lose) begin
11             out <= out;
12         end
13         else begin
14             out <= newpattern; //out becomes a new patter
15         end
16     end
17
18 endmodule
19
20 module shift_pipes_testbench();
21     logic Clock, Reset, lose;
22     logic [15:0] newpattern;
23     logic [15:0] out;
24
25     shift_pipes dut(Clock, Reset, newpattern, lose, out);
26
27     // Set up a simulated clock to toggle (from low to high or high to low)
28     // every 50 time steps
29     parameter CLOCK_PERIOD=100;
30     initial begin
31         Clock <= 0;
32         forever #(CLOCK_PERIOD/2) Clock <= ~Clock; //toggle the clock indefinitely
33     end
34
35     //Sets up possible combinations for simulations
36     //Each line represents a clock cycle.
37     // Simulation sends the state machine into both possible states
38     initial begin
39
40         repeat(1) @(posedge Clock
41     );
42         Reset <= 1;
43         repeat(1) @(posedge Clock
44     ); //reset on
45         Reset <= 0;    newpattern <= 16'b1100111111111111;
46         repeat(4) @(posedge Clock
47     );
48         Reset <= 1;
49         repeat(4) @(posedge Clock
50     );
51         Reset <= 0;    newpattern <= 16'b1111111100111111;
52         repeat(4) @(posedge Clock
53     );
54         newpattern <= 16'b1001111111111111;
55         repeat(4) @(posedge Clock);
56         newpattern <= 16'b1111111111110011;
57         repeat(4) @(posedge Clock
58     );
59         newpattern <= 16'b1111100111111111;
60         repeat(4) @(posedge Clock
61     );
62         newpattern <= 16'b1111111100111111;
63         repeat(4) @(posedge Clock
64     );
65         newpattern <= 16'b1110011111111111;
66         repeat(4) @(posedge Clock);
67         newpattern <= 16'b1100111111111111;
68         repeat(4) @(posedge Clock);
69         Repeat(1) @(posedge Clock
70     ); //reset on
71         $stop; // End the simulation.
72     end
73 endmodule

```

```

1      module LFSR2(Clock, Reset, out);
2
3      input logic Clock, Reset;
4      output logic [3:0] out;
5      logic connection;
6
7      assign connection = ~(out[0] ^ out[1]); //XNOR out1 and out0
8
9      always_ff @(posedge Clock) begin
10         if(Reset) begin
11             out <= 4'b0000; //4 bit output
12         end
13         else begin
14             out <= {connection, out[3:1]}; //Dff moves from connections to 2-3 and replaces
Dffs q1 to be
15                                     //the result of connection
16         end
17     end
18 endmodule
19
20
21 module LFSR2_testbench();
22     logic Clock, Reset;
23     logic [3:0] out;
24
25     LFSR2 dut (Clock, Reset, out);
26
27     // Set up a simulated clock to toggle (from low to high or high to low)
28     // every 50 time steps
29     parameter CLOCK_PERIOD=100;
30     initial begin
31         Clock <= 0;
32         forever #(CLOCK_PERIOD/2) Clock <= ~Clock; //toggle the clock indefinitely
33     end
34
35     //Sets up possible combinations for simulations
36     //Each line represents a clock cycle.
37     initial begin
38
39         @ (posedge Clock);
40         Reset <= 1; @ (posedge Clock); // Always reset FSMs at start
41         Reset <= 0; repeat(3) @ (posedge Clock);
42         Reset <= 1; @ (posedge Clock);
43         Reset <= 0; repeat(3) @ (posedge Clock);
44         Reset <= 1; @ (posedge Clock);
45         Reset <= 0; repeat(3) @ (posedge Clock);
46         @ (posedge Clock);
47         @ (posedge Clock);
48         @ (posedge Clock);
49         @ (posedge Clock);
50         @ (posedge Clock);
51         @ (posedge Clock);
52         $stop; // End the simulation.
53     end
54 endmodule

```

```

1  module gameOver(pipe, bird, lose);
2      input logic [15:0] bird, pipe;
3      output logic lose;
4
5      logic hit0, hit1, hit2, hit3, hit4, hit5;
6      logic hit6, hit7, hit8, hit9, hit10, hit11;
7      logic hit12, hit13, hit14, hit15;
8
9      assign hit0 = pipe[0] & bird[0];
10     assign hit1 = pipe[1] & bird[1];
11     assign hit2 = pipe[2] & bird[2];
12     assign hit3 = pipe[3] & bird[3];
13     assign hit4 = pipe[4] & bird[4];
14     assign hit5 = pipe[5] & bird[5];
15     assign hit6 = pipe[6] & bird[6];
16     assign hit7 = pipe[7] & bird[7];
17     assign hit8 = pipe[8] & bird[8];
18     assign hit9 = pipe[9] & bird[9];
19     assign hit10 = pipe[10] & bird[10];
20     assign hit11 = pipe[11] & bird[11];
21     assign hit12 = pipe[12] & bird[12];
22     assign hit13 = pipe[13] & bird[13];
23     assign hit14 = pipe[14] & bird[14];
24     assign hit15 = pipe[15] & bird[15];
25
26     //you losse if any of these logic combinations happen
27     assign lose = hit0 | hit1 | hit2 | hit3 | hit4 | hit5 |
28                 hit6 | hit7 | hit8 | hit9 | hit10 | hit11 |
29                 hit12 | hit13 | hit14 | hit15;
30 endmodule
31
32 module gameOver_testbench ();
33     logic [15:0] bird, pipe;
34     logic lose;
35
36     gameOver dut (bird, pipe, lose);
37
38     initial begin
39         bird = 16'b00000000000010000; pipe = 16'b0011111111111111; #10; //hit
40         bird = 16'b00000000000000001; pipe = 16'b1111111111111100; #10; //pass
41         bird = 16'b00000000100000000; pipe = 16'b1001111111111111; #10; //hit
42         bird = 16'b00000100000000000; pipe = 16'b1111100111111111; #10; //pass
43         bird = 16'b00000000000000010; pipe = 16'b1111001111111111; #10; //hit
44     end
45 endmodule
46

```

```

1  module clock_counter (Clock, Reset, out);
2      input logic Clock, Reset;
3      output logic out;
4
5      //state variables for present states and new states of pipe gaps
6      enum {pattern, gap1, gap2, gap3, gap4} ps, ns;
7
8      // This logic describes all the possible state transitions from ps to ns
9      always_comb begin
10         case (ps)
11             pattern: begin
12                 out = 1'b1;
13                 ns = gap1;
14             end
15             gap1: begin
16                 out = 1'b0;
17                 ns = gap2;
18             end
19             gap2: begin
20                 out = 1'b0;
21                 ns = gap3;
22             end
23             gap3: begin
24                 out = 1'b0;
25                 ns = gap4;
26             end
27             gap4: begin
28                 out = 1'b0;
29                 ns = pattern;
30             end
31         endcase
32     end
33
34     // D Flip Flop implementation (DFFs)
35     always_ff @(posedge Clock) begin
36         if (Reset) begin
37             ps <= pattern;
38         end
39         else begin
40             ps <= ns; //Otherwise, advances to next state in state diagram
41         end
42     end
43 endmodule
44
45 module clock_counter_testbench ();
46     logic Clock, Reset;
47     logic out;
48
49     clock_counter dut(Clock, Reset, out);
50
51     // Set up a simulated clock to toggle (from low to high or high to low)
52     // every 50 time steps
53     parameter CLOCK_PERIOD=100;
54     initial begin
55         Clock <= 0;
56         forever #(CLOCK_PERIOD/2) Clock <= ~Clock; //toggle the clock indefinitely
57     end
58
59     //Sets up possible combinations for simulations
60     //Each line represents a clock cycle.
61     initial begin
62
63         repeat(2) @(posedge Clock);
64         Reset <= 1; repeat(3) @(posedge Clock);
65         Reset <= 0; repeat(3) @(posedge Clock);
66         repeat(3) @(posedge Clock);
67         repeat(3) @(posedge Clock);
68         repeat(3) @(posedge Clock);
69         Reset <= 1; repeat(3) @(posedge Clock);
70         Reset <= 0; repeat(3) @(posedge Clock);
71         repeat(3) @(posedge Clock);
72         repeat(3) @(posedge Clock);
73         repeat(3) @(posedge Clock);
74         $stop; // End the simulation.
75     end
76 endmodule

```

77
78

```
1  module score(Clock, Reset, lose, increaseIN, increaseout, HEX);
2      input logic Clock, Reset;
3      input logic lose, increaseIN;
4
5      output logic [6:0] HEX;
6      output logic increaseout;
7
8      // State variables, resets to 0 after 9
9      enum { zero, one, two, three, four, five, six, seven, eight, nine } ps, ns;
10
11     // Next State logic
12     always_comb begin
13         case (ps)
14             zero: if(increaseIN) begin
15                     HEX = 7'b0000110; //1
16                     increaseout = 1'b0;
17                     ns = one;
18                 end
19                 else begin
20                     HEX = 7'b0111111;
21                     increaseout = 1'b0; //0
22                     ns = zero;
23                 end
24             one: if(increaseIN) begin
25                     HEX = 7'b1011011; //2
26                     increaseout = 1'b0;
27                     ns = two;
28                 end
29                 else begin
30                     HEX = 7'b0000110; //1
31                     increaseout = 1'b0;
32                     ns = one;
33                 end
34             two: if(increaseIN) begin
35                     HEX = 7'b1001111; //3
36                     increaseout = 1'b0;
37                     ns = three;
38                 end
39                 else begin
40                     HEX = 7'b1011011; //2
41                     increaseout = 1'b0;
42                     ns = two;
43                 end
44             three: if(increaseIN) begin
45                     HEX = 7'b1100110; //4
46                     increaseout = 1'b0;
47                     ns = four;
48                 end
49                 else begin
50                     HEX = 7'b1001111; //3
51                     increaseout = 1'b0;
52                     ns = three;
53                 end
54             four: if(increaseIN) begin
55                     HEX = 7'b1101101; //5
56                     increaseout = 1'b0;
57                     ns = five;
58                 end
59                 else begin
60                     HEX = 7'b1100110; //4
61                     increaseout = 1'b0;
62                     ns = four;
63                 end
64             five: if(increaseIN) begin
65                     HEX = 7'b1111101; //6
66                     increaseout = 1'b0;
67                     ns = six;
68                 end
69                 else begin
70                     HEX = 7'b1101101; //5
71                     increaseout = 1'b0;
72                     ns = five;
73                 end
74             six: if(increaseIN) begin
75                     HEX = 7'b0000111; //7
76                     increaseout = 1'b0;
```

```

77         ns = seven;
78     end
79     else begin
80         HEX = 7'b1111101; //6
81         increaseout = 1'b0;
82         ns = six;
83     end
84     seven: if(increaseIN) begin
85         HEX = 7'b1111111; //8
86         increaseout = 1'b0;
87         ns = eight;
88     end
89     else begin
90         HEX = 7'b0000111; //7
91         increaseout = 1'b0;
92         ns = seven;
93     end
94     eight: if(increaseIN) begin
95         HEX = 7'b1101111; //9
96         increaseout = 1'b0;
97         ns = nine;
98     end
99     else begin
100        HEX = 7'b1111111; //8
101        increaseout = 1'b0;
102        ns = eight;
103    end
104    nine: if(increaseIN) begin
105        HEX = 7'b0111111; //0
106        increaseout = 1'b1;
107        ns = zero;
108    end
109    else begin
110        HEX = 7'b1101111; //9
111        increaseout = 1'b0;
112        ns = nine;
113    end
114 endcase
115 end
116
117 always_ff @(posedge Clock) begin
118     if(Reset) begin
119         ps <= zero;
120     end
121     else if(lose) begin
122         ps <= ps;
123     end
124     else begin
125         ps <= ns;
126     end
127 end
128 endmodule
129
130 module score_testbench();
131     logic Clock, Reset;
132     logic lose, increaseIN;
133
134     logic [6:0] HEX;
135     logic increaseout;
136
137
138     score dut(.Clock, .Reset, .lose, .increaseIN, .increaseout, .HEX);
139
140     // Set up a simulated clock to toggle (from low to high or high to low)
141     // every 50 time steps
142     parameter CLOCK_PERIOD=100;
143     initial begin
144         Clock <= 0;
145         forever #(CLOCK_PERIOD/2) Clock <= ~Clock; // Forever toggle the clock
146     end
147
148     //Sets up possible combinations for simulations
149     //Each line represents a clock cycle.
150     initial begin
151

```

@ (posedge

Clock);

```
152     Reset <= 1;                                     @(posedge
153     Clock); Reset <= 0; increaseIN <= 1'b0; lose <= 1'b0; repeat(3)  @(posedge
154     Clock); Reset <= 0; increaseIN <= 1'b1; lose <= 1'b0; repeat(3)  @(posedge
155     Clock); Reset <= 1;                                     @(posedge
156     Clock); Reset <= 0; increaseIN <= 1'b0; lose <= 1'b0; repeat(3)  @(posedge
157     Clock); Reset <= 0; increaseIN <= 1'b0; lose <= 1'b1; repeat(3)  @(posedge
158     Clock); Reset <= 1;                                     @(posedge
159     Clock); Reset <= 0; increaseIN <= 1'b0; lose <= 1'b1; repeat(3)  @(posedge
160     Clock); Reset <= 0; increaseIN <= 1'b1; lose <= 1'b0; repeat(3)  @(posedge
161     Clock); Reset <= 1;                                     @(posedge
162     Clock); Reset <= 0;                                     @(posedge
163     Clock); Reset <= 0; increaseIN <= 1'b1; lose <= 1'b1; repeat(3)  @(posedge
164     Clock);                                               @(posedge
165     $stop; // End the simulation.
166     end
167 endmodule
168
169
```



```
1  module pipes(LFSR2_out, pattern, generate_clock);
2      input logic [3:0] LFSR2_out;
3      output logic [15:0] pattern;
4      input logic generate_clock;
5
6      always_comb begin
7          case (LFSR2_out)
8              4'b0000:
9                  if(generate_clock) begin
10                     pattern = 16'b1111110011111111; end // 0
11                 else begin
12                     pattern = 16'b0;
13                 end
14
15              4'b0001:
16                  if(generate_clock) begin
17                     pattern = 16'b1001111111111111; end // 1
18                 else begin
19                     pattern = 16'b0;
20                 end
21
22              4'b0010:
23                  if(generate_clock) begin
24                     pattern = 16'b1100111111111111; end // 2
25                 else begin
26                     pattern = 16'b0;
27                 end
28
29              4'b0011:
30                  if(generate_clock) begin
31                     pattern = 16'b1110011111111111; end // 3
32                 else begin
33                     pattern = 16'b0;
34                 end
35
36              4'b0100:
37                  if(generate_clock) begin
38                     pattern = 16'b1111001111111111; end // 4
39                 else begin
40                     pattern = 16'b0;
41                 end
42
43              4'b0101:
44                  if(generate_clock) begin
45                     pattern = 16'b1111100111111111; end // 5
46                 else begin
47                     pattern = 16'b0;
48                 end
49
50              4'b0110:
51                  if(generate_clock) begin
52                     pattern = 16'b1111110011111111; end // 6
53                 else begin
54                     pattern = 16'b0;
55                 end
56
57              4'b0111:
58                  if(generate_clock) begin
59                     pattern = 16'b1111111001111111; end // 7
60                 else begin
61                     pattern = 16'b0;
62                 end
63
64              4'b1000:
65                  if(generate_clock) begin
66                     pattern = 16'b1111111100111111; end // 8
67                 else begin
68                     pattern = 16'b0;
69                 end
70
71              4'b1001:
72                  if(generate_clock) begin
73                     pattern = 16'b1111111110011111; end // 9
74                 else begin
75                     pattern = 16'b0;
76                 end
```

```

77
78     4'b1010:
79     if(generate_clock) begin
80         pattern = 16'b1111111111001111 ; end// 10
81     else begin
82         pattern = 16'b0;
83     end
84
85     4'b1011:
86     if(generate_clock) begin
87         pattern = 16'b111111111100111 ; end// 11
88     else begin
89         pattern = 16'b0;
90     end
91
92     4'b1100:
93     if(generate_clock) begin
94         pattern = 16'b11111111110011 ; end// 12
95     else begin
96         pattern = 16'b0;
97     end
98
99     4'b1101:
100    if(generate_clock) begin
101        pattern = 16'b1111111111001 ; end // 13
102    else begin
103        pattern = 16'b0;
104    end
105
106    4'b1110:
107    if(generate_clock) begin
108        pattern = 16'b111111111111100 ; end// 14
109    else begin
110        pattern = 16'b0;
111    end
112
113    default: begin
114        pattern = 16'b0000000000000000 ; // default
115    end
116 endcase
117 end
118 endmodule
119
120 module pipes_testbench();
121     logic [3:0] LFSR2_out;
122     logic [15:0] pattern;
123     logic generate_clock;
124
125     pipes dut(LFSR2_out ,pattern);
126
127     integer i;
128     integer j;
129     initial begin
130         generate_clock = 1'b1;
131         for(i = 0; i < 16; i++)
132             begin LFSR2_out = i; #10;
133         end
134
135         generate_clock = 1'b0;
136         for(j = 0; j < 16; j++)
137             begin LFSR2_out = j; #10;
138         end
139     end
140 end
141 endmodule
142
143
144

```

Analysis & Synthesis Resource Utilization by Entity																				
cdl@ethz.ch																				
Compilation Hierarchy Node			Combinational ALUTs		Dedicated Logic Registers		Block Memory Bits		DSP Blocks		Pins		Virtual Pins		Full Hierarchy Name		Entity Name		Library Name	
1	✓	[DE1_Soc]	211 (1)		325 (0)	0		0		103	0					[DE1_Soc]	DE1_Soc	work		
1		[LEDDriverDriver]	100 (100)		1 (0)	0		0		0	0					[DE1_Soc][LEDDriverDriver]	LEDDriver	work		
2		[LF502LF502]	1 (1)		4 (4)	0		0		0	0					[DE1_Soc][LF502LF502]	LF502	work		
3		[centerled1seven]	2 (2)		1 (1)	0		0		0	0					[DE1_Soc][centerled1seven]	centerled1	work		
4		[clock_countercount]	4 (4)		3 (0)	0		0		0	0					[DE1_Soc][clock_countercount]	clock_counter	work		
5		[clock_dividercdv]	26 (26)		26 (26)	0		0		0	0					[DE1_Soc][clock_dividercdv]	clock_divider	work		
6		[gameOverendgame]	8 (8)		0 (0)	0		0		0	0					[DE1_Soc][gameOverendgame]	gameOver	work		
7		[metastablemet1]	0 (0)		1 (1)	0		0		0	0					[DE1_Soc][metastablemet1]	metastable	work		
8		[metastablemet2]	0 (0)		1 (1)	0		0		0	0					[DE1_Soc][metastablemet2]	metastable	work		
9		[normalisedeight]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedeight]	normalised	work		
10		[normalisedeleven]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedeleven]	normalised	work		
11		[normalisedthreen]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedthreen]	normalised	work		
12		[normalisedfive]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedfive]	normalised	work		
13		[normalisedfourteen]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedfourteen]	normalised	work		
14		[normalisedfour]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedfour]	normalised	work		
15		[normalisednine]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisednine]	normalised	work		
16		[normalisedone]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedone]	normalised	work		
17		[normalisedsix]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedsix]	normalised	work		
18		[normalisedten]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedten]	normalised	work		
19		[normalisedthirteen]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedthirteen]	normalised	work		
20		[normalisedthree]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedthree]	normalised	work		
21		[normalisedtwelve]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedtwelve]	normalised	work		
22		[normalisedtwo]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedtwo]	normalised	work		
23		[normalisedzero]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][normalisedzero]	normalised	work		
24		[pipespipe]	6 (6)		0 (0)	0		0		0	0					[DE1_Soc][pipespipe]	pipes	work		
25		[playernputplayer_1]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][playernputplayer_1]	playernput	work		
26		[playernputplayer_2]	1 (1)		1 (1)	0		0		0	0					[DE1_Soc][playernputplayer_2]	playernput	work		
27		[scorefirst_digit]	12 (12)		4 (4)	0		0		0	0					[DE1_Soc][scorefirst_digit]	score	work		
28		[scoresecond_digit]	12 (12)		4 (4)	0		0		0	0					[DE1_Soc][scoresecond_digit]	score	work		
29		[scorethird_digit]	11 (11)		4 (4)	0		0		0	0					[DE1_Soc][scorethird_digit]	score	work		
30		[xhifp_pipesnew0]	0 (0)		16 (16)	0		0		0	0					[DE1_Soc][xhifp_pipesnew0]	xhifp_pipes	work		
31		[xhifp_pipesnew10]	0 (0)		16 (16)	0		0		0	0					[DE1_Soc][xhifp_pipesnew10]	xhifp_pipes	work		
32		[xhifp_pipesnew11]	0 (0)		16 (16)	0		0		0	0					[DE1_Soc][xhifp_pipesnew11]	xhifp_pipes	work		
33		[xhifp_pipesnew12]	0 (0)		16 (16)	0		0		0	0					[DE1_Soc][xhifp_pipesnew12]	xhifp_pipes	work		

Analysis & Synthesis Resource Utilization by Entity											
Compilation Hierarchy Node		Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name	Entity Name	Library Name	
12	[normalfirst]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirst	normalfirst	work	
13	[normalfirstfourteen]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstfourteen	normalfirst	work	
14	[normalfirstfour]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstfour	normalfirst	work	
15	[normalfirstnine]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstnine	normalfirst	work	
16	[normalfirstone]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstone	normalfirst	work	
17	[normalfirstsix]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstsix	normalfirst	work	
18	[normalfirstten]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstten	normalfirst	work	
19	[normalfirstthirteen]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstthirteen	normalfirst	work	
20	[normalfirstthree]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstthree	normalfirst	work	
21	[normalfirsttwelve]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirsttwelve	normalfirst	work	
22	[normalfirsttwo]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirsttwo	normalfirst	work	
23	[normalfirstzero]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]normalfirstzero	normalfirst	work	
24	[playerinput_player_1]	648	0 (0)	0	0	0	0	[DE1_Soc]playerinput_player_1	playerinput	work	
25	[playerinput_player_1]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]playerinput_player_1	playerinput	work	
26	[playerinput_player_2]	1 (1)	1 (1)	0	0	0	0	[DE1_Soc]playerinput_player_2	playerinput	work	
27	[scorefirst_digit]	12 (12)	4 (4)	0	0	0	0	[DE1_Soc]scorefirst_digit	score	work	
28	[scoresecond_digit]	12 (12)	4 (4)	0	0	0	0	[DE1_Soc]scoresecond_digit	score	work	
29	[scorethird_digit]	11 (11)	4 (4)	0	0	0	0	[DE1_Soc]scorethird_digit	score	work	
30	[shift_pipes_row0]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row0	shift_pipes	work	
31	[shift_pipes_row10]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row10	shift_pipes	work	
32	[shift_pipes_row11]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row11	shift_pipes	work	
33	[shift_pipes_row12]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row12	shift_pipes	work	
34	[shift_pipes_row13]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row13	shift_pipes	work	
35	[shift_pipes_row14]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row14	shift_pipes	work	
36	[shift_pipes_row15]	10 (10)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row15	shift_pipes	work	
37	[shift_pipes_row1]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row1	shift_pipes	work	
38	[shift_pipes_row2]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row2	shift_pipes	work	
39	[shift_pipes_row3]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row3	shift_pipes	work	
40	[shift_pipes_row4]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row4	shift_pipes	work	
41	[shift_pipes_row5]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row5	shift_pipes	work	
42	[shift_pipes_row6]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row6	shift_pipes	work	
43	[shift_pipes_row7]	1 (1)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row7	shift_pipes	work	
44	[shift_pipes_row8]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row8	shift_pipes	work	
45	[shift_pipes_row9]	0 (0)	16 (16)	0	0	0	0	[DE1_Soc]shift_pipes_row9	shift_pipes	work	