University of Manitoba

University of Manitoba

Price Faculty of Engineering

ECE 2220: Digital Logic Systems

Term Project Report

Whac-A-Mole

Authors:

Ohm Patel (7928827)

Hana Dunlop (7907031)

Manal Shahab (7933204)

Aida Mesgar Zadeh (7933484)

Instructor:

Douglas Thomson

Date of Submission:

December 6, 2022

1 Abstract

The purpose of this project was to use the material taught in the ECE 2220 - Digital Logic Systems course and create a project using Verilog that has a purpose and functions to achieve certain tasks on the DE10 board. Our team decided to create an original idea instead of the proposed examples in the class. Inspired by the arcade game Whac-A-Mole, we decided to mimic the game onto the FPGA. We used 10 LEDs to indicate where the mole was, and 10 slide switches to wack the mole by toggling the LED's corresponding switch. Each time the mole was whacked, the score increased by 1, to a maximum score of 32 and displayed on the seven segment displays. The user was also given the ability to set a timer of 30, 20 or 10 seconds prior to starting the game through push-buttons and was displayed on the seven segment displays. One push-button was dedicated to starting and restarting the game. Only when both the timer button and the reset button was the game playable. The largest challenge was creating a pseudo-random sequence that the moles appeared in. Verilog does not have any function that could accomplish this on the FPGA. Instead, we manually assigned the LED that the mole will appear in for all 32 score points. Although the programmer would be aware of the sequence that the moles would appear in, the used would not have prior knowledge of the sequence and would have the experience of being random. Thus, we believed that we were successful in implementing the Whac-A-Mole game onto the DE10 standard FPGA board. An improvement would be to add starting and finishing messages on the seven segment displays to indicate when the game has begun and ended.

Contents

List of Tables

List of Figures

2 Team Members and Contributions

Ohm Patel:

- Idea of project and functions
- Report write up
- sub_7segDisplay (sub module)
- sub_bin2bcd (sub module)
- Whac-A-Mole (top module)

Manal Shahab:

- Idea of project and functions
- Planning of project
- sub_timerCount (sub module)

Aida Mesgar Zadeh:

- Idea of project and functions
- Challenges and improvements
- sub_scoreCount (sub module)

Hana Dunlop:

- Idea of project and functions
- Code debugging
- sub_timerClock (sub module)

3 Introduction

For our term project for ECE 2220, we decided to implement a Whac-A-Mole game using Verilog and a FPGA. The code for simulating the game was written using Verilog and demonstrated on the DE10-Standard FPGA-SoC. To provide a user friendly interface, the game utilized 4 push-buttons, 4 seven segment displays, 10 slide switches and 10 LEDs on the FPGA to perform all of the functions of the game.

The idea of mimicking a Whac-A-Mole game on the FPGA depends on the LEDs and slide switches. The board would randomly generate a mole on one of the 10 LEDs. This would be indicated by the LED lighting up. The user would have to toggle the corresponding slide switch to whack the mole. Upon a successful hit, the score would be increase by 1 point on the seven segment displays up to a maximum score of 32.

Additionally, the board would display a timer of 30, 20 or 10 seconds selected by the player before the game is started. The objective is to hit as many moles as possible, to a maximum of 32, within the timer. Successfully hitting all 32 moles with the time would turn all LEDs off indicating that the game is over. A single push button is responsible for starting and resetting the game while the other three push buttons set the timer.

4 Components and Parameters

To implement user friendly controls, the game relies on multiple components. The following table displays the component, its pin name on the FPGA, and its function.

Table 1: A table showing all functional components used on the DE10 board.

	Component	**Pin name**	**Function**
1	10 slide switches	SW[0]	Toggle switches to hit a mole
		SW[1]	
		SW[2]	
		SW[3]	
		SW[4]	
		SW[5]	
		SW[6]	
		SW[7]	
		SW[8]	
		SW[9]	
2	10 LEDs	LEDR[0]	Randomly display moles on the LEDs
		LEDR[1]	
		LEDR[2]	
		LEDR[3]	
		LEDR[4]	
		LEDR[5]	
		LEDR[6]	
		LEDR[7]	
		LEDR[8]	
		LEDR[9]	
3	4 seven-segment displays	HEX0[6:0]	Display ones digit for score
		HEX1[6:0]	Display tens digit for score
		HEX4[6:0]	Display ones digit for timer
		HEX5[6:0]	Display tens digit for timer
4	4 push-buttons	KEY[0]	Start/Restart the game
		KEY[1]	Set 10 second timer
		KEY[2]	Set 20 second timer
		KEY[3]	Set 30 second timer

5 Functionality of the game

Whac-A-Mole is a common arcade game that involves hitting randomly appearing moles that surface from holes using a single hammer within a certain time period. We wanted to mimic the rules of this game into the FPGA.

The idea of the game on the FPGA is that one of the ten LEDs will randomly light up. To hit the mole, the player must toggle the corresponding slide switch. For instance if LEDR[4] is on then SW[4] must be toggled. Upon a successful hit, the score increases by one, the current LED turns off, and another LED is randomly light up. If the switch is already toggled in the on position from a previous hit, toggling the switch off will count as a hit. This would prevent the player from having to reset the switch to the off state each time.

The first rule of Whac-A-Mole is the timer that gives the time frame to hit all the moles within. We added a timer to display on the left most seven-segment displays. To also implement a control over the difficulty of the game, we implemented three timers to increase or decrease the speed that one must play. Initially the timer is blank and displays 00. A 30, 20 and 10 second timer can be set be pressing push buttons 3, 2 and 1 respectively. A different timer can be set at any moment by pressing its push button.

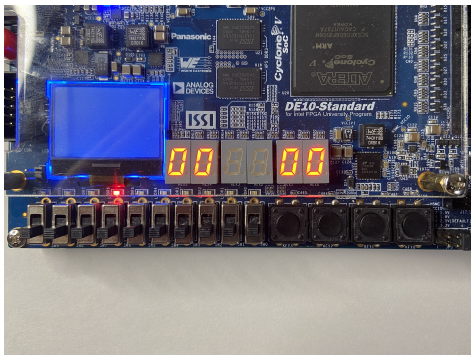

Figure 1: Initial state of DE10 booard

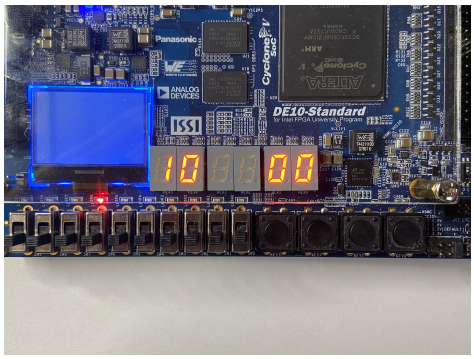

Figure 2: 10 second timer set on board

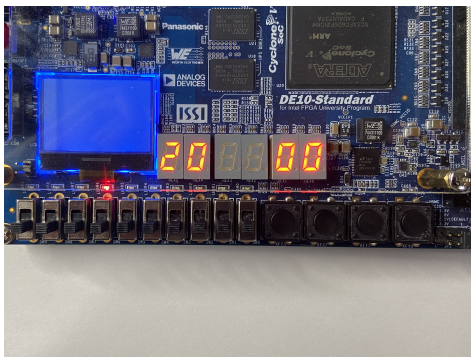

Figure 3: 20 second timer set on board

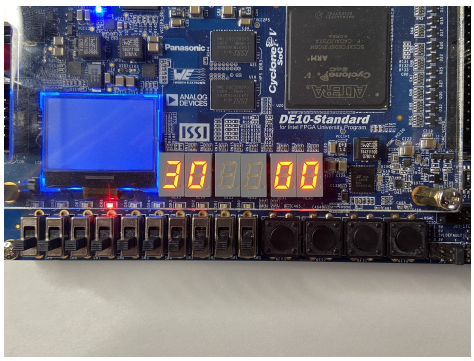

Figure 4: 30 second timer set on board

Another rule for Whac-A-Mole is that there is a score of how many moles you hit. To simplify the scoring system, the game has a maximum score of 32. The right most seven-segment displays the current score during the game. Initially the timer is blank and displays 00. If a mole is hit by toggling the switch, the score increase by one point to a maximum of 32 points. Upon a successful game of hitting all 32 moles, the LEDs would turn off to prevent the game from continuing.

The final rule for this Whac-A-Mole game is that there is only one hammer. To have the restriction of one hammer on the FPGA, we wanted to force the player into only being able to use one hand to toggle the switches. The first pusbutton (KEY[0]) is the restart and start key. Without pressing and holding the button, the score will not increment. Additionally, without pressing and holding the 30, 20 or 10 second timer button, the timer will not count down. Thus, the player must press and hold the reset button and the timer button at the same time to play the game. This would effectively occupy one hand and leave the other hand to play the game with. If the player releases either of the push buttons during the game, the score would reset and the timer would be paused. Once the game is over, all switches have to be toggled to the off position to play again.

After successfully implementing these rules into a simplified version on the FPGA, we believed that we were able to provide a very similar and accurate experience of playing Whac-A-Mole to the player. To view a demonstration of the gameplay and the functionality of the game, refer to the YouTube link in the appendices.

6 Modules

The game on the FPGA was implemented through multiple sub modules and then called in the top module. The game has inputs of the 10 switches, the reset push button, the three timer push buttons, and the default 50 MHz clock on the DE10 board. The outputs are then the 10 LEDs and the 4 seven segments displays. Below is a block diagram of the modules and sum modules used to create the game.

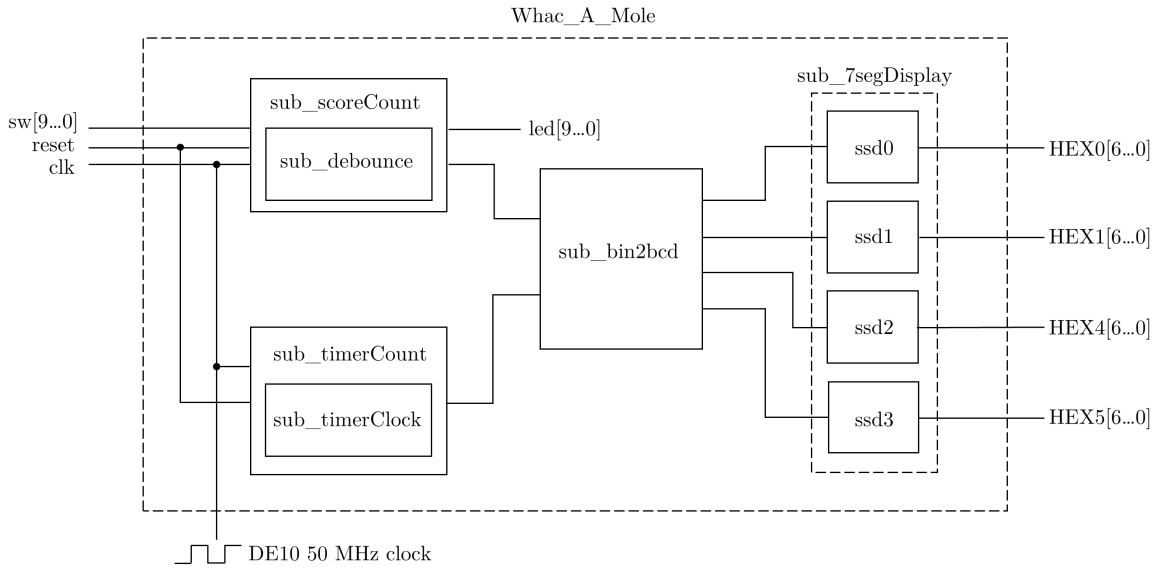

Figure 5: Block diagram of modules used to drive the game on the FPGA.

To control the timer, there was a sub_timerCount module created that called another sub module. The score keeping and the generation of moles was done through the sub_scoreCount module. These two independent modules provided outputs in binary of the timer and score count. Both of these outputs are sent through the sub_bin2bcd module to convert them into displayable outputs. The sub_7segDisplay module provides outputs on the seven segments displays on the FPGA.

The following sections provide more detail into the function of each module and its role in the functionality of the game. To see the Verilog code for each module, please refer to the appendices.

6.1 sub_timerClock

This module was responisble for initializing the clock on the FPGA board. The default 50 MHz clock from the DE10 board was used to give a timer with regular intervals. Increasing the frequency would make the timer count down slower and faster for a lower frequency. This clock was used in the sub_timerCount module.

6.2 sub_timerCount

The function of this module is to set the timer options to each push-button. The module calls the submodule sub_timerClock to initialize the 50MHz frequency of the DE10 board. Within the sub_timerCount module, three if statements were used to set a timer of 30, 20 or 10 seconds depending on which push-button is pressed. Each if statement also includes the conditions for the timer to work. The reset button must be pressed along with the timer button for the timer to count down. If the timer counts down to 0 then the timer remains at 0 until is reset. If the timer is greater than or equal to 1, it is reduced by 1 to countdown each second. This module provides the timer for the game on the FPGA in binary which is later converted into a BCD in the sub_bin2bcd module.

6.3 sub_scoreCount

This module is in charge of keeping a count of the score and to provide the output of the final score. This module takes the 10 switches as inputs and 10 LEDs as outputs. The score count only function when both the reset button and the timer button is pressed down. The score starts of at 00 and increments each time a switch to the corresponding LED is toggled. The maximum score possible is 32 as there are 32 case statements within the module. Each case statement state controls the display on the FPGA at a certain time. At each case, the LEDs are all turned off and then a single LED is turned on. If the corresponding switch is toggled, the score increases by 1. If a different switch is toggled then score does not change. The score is outputted as a binary number which is later converted into a BCD in the sub_bin2bcd module.

6.4 sub_debounce

To minimize the mechanical issue with the push-buttons on the FPGA, this module was used. This push-button debouncing module ensured that the push-button would be read as if it were pressed down and prevent any interruptions when the two contacts

vibrate. This module was called in sub_scoreCount to debounce the reset button and prevent the game from restarting randomly. Note that this code was taken from the UMLearn page [1].

6.5 sub_bin2bcd

This module takes the 8bit binary numbers of the timer and the score, and converts them to BCDs. This essentially splits the number into a binary number for the tens and the ones. For instance, the number 29 in binary is 11101, the BCD conversion would be 0011 for the tens place which is 2 in binary and then 1001 for the ones place which is 9. Verilog automatically reads decimals as binary and vice versa without the need to convert them. The following code is part of the module and was used to split the number into the tens and ones.

```
1 input [7:0] scoreInput, // input of 00011101 or 29
2 output [3:0] scoreOnes,
3 output [3:0] scoreTens
4 assign scoreOnes = scoreInput % 10;
5                                  // 29 % 10 outputs 2
6 assign scoreTens = ((scoreInput-scoreOnes)/10) % 10;
7                                  // ((29-2)/10)%10 outputs 9
```

6.6 sub_7segDisplay

In order to display the output from the sub_bin2bcd module on the seven segment displays, this module was used. It took the input of each digit and assigned them to each seven segment display. The code for this module was taken directly from a previous lab as it required no changes.

6.7 Whac_A_Mole

This was the top module that called each sub module onto the board. The entire game took inputs of the clock, 10 switches, 4 push-buttons and gave outputs to the 10 LEDs and the 4 seven segments displays on the FPGA. Internal wires were also defined to connect the individual modules to each other.

7 Challenges

Our initial plan was to provide options for slower/faster second intervals not determine the difficulty. Instead we kept the same frequency used for the time (50 MHz) and increased the number of seconds or cycles.

Additionally, there were issues with the hardware itself which would not allow for the entirety of the game to be played out. This was due to the mechanics of the push buttons and the slide switches, which made the board's components sensitive to any minor movements or touches to the board. For example, the push buttons do not fully remain in contact with the metal beneath the button when pressed, which causes the board to not register the button being "on". This would end the game prematurely. To reduce this, we added a debouncing module (sub_debounce) which prevented this issue from happening frequently.

Another issue that was faced during the making of the program was implementing the randomness of mole pop-ups, which in terms of the board would be randomness at which the LEDs would turn on. In Verilog, there is a random function ($random) that can output a pseudo-random 32bit signed integer. However, this function can not be synthesizable on the FPGA. The issue arises when the random outcomes have to be assigned to an LED pin out. Since the pin-out assignments are done manually, the game cannot have a completely randomized set of LEDs. To accommodate this, the sub_scoreCount module in the code is assigned to one of the 10 LEDs thirty-two times, (switching to a different LED on the board each time the mole is hit or the correct slide switch has been moved). The programmer would know what order the LEDs would activate but since the user will not have any previous knowledge of the order at which the LED lights turn on, it will appear random to the user. This ultimately provides the user with the experience of playing a randomized game.

8 Improvements

The board could potentially have start and end messages displayed on the SSD to give the game an official start and end, and to provide the player with a sense of when the game begins and ends. This could be implemented by displaying the words "Start" and "Finish" on the seven segment displays of the DE10 board. For visual purposes, the words displaying on the SSD can be flashing/blinking, particularly for the word "Finish".

Another improvement would be to create a module for mimicking the $random. This could be done by creating a linear feedback shift register (LSFR). This essentially consists of multiple flip-flops connected in shift registers. The output from the shift register taps would become inputs in either an XOR or XNOR gate. The output from this gate is then looped back into the shift register giving a feedback loop. The pattern generated by the flip-flops is pseudo-random allowing us to have outputs as close to random. This could be used for selecting which of the 10 LEDs to light up without the programmer having to hard code the assignments.

9 References

[1] D. Thomson, "debouncer_code," UM Learn, 2022. [Online]. Available: https://universityofmanitoba.desire2learn.com/d2l/le/content/494935/viewContent/3040802/View.[Accessed: 04-Dec-2022].

10 Appendices

10.1 Links to project

The GitHub link includes all the codes used to implement the game onto the FPGA. The code is given also in the following section for quick reference.
https://github.com/ohmpatel4/ECE-2220-Term-Project

The YouTube link provides a demo of the game preformed on the DE10 board and its functionality.
https://youtu.be/2FYRVKA54mc

10.2 Verilog Code

```
/*
Whac_A_Mole.v

COURSE:             ECE 2220 - Digital logic systems - A01
INSTRUCTOR:         Douglas Thomson
ASSIGNMENT:         Term Project
AUTHORS:            Ohm Patel - 7928829
                    Manal Shahab - 7933204
                    Aida Mesgar Zadeh - 7933484
                    Hana Dunlop - 7907031
VERSION:            December 02, 2022
*/

`timescale 1ns/1ps

module Whac_A_Mole( //top module
    input clk,
    input [9:0] sw,
    input reset,
    input [2:0] key,
    output [9:0] led,
    output [6:0] HEX0,
    output [6:0] HEX1,
    output [6:0] HEX2,
    output [6:0] HEX3
    );

    wire timer_out;
```

```
29      wire [1:0] counter_out;
30      wire [3:0] timer_ones,timer_tens,score_ones,score_tens;
31      wire [6:0] score_count;
32      wire [4:0] timer_count;
33
34      sub_bin2bcd bcd(timer_count, score_count, timer_ones,
35                      timer_tens, score_ones, score_tens);
36
37      sub_7segDisplay ssd0(score_ones,HEX0[6:0]);
38      sub_7segDisplay ssd1(score_tens,HEX1[6:0]);
39      sub_7segDisplay ssd2(timer_ones,HEX2[6:0]);
40      sub_7segDisplay ssd3(timer_tens,HEX3[6:0]);
41
42      sub_timerClock tclk(clk, timer_out);
43      sub_timerCount tcnt(clk, reset, timer_count, key);
44
45      sub_scoreCount scr(clk, sw, reset, key, led, score_count);
46
47  endmodule
48
49
50  module sub_bin2bcd(
51      input [7:0] timerInput,
52      input [7:0] scoreInput,
53      output [3:0] timerOnes,
54      output [3:0] timerTens,
55      output [3:0] scoreOnes,
56      output [3:0] scoreTens
57      );
58
59      assign timerOnes = timerInput % 10;
60      assign timerTens = ((timerInput-timerOnes)/10) % 10;
61
62      assign scoreOnes = scoreInput % 10;
63      assign scoreTens = ((scoreInput-scoreOnes)/10) % 10;
64
65  endmodule
66
67
68  module sub_7segDisplay(
69      input[3:0] hex,
70      output reg[6:0] ssd
71      );
```

```
72      always@(hex)
73          begin
74              case(hex)
75                  0: ssd=7'b1000000;
76                  1: ssd=7'b1111001;
77                  2: ssd=7'b0100100;
78                  3: ssd=7'b0110000;
79                  4: ssd=7'b0011001;
80                  5: ssd=7'b0010010;
81                  6: ssd=7'b0000010;
82                  7: ssd=7'b1111000;
83                  8: ssd=7'b0000000;
84                  9: ssd=7'b0010000;
85                  10: ssd=7'b0001000;
86                  11: ssd=7'b0000011;
87                  12: ssd=7'b1000110;
88                  13: ssd=7'b0100001;
89                  14: ssd=7'b0000110;
90                  15: ssd=7'b0001110;
91              endcase
92          end
93  endmodule
94
95
96  module sub_timerClock(
97      input clk_in,
98      output  reg clk_out
99      );
100
101     reg [27:0] period_count = 0;
102
103     always @ (posedge clk_in)
104         if (period_count != 50000000 - 1) //50 Mhz DE10 clock
105             begin
106                 period_count<=  period_count + 1;
107                 clk_out <= 0;
108             end
109         else
110             begin
111                 period_count <= 0;
112                 clk_out <= 1;
113             end
114
```

```
115 endmodule
116
117
118 module sub_timerCount(
119     input clk,
120     input reset,
121     output [4:0] count,
122     input [2:0] key
123     );
124
125     //note that the keys were assigned to key[3:1]
126     //on the FPGA. key[0] was used for reset button
127
128     wire clk_out;
129
130     sub_timerClock c1(clk, clk_out);
131
132     reg [4:0] timer = 0;
133
134     always @(posedge clk_out)
135         begin
136
137         if (key[2]==0) //press key[2] to set 30 second timer
138                 begin
139                     if(reset)
140                         timer <= 30;
141                     else if(timer == 0)
142                         timer <= timer; //stop timer at 0 seconds
143                     else if(timer >= 1)
144                         timer <= timer - 1;
145                     else
146                         timer <= timer;
147                 end
148
149         if (key[1]==0) //press key[1] to set 20 second timer
150                 begin
151                     if(reset)
152                         timer <= 20;
153                     else if(timer == 0)
154                         timer <= timer;
155                     else if(timer >= 1)
156                         timer <= timer - 1;
157                     else
```

```
158                         timer <= timer;
159                 end
160
161         if (key[0]==0) //press key[0] to set 10 second timer
162                 begin
163                     if(reset)
164                         timer <= 10;
165                     else if(timer == 0)
166                         timer <= timer;
167                     else if(timer >= 1)
168                         timer <= timer - 1;
169                     else
170                         timer <= timer;
171                 end
172
173         end
174
175     assign count = timer; //provide output of the count
176
177 endmodule
178
179
180 module sub_debounce(
181     input clk,
182     input reset,
183     output reg reset_db
184     );
185
186     reg key1,key2;
187     reg [15:0] count;
188
189     always @(posedge clk)
190         begin
191             key1 <= reset;
192             key2 <= key1;
193             if(reset_db==key2)
194                 count <=0;
195             else
196                 begin
197                     count <= count + 1'b1;
198                     if(count==16'hffff)
199                         reset_db  <= ~reset_db;
200                 end
```

```
201         end
202 endmodule
203
204
205 module sub_scoreCount(
206     input clk,
207     input [9:0] sw,
208     input rst,
209     input [2:0] key,
210     output reg [9:0] led,
211     output reg [5:0] score
212     );
213
214     localparam //set the score count to 32
215         S000000 = 0,
216         S000001 = 1,
217         S000010 = 2,
218         S000011 = 3,
219         S000100 = 4,
220         S000101 = 5,
221         S000110 = 6,
222         S000111 = 7,
223         S001000 = 8,
224         S001001 = 9,
225         S001010 = 10,
226         S001011 = 11,
227         S001100 = 12,
228         S001101 = 13,
229         S001110 = 14,
230         S001111 = 15,
231         S010000 = 16,
232         S010001 = 17,
233         S010010 = 18,
234         S010011 = 19,
235         S010100 = 20,
236         S010101 = 21,
237         S010110 = 22,
238         S010111 = 23,
239         S011000 = 24,
240         S011001 = 25,
241         S011010 = 26,
242         S011011 = 27,
243         S011100 = 28,
```

```
244         S011101 = 29,
245         S011110 = 30,
246         S011111 = 31,
247         S100000 = 32;
248 
249     reg [5:0] current=0; //current score
250     reg [5:0] next=0; //next score
251 
252     sub_debounce(clk,rst,reset);
253 
254     always @(posedge clk)
255         begin
256             if (reset)
257                 current <= S000000; //reset score
258             else
259                 current <= next;
260         end
261 
262     always @(current, sw[9:0])
263 
264         if (key[2]==0 | key[1]==0 | key[0]==0)
265 
266             begin
267                 case(current)
268 
269                     S000000: begin //0
270                             next <= S000000;
271                             led[9:0] <= 0;
272                             score <= 6'b000000;
273                             led[6] <= 1;
274                                 if (sw[6]==1)
275                                     //if correct switch toggled
276                                     next <= S000001;
277                                     //increment score
278                                 else
279                                     next <= S000000;
280                                     //else remain same
281                         end
282 
283                     S000001: begin //1
284                             score <= 6'b000001;
285                             led[9:0] <= 0;
286                             led[0] <= 1;
```

```
287                             if (sw[0]==1)
288                                 next <= S000010;
289                             else
290                                 next <= S000001;
291                         end
292
293                     S000010: begin //2
294                             score <= 6'b000010;
295                             led[9:0] <= 0;
296                             led[8] <= 1;
297                             if (sw[8]==1)
298                                 next <= S000011;
299                             else
300                                 next <= S000010;
301                         end
302
303                     S000011: begin //3
304                             score <= 6'b000011;
305                             led[9:0] <= 0;
306                             led[4] <= 1;
307                             if (sw[4]==1)
308                                 next <= S000100;
309                             else
310                                 next <= S000011;
311                         end
312
313                     S000100: begin //4
314                             score <= 6'b000100;
315                             led[9:0] <= 0;
316                             led[9] <= 1;
317                             if (sw[9]==1)
318                                 next <= S000101;
319                             else
320                                 next <= S000100;
321                         end
322
323                     S000101: begin //5
324                             score <= 6'b000101;
325                             led[9:0] <= 0;
326                             led[3] <= 1;
327                             if (sw[3]==1)
328                                 next <= S000110;
329                             else
```

```
330                                 next <= S000101;
331                         end
332
333                     S000110: begin //6
334                             score <= 6'b000110;
335                             led[9:0] <= 0;
336                             led[2] <= 1;
337                                 if (sw[2]==1)
338                                     next <= S000111;
339                                 else
340                                     next <= S000110;
341                         end
342
343                     S000111: begin //7
344                             score <= 6'b000111;
345                             led[9:0] <= 0;
346                             led[7] <= 1;
347                                 if (sw[7]==1)
348                                     next <= S001000;
349                                 else
350                                     next <= S000111;
351                         end
352
353                     S001000: begin //8
354                             score <= 6'b001000;
355                             led[9:0] <= 0;
356                             led[1] <= 1;
357                                 if (sw[1]==1)
358                                     next <= S001001;
359                                 else
360                                     next <= S001000;
361                         end
362
363                     S001001: begin //9
364                             score <= 6'b001001;
365                             led[9:0] <= 0;
366                             led[5] <= 1;
367                                 if (sw[5]==1)
368                                     next <= S001010;
369                                 else
370                                     next <= S001001;
371                         end
372
```

```
373                     S001010: begin //10
374                             score <= 6'b001010;
375                             led[9:0] <= 0;
376                             led[1] <= 1;
377                                 if (sw[1]==0)
378                                     next <= S001011;
379                                 else
380                                     next <= S001010;
381                         end
382
383                     S001011: begin //11
384                             score <= 6'b001011;
385                             led[9:0] <= 0;
386                             led[7] <= 1;
387                                 if (sw[7]==0)
388                                     next <= S001100;
389                                 else
390                                     next <= S001011;
391                         end
392
393                     S001100: begin //12
394                             score <= 6'b001100;
395                             led[9:0] <= 0;
396                             led[5] <= 1;
397                                 if (sw[5]==0)
398                                     next <= S001101;
399                                 else
400                                     next <= S001100;
401                         end
402
403                     S001101: begin //13
404                             score <= 6'b001101;
405                             led[9:0] <= 0;
406                             led[9] <= 1;
407                                 if (sw[9]==0)
408                                     next <= S001110;
409                                 else
410                                     next <= S001101;
411                         end
412
413                     S001110: begin //14
414                             score <= 6'b001110;
415                             led[9:0] <= 0;
```

```
416                         led[6] <= 1;
417                             if (sw[6]==0)
418                                 next <= S001111;
419                             else
420                                 next <= S001110;
421                     end
422
423                 S001111: begin //15
424                         score <= 6'b001111;
425                         led[9:0] <= 0;
426                         led[0] <= 1;
427                             if (sw[0]==0)
428                                 next <= S010000;
429                             else
430                                 next <= S001111;
431                     end
432
433                 S010000: begin //16
434                         score <= 6'b010000;
435                         led[9:0] <= 0;
436                         led[4] <= 1;
437                             if (sw[4]==0)
438                                 next <= S010001;
439                             else
440                                 next <= S010000;
441                     end
442
443                 S010001: begin //17
444                         score <= 6'b010001;
445                         led[9:0] <= 0;
446                         led[3] <= 1;
447                             if (sw[3]==0)
448                                 next <= S010010;
449                             else
450                                 next <= S010001;
451                     end
452
453                 S010010: begin //18
454                         score <= 6'b010010;
455                         led[9:0] <= 0;
456                         led[8] <= 1;
457                             if (sw[8]==0)
458                                 next <= S010011;
```

```
459                             else
460                                 next <= S010010;
461                         end
462
463                     S010011: begin //19
464                             score <= 6'b010011;
465                             led[9:0] <= 0;
466                             led[2] <= 1;
467                                 if (sw[2]==0)
468                                     next <= S010100;
469                                 else
470                                     next <= S010011;
471                         end
472
473                     S010100: begin //20
474                             score <= 6'b010100;
475                             led[9:0] <= 0;
476                             led[8] <= 1;
477                                 if (sw[8]==1)
478                                     next <= S010101;
479                                 else
480                                     next <= S010100;
481                         end
482
483                     S010101: begin //21
484                             score <= 6'b010101;
485                             led[9:0] <= 0;
486                             led[7] <= 1;
487                                 if (sw[7]==1)
488                                     next <= S010110;
489                                 else
490                                     next <= S010101;
491                         end
492
493                     S010110: begin //22
494                             score <= 6'b010110;
495                             led[9:0] <= 0;
496                             led[2] <= 1;
497                                 if (sw[2]==1)
498                                     next <= S010111;
499                                 else
500                                     next <= S010110;
501                         end
```

```
502
503                     S010111: begin //23
504                             score <= 6'b010111;
505                             led[9:0] <= 0;
506                             led[4] <= 1;
507                                 if (sw[4]==1)
508                                     next <= S011000;
509                                 else
510                                     next <= S010111;
511                         end
512
513                     S011000: begin //24
514                             score <= 6'b011000;
515                             led[9:0] <= 0;
516                             led[4] <= 1;
517                                 if (sw[4]==1)
518                                     next <= S011001;
519                                 else
520                                     next <= S011000;
521                         end
522
523                     S011001: begin //25
524                             score <= 6'b011001;
525                             led[9:0] <= 0;
526                             led[0] <= 1;
527                                 if (sw[0]==1)
528                                     next <= S011010;
529                                 else
530                                     next <= S011001;
531                         end
532
533                     S011010: begin //26
534                             score <= 6'b011010;
535                             led[9:0] <= 0;
536                             led[6] <= 1;
537                                 if (sw[6]==1)
538                                     next <= S011011;
539                                 else
540                                     next <= S011010;
541                         end
542
543                     S011011: begin //27
544                             score <= 6'b011011;
```

```
545                             led[9:0] <= 0;
546                             led[9] <= 1;
547                                 if (sw[9]==1)
548                                     next <= S011100;
549                                 else
550                                     next <= S011011;
551                         end
552 
553                     S011100: begin //28
554                             score <= 6'b011100;
555                             led[9:0] <= 0;
556                             led[3] <= 1;
557                                 if (sw[3]==1)
558                                     next <= S011101;
559                                 else
560                                     next <= S011100;
561                         end
562 
563                     S011101: begin //29
564                             score <= 6'b011101;
565                             led[9:0] <= 0;
566                             led[5] <= 1;
567                                 if (sw[5]==1)
568                                     next <= S011110;
569                                 else
570                                     next <= S011101;
571                         end
572 
573                     S011110: begin //30
574                             score <= 6'b011110;
575                             led[9:0] <= 0;
576                             led[0] <= 1;
577                                 if (sw[0]==0)
578                                     next <= S011111;
579                                 else
580                                     next <= S011110;
581                         end
582 
583                     S011111: begin //31
584                             score <= 6'b011111;
585                             led[9:0] <= 0;
586                             led[1] <= 1;
587                                 if (sw[1]==0)
```

```
588                                     next <= S100000;
589                                 else
590                                     next <= S011111;
591                             end
592
593                         S100000: begin //32
594                                 next <= S100000;
595                                 score <= 6'b100000;
596                                 led[9:0] <= 0;
597                             end
598
599                         default: begin
600                                 next = S000000;
601                             end
602                 endcase
603             end
604 endmodule
605
606 //End of program
```