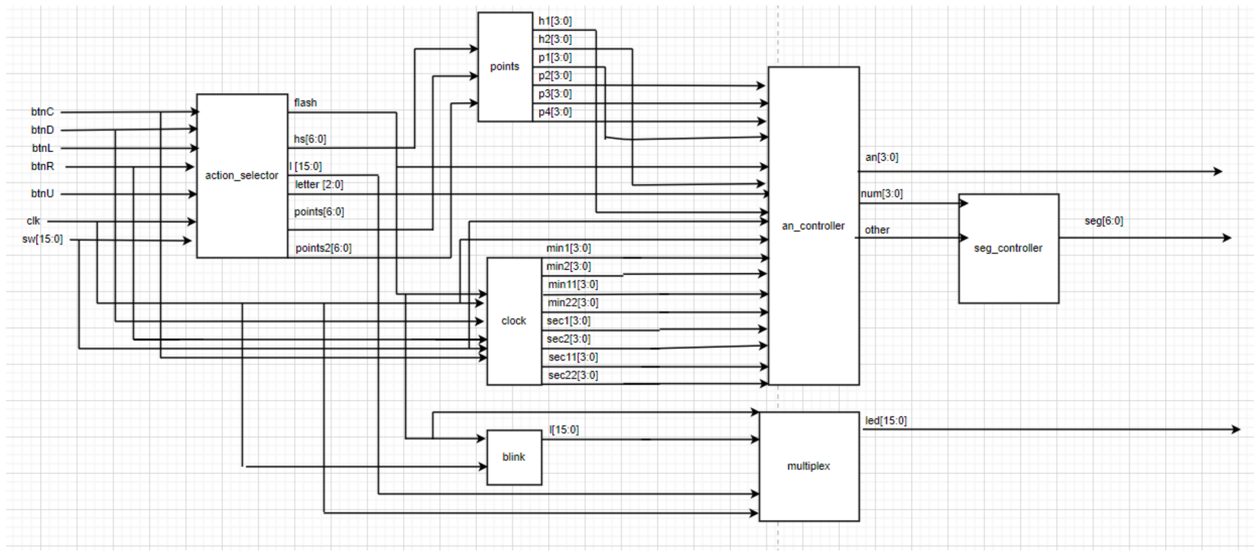
<u>Bop it Game Final Project Report</u>

Ethan Nagelvoort
Compe 470L
821234668
5/5/2021

Introduction

This project replicates a game called bop-it that keeps track of two users' scores through the 7 segment display on a Basys3 FPGA. In this game, the user has to switch up switches on the Basys3 whenever a led lights up above it. They also have to press the corresponding button depending on what letter appears in the furthest right seven segment display. It is important to note that all of this happens at a single time and so if either a letter can appear of the seven segment led or and led can light up above a switch. It utilizes an LFSR to randomly generate what LED will turn on or what letter will appear on the seven segment display. If the user takes longer than three seconds to turn an led off, then all the leds on the basys3 will blink. This indicates that the user has lost the game. This can also happen if the user flips the wrong switch or presses the wrong button than what is needed to proceed in the game. There is also a clock that keeps track of how long the user has lasted in the game and a module that keeps track of the user's points. The game is designed to be played with an opponent, so two different clock values and point values are saved at a time. This means that for every two games, both of those games point values and clock values will be saved and accessible in the Basys3 FPGA by use of manipulating certain switches when all leds are blinking which indicates a game is not being played at the moment. All point values and clock values saved can be manually changed if needed as well. This game also includes a difficulty setting and the ability to view the high score.

Block Diagram



Modules

Action selector and clk_div3 modules:

The action_selector module has all the buttons, clk, and all of sw as inputs. Its outputs are [15:0] l which represents the switches, b, flash, points, points2, letter, and hs. The regs declared are diff_sw, which represents a switch that controls difficulty, rand, rand2, rand3, lastRand, counter, flag, blink, pcounter, pholder, pcounter2, high_score, fb, and counter2. The rands represent random values that all take part in the lsfr to make rand random. This module instantiates clk into clk_div3 which allows the first always block for action_selector to have its clock operate at ¼ of a second. The action_selector is used to select which led will turn on or letter will appear that the user has to flip a switch or press a button for. This is done by using two always blocks with one having a case statement. Before this case statement the code checks in blink is on, if it is then we do not proceed to the case statement. It also makes sure rand is between 1 to 15. In the case statement is based on rand, which is a reg that is generated through an lfsr found in the other always block. This will turn on the leds and letters randomly instead of a predefined order. Inside each case statement, I check to see if the correct switch is flipped or button is pressed. If it is, I then check a reg, pcounter, that represents total points of the current player and see if it is less than 99. If it is then I set the current LED that is on to off, counter to 0, increment pcounter, and then check if diff_sw is 1 or 0. This reg determines the difficulty and so if it is 1, then the game will only operate with the first 7 leds. If it is 1 then check if {rand[2:0] , fb})>7 and if its is then rand <= {rand[2:0] , fb}/2-1 or rand <= ({rand[2:0] , fb})/2-rand3, depending on which case we are in. This will allow for the rand to be less than 7. If {rand[2:0] , fb}) <7 then rand <= {rand[2:0] , fb}, fb is a reg that will be explained later. If diff_sw is 0, then if counter2 is even, rand <= {rand[2:0] , fb}, else rand <= {rand[2:0] ,rand[4]^rand[3]}. Counter2 is a 32 bit number that increments every clock cycle in this always block. This is just to implement a more random rand generator because if I implemented the lfsr in the standard way, then there would be too much of a pattern. Then there is an else statement that is connected to the very first if previously described in this case statement description. In it the letter will be either 0 or a number that represents the letter needed. This will be described in the anode/seg module. L will represent what the led configuration needs to look like. Another if statement occurs that checks counter >= 1 or if the wrong switch or button is pressed. If so, then blink is 1, else counter is incremented. Counter represents the duration of time the player is on in this case statement. There are about 15 cases that generally operate like what was described above. Then after this case statement ends and else occurs based on the second if statement before the case statement. It has rand <= rand2 and then checks if counter>12, if it is then blink = 1. Then an end also occurs for the first if statement before the case statement and after I check if blink is 1. If it is, then pholder will equal pcounter. I then check if pholder is greater than high_score, if it is then high_score equals pholder. After I have a series of if statements that help allow the user to manually manipulate high_counter, pcounter, and pcounter2. Then another if statement occurs where if btnR is 1 and blink is 1, blink =0, counter = 0, pcounter2 = pholder, and pcounter = 0. I use btnR as a button to restart the game for a new player. I use pcounter, pholder, and pcounter2 in this way in order to save the total points from the last game and the game that just ended. After that I do a series of assigns that have the outputs equal the corresponding regs that they need to equal. Then I have my second always block that is just based on clk, the internal clock. In it, I increment rand2 and rand3. Then if rand2 equals 15, it gets set back to 1 and if rand3 equals 3, it gets set back to 1. Then in the last always block that is based on *, fb = and2[4]^rand2[3]. This is the feedback portion of the lfsr that allows for a random bit to be shifted into rand as seen in the earlier code.

Points

The points module has three 7 bit inputs, counter, counter2, and hs. The outputs are all 4 bit and they are p1, p2, p3, p4, h1, and h2. The inputs represent some outputs from the action_select module and represent its points, points2, and hs(high score) outputs. I then assign p1 to counter%10, this gives the first digit of counter to equal p1. I then assign p2 to counter/10, this gives the second digit of counter to equal p2. I then proceed to do that same process with the other outputs with corresponding inputs. This module acts as a way to split the inputs into two separate values representing its 2 digits.

Blink and clk_div2

This module takes in clk and flash as inputs and has a 16 bit output called l. Flash is connected to the flash output in the action_selector module. l represents the 16 leds on the basys3 and so is connected to led in the top module. I set a 16 bit reg called blink and a wire called CLK. I instantiate CLK and clk in clk_div2, this has the following always block that is based on CLK operating at ¼ of a second. Then in the always block, if flash is 1, then blink <= ~blink. This changes blink from all 1s to all 0s then back to all 1s really quickly. Then if flash is 0, blink remains as 0. Then at the end of the module, I assign l to blink. Doing this will make the leds flash on and off at ¼ of a second if the flash input is 1. Recall that if flash is 1, then the game is lost. Flashing leds indicate the player has lost.

Multiplex

The inputs in this module are 16 bit l1, 16 bit l2, 1 bit flash, and 1 bit clk. The output is 16 bit led. Since there is a conflict on which output led will use, either from action_selector or from blink, a multiplex module is used to determine which signal led will represent. This module takes in the led state from the action selector module and blink. It also takes in blink from the action selector module, now called flash. On the posedge of clk, if flash = 1, then the leds would equal the led state from blink (l2), and if flash = 0, then the leds would equal the led state from action_selector (l1).

Clock and clk_div

The inputs and outputs of this module are the following; input[15:0] sw, input clk,btnD, btnC, btnR, flash, output [3:0] min1, min2, sec1, sec2, min11, min22, sec11, and sec22. This module is in charge of keeping track of how long the player takes to play the game. I first declare 6 6-bit regs called, cm, cs, cm2, cs2, cmholder, and csholder. They are all initially equal to 0. I also instantiate clk and a wire called CLK to clk_div and this to make the first always block operate at a second. First in the always block, I check if btnR is pressed and if it is, cm2 = cmholder, cs2 = csholder, cm = 0, and cs = 0. Pressing btnR restarts the game and I want to be able to save the minutes and seconds of the game played before, this code does that. Csholder and cmholder will be explained later on. If btnR and flash is not 1, then it checks if cm is not 99 and cs is not 59. If this is true then cs is incremented. Then if cs is equal to 59, cm is incremented and cs is set back to 0. This code basically just increments a timer every second with max time at 99 minutes and 59 seconds. Then once flash equals 1, indicating game is lost, cmholder equals cm and csholder equals cs. This helps save these values for the future. After the always block, I split cm, cs, cm2, and cs2's digits and make them equal to the corresponding outputs. Hence, for example I do assign sec1 = cs%10 and assign sec2 = cs/10 to split cs into two outputs representing each digit in cs.

<u>An_controller and clk_div4</u>

The inputs and outputs of this module are as followed; input [15:0] sw, input [2:0] letter, input clk, flash, input [3:0] min1, min2, sec1, sec2, p1, p2, p3, p4, min11, min22, sec11, sec22, h1, h2, output reg [3:0] an, output reg [3:0] num,and output reg other. I declare 3 regs, an_pos, display_con, second_time, and hs_disp. I also declare a wire called CLK. I use this and clk to instantiate into clk_div4 to have the following always block operate fast enough to access to change every individual anode fast enough. In the always block, I declare display_con = sw[15], second_time = sw[14], and hs_disp = sw[13]. Hence, if display_con was 1 and the other two was 0, then player 1's time would be shown. If second_time was 1 and nothing else was 1, then player 2's time would be shown. If every one of those regs was 0 and flash was 1, then player1's and player2's points would be shown. If flash was 1 and hs_disp was 1 and the other regs were 0, then high score would be shown. If display_con was 0 and flash was 0, then the current players points would show with a letter, if there was one at the time, that the player would have to press a button for. In each one of these if statements there is a case statement that cycles through each of the anodes and declares the output num to what should be shown in that anode. Some of these case statements use the output other, and these are for when the anode needs to display a letter instead of a number. Hence if other is 1, then the anode needs to display a letter and num would equal the input letter. That input is connected to the output letter from the action_selector module. In the if statements that involve case statements that do not use other, other is equal to 0 right before the case statement.

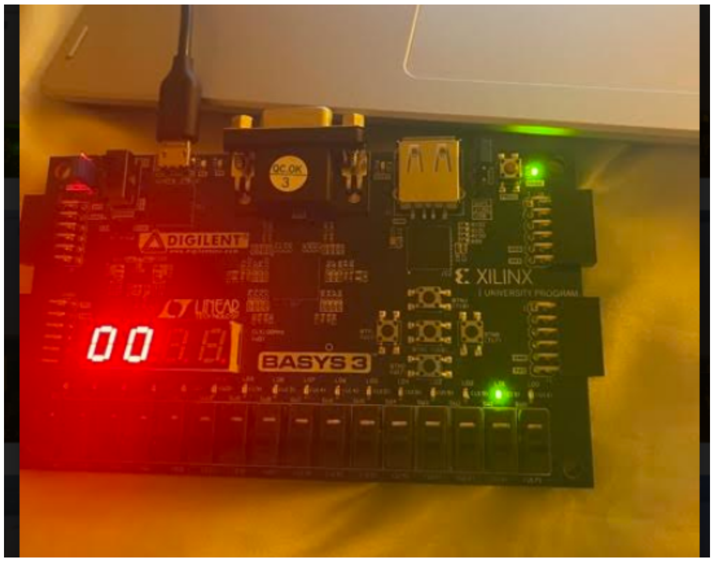
Seg_controller

______The inputs and outputs of this module are as followed; input [3:0] num, input other, and output reg[6:0] seg. This module is in charge of controlling the seven-segment display by controlling its leds. It has an always block based on * and first checks to see if other is 0. This other is connected to the other output in the anode_controller module. Hence, if other is 0, then the current anode does not need to display a letter and so a case statement based on num will be used to have seg equal the corresponding number to light it up on the seven segment display. Num is connected to the num output from the anode_controller module as well. Then if other is not 0, a case statement based on num will proceed to have seg equal a value that will light up the required letter on the seven segment. These include L, U, C, d, H, and I.

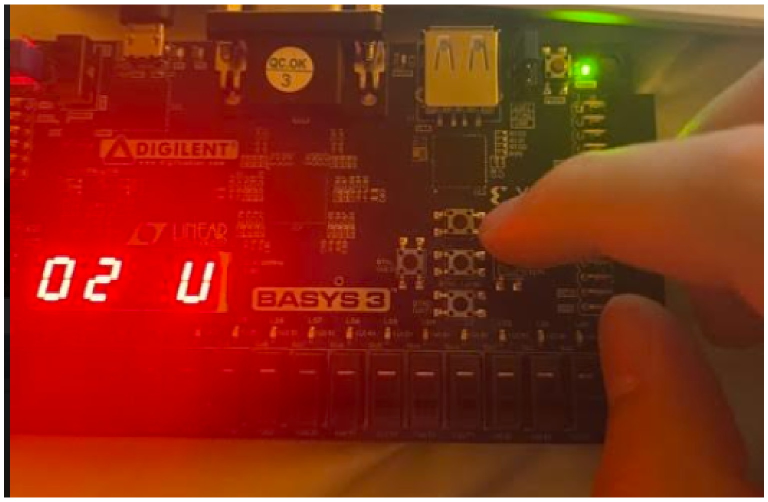
User Instructions

Once the game has started, flip the switch up and down when you see a led light up above it. If a letter appears on the far right side of the seven segment display, press the buttons to the right of it. As you do this, your point total should be increasing on the left side of the seven segment display. You have a period of 3 seconds to either flip the right switch or press the right button or all the leds will flash, indicating you lost. If you flip the wrong switch or press the wrong button, then the leds will flash too. Once all leds are flashing, the seven segment will show your score on the far left and the previous games score on the right. You can manually increment these values by pressing buttons btnC and btnD. You can then flip only switch 15 up to view how long you lasted in the game. You can manually change this by setting the first 7 switches to the binary value and pressing btnC to change seconds to that value. You can also press btnD to change minutes to that value. Then if you only flip switch 14 up, you can view how long the game before yours lasted. You can manipulate this timing the same way as previously described. If you only flip switch 13 up, you can view the high score, which is the highest score out of any game played while the Basys3 has been running. The seven segment display will display HI and then the score. You can manually increment this score by pressing btnC. If switch 11 is flipped up, then while you are playing the game, the fpga will only use the first 7 leds. It will not use the other leds or utilize letters or buttons in the game. This is basically an easy version of the game. You can restart the game once all leds are blinking by pressing btnR.

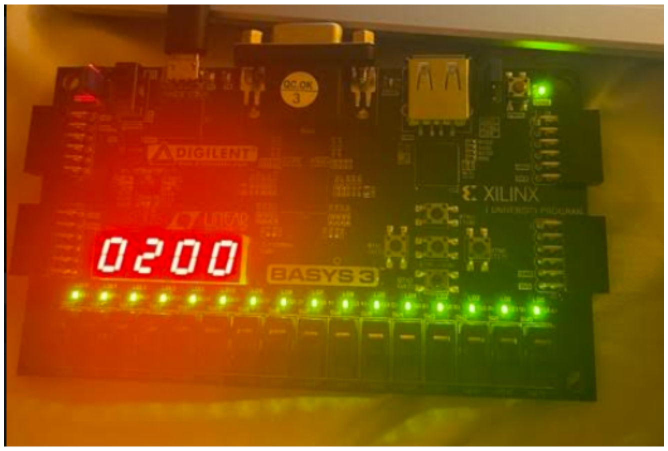
Test and Validation Results



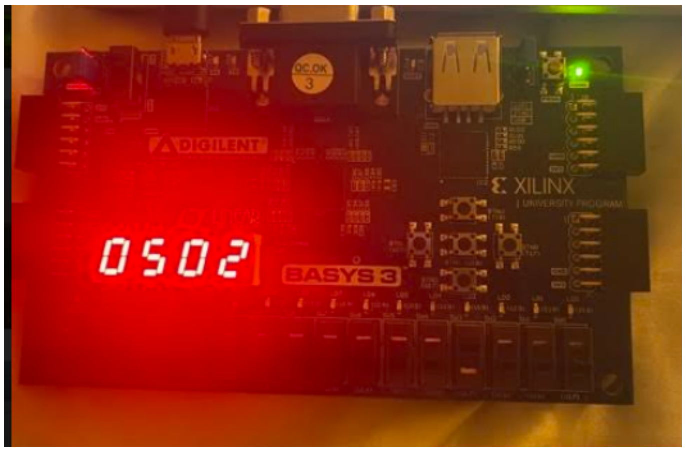
The image above shows when the game starts, one of the leds is on and 00 is shown to show that I have 0 points. Once I flipped switch 2 up and down, the led turned off and I gained a point. A different led turned on after.



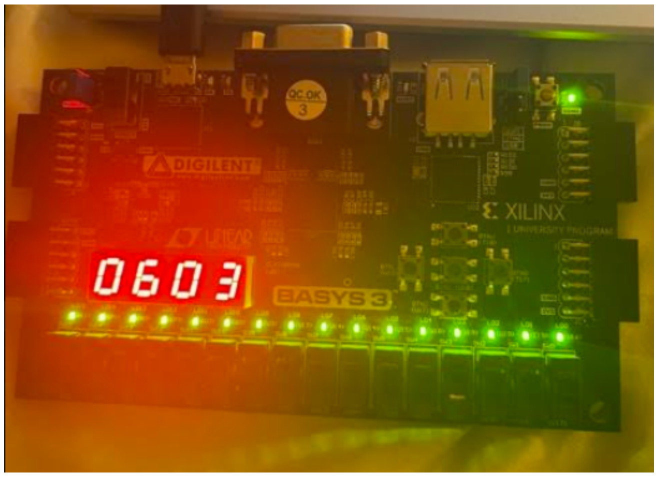
This was during the game, a letter showed up on the seven segment display which prompted me to press btnU. I gained a point after that and proceeded to play the rest of the game. This showed me that my letters are working correctly.



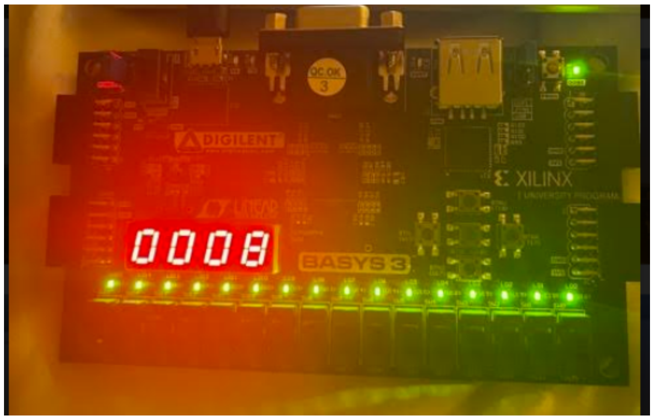
I started a new game to test for failure. During that game, after gaining 2 points, I waited 3 seconds. All the leds started to flash, indicating I had lost the game.



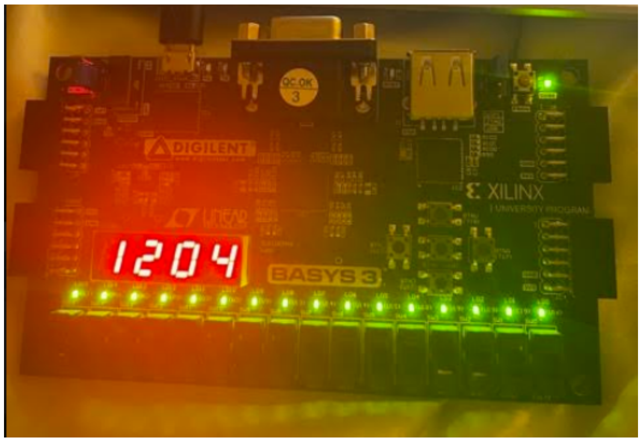
Then, after pressing btnR, the game restarted and I gained 14 points. This time I purposely flipped the wrong switch in order to test that the game would end if the wrong action was taken. The game ended and the leds started to flash, indicating that this function operates correctly. Also, after the game ended both the game I just played and the previous game's total points both appeared in the seven segment display in the correct positions.



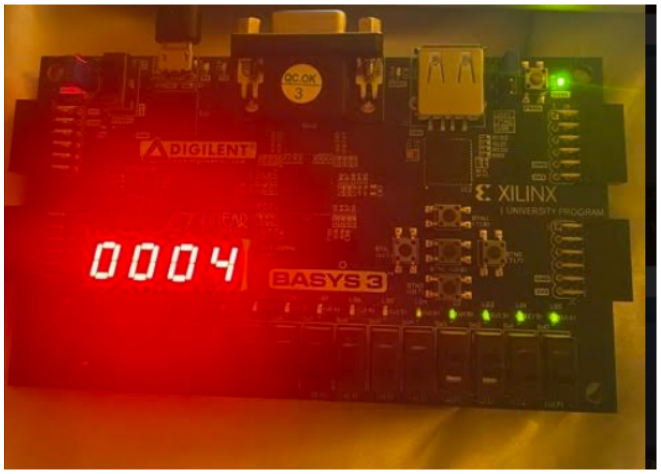
I then pressed btnC and btnD to increment the points and as shown by the picture above, it did increment. This means that this function works correctly.



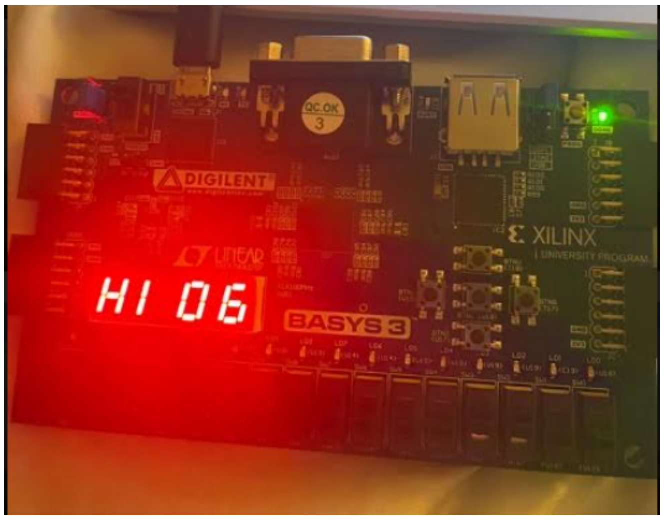
I then flipped switch 15 to see the timer for the game I just played and it is displayed above.



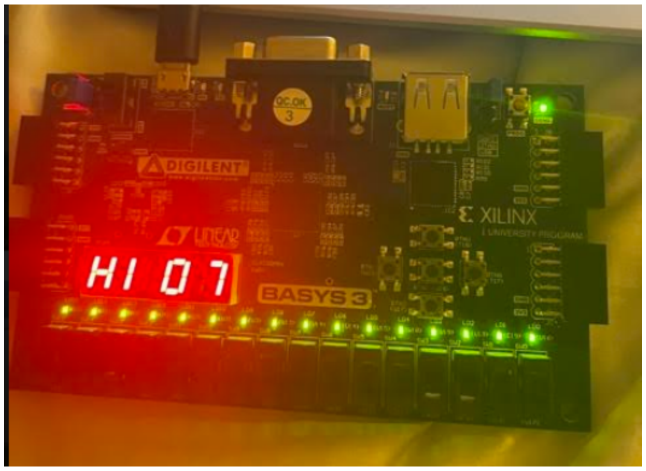
I then flipped the first couple of switches to represent the binary value of 4 and then pressed btnC to change the seconds to 4. I then changed the first couple of switches to represent binary 12 and then pressed btnD to change minutes to 12. This shows that manipulating the timers works as needed.



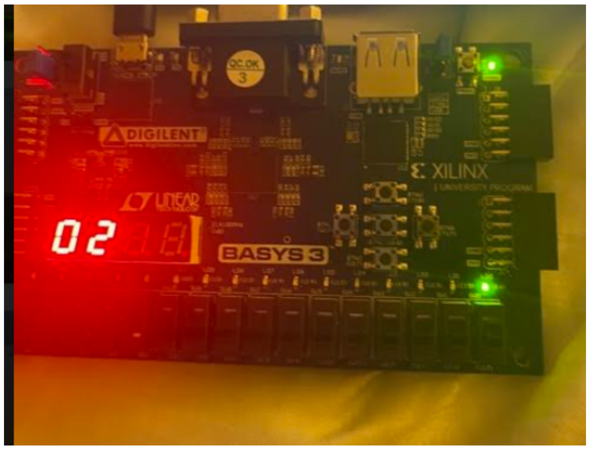
I then flipped up switch 14 to view the previous games timing and this showed up. This means it took me 4 seconds to play the previous game which is correct.

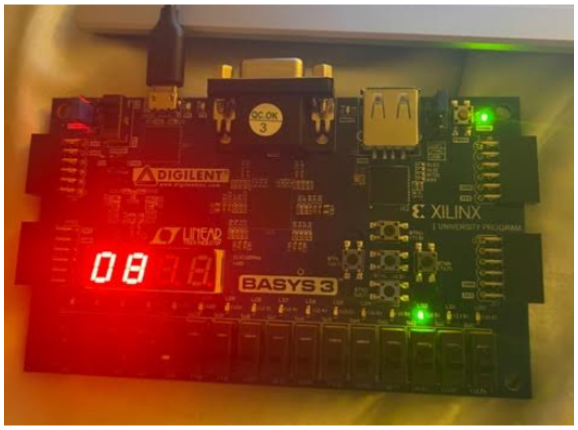


I then flipped up switch 13 to view the high score and this showed up. This is the correct high score and it is good to see HI also shows up indicating that it is showing the high score.



I then manually incremented the high score by pressing btnC. As shown, 6 was incremented to 7 which means that this function works correctly.





I then flipped up switch 11 to test out the easy mode of the game. I pressed btnR and proceeded to play the game. During the game, only the first 7 leds lit up, the other ones did not and no letters appeared on the seven segment display. This is shown in the above two images. This means that the easy mode of this game works correctly.

Source Code

```
module action_selector(input btnR, btnU, btnL, btnC, btnD, clk, input [15:0] sw,  output reg
[15:0] l, output wire b, output flash, output [6:0] points, output [6:0] points2, output reg [2:0]
letter, output [6:0] hs);
wire CLK;
reg busy = 0;
reg diff_sw = 0;
reg [3:0] rand = 2;
reg [2:0] rand3 = 1;
reg [3:0] rand2 = 1;
reg [4:0] lastRand = 1;
reg [3:0] counter = 0;
reg flag = 0;
reg blink = 0;
reg [6:0] pcounter = 0;
reg [6:0] pholder = 0;
reg [6:0] pcounter2 = 0;
reg [6:0] high_score = 0;
reg fb;
reg [31:0] counter2 = 0;
clk_div3 DUT(.clk(clk), .slowerClk(CLK));
always@(posedge CLK) begin
diff_sw <= sw[11];
counter2 <= counter2 +1;
if(blink==0) begin
if (rand>0 && rand<16) begin
case(rand)
1: begin
if(sw[0] == 1 && sw[10:1] == 0 && btnL == 0 && btnU == 0 && btnC == 0 && btnD == 0)
begin
busy <= 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
l[0] <= 1'b0;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= {rand[2:0] , fb}/2-1;
end
```

```
else
rand <= {rand[2:0] , fb};
end
if(diff_sw ===0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[0] <= 1'b1;
l[15:1] <= 0;
flag <= 0;
if(counter >= 1 && (sw[10:1] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD == 1))
blink <= 1;
else
counter <= counter + 1;
end
end

2: begin
if(sw[1] ==1 && sw[0]==0 && sw[10:2] == 0 && btnL == 0 && btnU == 0 && btnC == 0 &&
btnD == 0) begin
l[1] <= 0;
busy <= 0;
flag <= 1;
if(pcounter < 99)
pcounter <= pcounter + 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-1;
end
else
rand <= {rand[2:0] , fb};
```

```
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[1] <= 1'b1;
l[0] <= 0;
l[15:2] = 0;
flag <= 0;
if (counter>=1 && (sw[0]>=1 || sw[10:2] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD ==
1))
blink <= 1;
else
counter <= counter +1;
end
end

3: begin
if(sw[2] ==1 && sw[1:0] == 0 && sw[10:3] == 0 && btnL == 0 && btnU == 0 && btnC == 0
&& btnD == 0) begin
l[2] <= 0;
busy <= 0;
flag <= 1;
if(pcounter < 99)
pcounter <= pcounter + 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-1;
end
else
rand <= {rand[2:0] , fb};
```

```
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[2] <= 1'b1;
l[1:0] <= 0;
l[15:3] <= 0;
flag <= 0;
if(counter >= 1 && (sw[1:0] >= 1 || sw[10:3] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD
== 1))
blink <= 1;
else
counter <= counter +1;
end
end

4: begin
if(sw[3] ==1 && sw[2:0] == 0 && sw[10:4] == 0 && btnL == 0 && btnU == 0 && btnC == 0
&& btnD == 0) begin
l[3] <= 0;
busy <= 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
```

```
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[3] <= 1'b1;
l[2:0] <= 0;
l[15:4] <= 0;
flag <= 0;
if(counter>=1 && (sw[2:0] >= 1 || sw[10:4] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD ==
1))
blink <= 1;
else
counter <= counter +1;
end
end

5: begin
if(sw[4] ==1 && sw[10:5] == 0 && sw[3:0] == 0 && btnL == 0 && btnU == 0 && btnC == 0
&& btnD == 0) begin
l[4] <= 0;
busy <= 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
```

```
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[4] <= 1'b1;
l[3:0] <= 0;
l[15:5] <= 0;
flag <= 0;
if(counter >= 1 && (sw[10:5] >= 1 || sw[3:0] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD
== 1))
blink <= 1;
else
counter <= counter +1;
end
end

6: begin
if(sw[5] ==1 && sw[4:0] == 0 && sw[10:6] == 0 && btnL == 0 && btnU == 0 && btnC == 0
&& btnD == 0) begin
l[5] <= 0;
busy <= 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
```

```
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[5] <= 1'b1;
l[15:6] <= 0;
l[4:0] <= 0;
flag <= 0;
if(counter >= 1 && (sw[4:0] >= 1 || sw[10:6] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD
== 1))
blink <= 1;
else
counter <= counter +1;
end
end

7: begin
if(sw[6] ==1 && sw[5:0] == 0 && sw[10:7] == 0 && btnL == 0 && btnU == 0 && btnC == 0
&& btnD == 0) begin
l[6] <= 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
busy <= 0;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
```

```
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[6] <= 1'b1;
l[5:0] <= 0;
l[15:7]<=0;
flag <= 0;
if(counter >= 1 && (sw[10:7] >= 1 || sw[5:0] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD
== 1))
blink <= 1;
else
counter <= counter +1;
end
end

8: begin
if(sw[7] ==1 && sw[6:0] == 0 && sw[10:8] == 0 && btnL == 0 && btnU == 0 && btnC == 0
&& btnD == 0) begin
l[7] <= 0;
busy <= 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
```

```
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[7] <= 1'b1;
l[6:0] <= 0;
l[15:8] <= 0;
flag <= 0;
if(counter>=1 && (sw[10:8] >= 1 || sw[6:0] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD ==
1))
blink <= 1;
else
counter <= counter +1;
end
end

9: begin
if(sw[8] ==1 && sw[7:0]==0 && sw[10:9] == 0 && btnL == 0 && btnU == 0 && btnC == 0
&& btnD == 0) begin
l[8] = 0;
busy = 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
```

```
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[8] = 1'b1;
l[15:9] <= 0;
l[7:0] <= 0;
flag <= 0;
if(counter>=1 && (sw[10:9] >= 1 || sw[7:0] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD ==
1))
blink <= 1;
else
counter <= counter +1;
end
end

10:begin
if(sw[10:0] == 0 && btnL == 1 && btnU == 0 && btnC == 0 && btnD == 0) begin
busy = 0;
letter <= 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
end
```

```
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 1;
busy <= 1;
flag <= 0;
if(counter>=1 && (sw[10:0] >= 1 || btnU == 1 || btnC == 1 || btnD == 1))
blink <= 1;
else
counter <= counter +1;
end
end

11:begin
if(sw[10:0] == 0 && btnL == 0 && btnU == 1 && btnC == 0 && btnD == 0) begin
busy = 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
letter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
```

```
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 2;
busy <= 1;
flag <= 0;
if(counter>=1 && (sw[10:0]>= 1 || btnL == 1 || btnC == 1 || btnD == 1))
blink <= 1;
else
counter <= counter +1;
end
end

12:begin
if(sw[10:0] == 0 && btnL == 0 && btnU == 0 && btnC == 1 && btnD == 0) begin
busy = 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
letter <= 0;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
```

```
letter <= 3;
busy <= 1;
flag <= 0;
if(counter>=1 && (sw[10:0]>= 1 || btnL == 1 || btnU == 1 || btnD == 1))
blink <= 1;
else
counter <= counter +1;
end
end

13:begin
if(sw[10:0] == 0 && btnL == 0 && btnU == 0 && btnC == 0 && btnD == 1) begin
busy = 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
letter <= 0;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 4;
busy <= 1;
flag <= 0;
if(counter >=1 && (sw[10:0]>= 1 || btnL == 1 || btnU == 1 || btnC == 1))
blink <= 1;
```

```
else
counter <= counter +1;
end
end

14: begin
if(sw[9] ==1 && sw[8:0]==0 && sw[10] == 0 && btnL == 0 && btnU == 0 && btnC == 0 &&
btnD == 0) begin
l[9] = 0;
busy = 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[9] = 1'b1;
l[15:10] <= 0;
l[8:0] <= 0;
flag <= 0;
if(counter>=1 && (sw[10] >= 1 || sw[8:0] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD ==
1))
blink <= 1;
```

```
else
counter <= counter +1;
end
end

15: begin
if(sw[10] ==1 && sw[9:0]==0  && btnL == 0 && btnU == 0 && btnC == 0 && btnD == 0)
begin
l[10] = 0;
busy = 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
end
if(diff_sw==0)begin
if(counter2%2==0) begin
if({rand[2:0] , fb} == 15)
rand <= rand2;
else
rand <= {rand[2:0] , fb};
end
else begin
if({rand[2:0] ,rand[4]^rand[3]}==15)
rand <= rand2;
else
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[10] = 1'b1;
```

```
l[15:11] <= 0;
l[9:0] <= 0;
flag <= 0;
if(counter>=1 && (sw[9:0] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD == 1))
blink <= 1;
else
counter <= counter +1;
end
end

default: begin
if(sw[0] == 1 && sw[10:1] == 0 && btnL == 0 && btnU == 0 && btnC == 0 && btnD == 0)
begin
busy <= 0;
if(pcounter < 99)
pcounter <= pcounter + 1;
l[0] <= 1'b0;
flag <= 1;
counter <= 0;
if(diff_sw == 1)begin
if(({rand[2:0] , fb})>7) begin
rand <= ({rand[2:0] , fb})/2-rand3;
end
else
rand <= {rand[2:0] , fb};
end
if(diff_sw==0)begin
if(counter2%2==0) begin
rand <= {rand[2:0] , fb};
end
else begin
rand <= {rand[2:0] ,rand[4]^rand[3]};
end
end
end
else begin
letter <= 0;
busy <= 1;
l[0] <= 1'b1;
l[15:1] <= 0;
```

```
flag <= 0;
if(counter>=1 && (sw[10:2] >= 1 || btnL == 1 || btnU == 1 || btnC == 1 || btnD == 1))
blink <= 1;
else
counter <= counter + 1;
end
end

endcase
end
else begin
rand <= rand2;
end
end
if(counter > 12) begin
blink <= 1;
end
if(blink ==1)
pholder <= pcounter;
if(high_score < pholder)
high_score <= pholder;
if(blink==1 && btnC == 1 && sw[13]==1 && sw[15]==0 && sw[14]==0)
high_score <= high_score + 1;
if(blink == 1 && btnC == 1 && sw[15] ==0 && sw[14]==0)
pcounter <= pcounter+1;
if(blink == 1 && btnD == 1 && sw[15] == 0 && sw[14]==0)
pcounter2 <= pcounter2 + 1;
if(btnR == 1 && blink == 1)
begin
blink <= 0;
counter <= 0;
pcounter2 <= pholder;
pcounter <= 0;
end
end
assign b = busy;
//assign flag2 = flag;
assign flash = blink;
assign points = pcounter;
assign points2 = pcounter2;
assign hs = high_score;
always @(posedge clk) begin
rand2 <= rand2 + 1;
rand3 <= rand3 +1;
if(rand2 == 15)
rand2 <= 1;
if(rand3 == 3)
rand3 <= 1;
end
always @(*)begin
fb = rand2[4]^rand2[3];
end

endmodule

module points(input [6:0] counter, input [6:0] counter2, hs, output [3:0] p1, p2, p3, p4, h1, h2);
assign p1 = counter%10;
assign p2 = counter/10;
assign p3 = counter2%10;
assign p4 = counter2/10;
assign h1 = hs%10;
assign h2 = hs/10;
endmodule

module blink(input clk, flash, output [15:0] l);
reg [15:0] blink = 1'b0000000000000000;
wire CLK;
clk_div2 DUT(.clk(clk), .slowerClk(CLK));
always@(posedge CLK) begin
if(flash) begin
blink <= ~blink;
end
else
blink<= 0;
end
assign l = blink;
endmodule

module multiplex(input [15:0] l1, l2, input flash, clk, output reg [15:0] led);
always@(posedge clk) begin
```

```
if(flash)
led <= l2;
else
led <= l1;
end
endmodule

module clock(input[15:0] sw, input clk,btnD, btnC, btnR, flash, output [3:0] min1, min2, sec1,
sec2, min11, min22, sec11, sec22);
wire CLK;
clk_div DUT(.clk(clk), .slowerClk(CLK));
reg [6:0] cm = 6'b0000000; //min and sec counter
reg [6:0] cs = 6'b0000000;
reg [6:0] cm2 = 6'b0000000; //min and sec counter
reg [6:0] cs2 = 6'b0000000;
reg [6:0] cmholder = 6'b0000000; //min and sec counter
reg [6:0] csholder = 6'b0000000;
always@(posedge CLK) begin
if(btnR == 1)
begin
cm2 <= cmholder;
cs2 <= csholder;
cm <= 0;
cs <= 0;
end
if(btnR!=1) begin
if(flash != 1) begin
if(cm != 99 && cs != 59) begin
cs <= cs+1;
if(cs == 59) begin
cm <= cm + 1;
cs <= 0;
end
end
end
end
if(flash == 1)begin
cmholder <= cm;
csholder <= cs;
end
```

```
if(flash == 1 && sw[15] == 1 && sw[14]==0 && btnC==1)
cs<=sw[6:0];
if(flash == 1 && sw[15] == 1 && sw[14]==0 && btnD==1)
cm<=sw[6:0];
if(flash == 1 && sw[15] == 0 && sw[14]==1 && btnC==1)
cs2<=sw[6:0];
if(flash == 1 && sw[15] == 0 && sw[14]==1 && btnD==1)
cm2<=sw[6:0];
end
assign sec1 = cs%10;
assign sec2 = cs/10;
assign min1 = cm%10;
assign min2 = cm/10;
assign sec11 = cs2%10;
assign sec22 = cs2/10;
assign min11 = cm2%10;
assign min22 = cm2/10;
endmodule

module an_controller(input [15:0] sw, input [2:0] letter, input clk, flash, input [3:0] min1, min2,
sec1, sec2, p1, p2, p3, p4, min11, min22, sec11, sec22, h1, h2, output reg [3:0] an, output reg
[3:0] num, output reg other);
reg[3:0] an_pos = 0;
wire CLK;
reg display_con = 1'b0;
reg second_time = 1'b0;
reg hs_disp = 1'b0;
clk_div4 DUT(.clk(clk), .slowerClk(CLK));
always @(posedge CLK)
begin
display_con = sw[15];
second_time = sw[14];
hs_disp = sw[13];
if(display_con == 0 && second_time == 1 && hs_disp == 0) begin
other <= 0;
   case (an_pos)
      4'b0000: begin
         an <= 4'b1110;
         num <= sec11;
         an_pos <= 1;
```

```
            end
        4'b0001: begin
            an <= 4'b1101;
            num <= sec22;
            an_pos <= 2;
            end
        4'b0010: begin
            an <= 4'b1011;
            num<= min11;
            an_pos <= 3;
            end
        4'b0011: begin
            an <= 4'b0111;
            num <= min22;
            an_pos <= 0;
            end
            endcase
end
if(display_con == 1 && second_time == 0 && hs_disp==0) begin
other <= 0;
    case (an_pos)
        4'b0000: begin
            an <= 4'b1110;
            num <= sec1;
            an_pos <= 1;
            end
        4'b0001: begin
            an <= 4'b1101;
            num <= sec2;
            an_pos <= 2;
            end
        4'b0010: begin
            an <= 4'b1011;
            num<= min1;
            an_pos <= 3;
            end
        4'b0011: begin
            an <= 4'b0111;
            num <= min2;
            an_pos <= 0;
            end
            endcase
    end
    if(display_con == 0 && flash == 1 && second_time == 0 && hs_disp==0)
    begin
    other <= 0;
    case (an_pos)
        4'b0000: begin
            an <= 4'b1110;
            num <= p3;
            an_pos <= 1;
            end
        4'b0001: begin
            an <= 4'b1101;
            num <= p4;
            an_pos <= 2;
            end
        4'b0010: begin
            an <= 4'b1011;
            num<= p1;
            an_pos <= 3;
            end
        4'b0011: begin
            an <= 4'b0111;
            num <= p2;
            an_pos <= 0;
            end
            endcase
    end
    if(display_con == 0 && flash == 1 && second_time == 0 && hs_disp==1)
    begin
    case (an_pos)
        4'b0000: begin
            an <= 4'b1110;
            num <= h1;
            an_pos <= 1;
            other <= 0;
            end
        4'b0001: begin
            an <= 4'b1101;
            num <= h2;
            other <= 0;
            an_pos <= 2;
            end
        4'b0010: begin
            an <= 4'b1011;
            num<= 6;
            other<=1;
            an_pos <= 3;
            end
        4'b0011: begin
            an <= 4'b0111;
            num <= 5;
            other<=1;
            an_pos <= 0;
            end
            endcase
end
if(display_con == 0 && flash == 0)
begin
case (an_pos)
        4'b0010: begin
            an <= 4'b1011;
            num <= p1;
            other <= 0;
            an_pos <= 1;
            end
        4'b0011: begin
            an <= 4'b0111;
            num <= p2;
            other <= 0;
            an_pos <= 2;
            end
        4'b0000: begin
            an <= 4'b1101;
            other<=1;
            num<= 0;
            an_pos <= 3;
            end
        4'b0001: begin
```

```
            an <= 4'b1110;
            other <= 1;
            num <= letter;
            an_pos <= 0;
            end
            endcase
end
end

endmodule

module seg_controller(input [3:0] num, input other, output reg[6:0] seg);

always @(*)
begin
if(other == 0) begin
    case(num)
        4'b0000: seg = 7'b1000000; // 0000
        4'b0001: seg = 7'b1111001; // 0001
        4'b0010: seg = 7'b0100100; // 0010
        4'b0011: seg = 7'b0110000; // 0011
        4'b0100: seg = 7'b0011001; // 0100
        4'b0101: seg = 7'b0010010; // 0101
        4'b0110: seg = 7'b0000010; // 0110
        4'b0111: seg = 7'b1111000; // 0111
        4'b1000: seg = 7'b0000000; // 1000
        4'b1001: seg = 7'b0010000; // 1001
        default: seg = 7'b1000000; // 0000
    endcase
end
else begin
 case(num)
        4'b0000: seg = 7'b1111111; // nothing on
        4'b0001: seg = 7'b1000111; // L
        4'b0010: seg = 7'b1000001; // U
        4'b0011: seg = 7'b1000110; // C
        4'b0100: seg = 7'b0100001; // D
        4'b0101: seg = 7'b0001001; // H
        4'b0110: seg = 7'b1001111; // I
        default: seg = 7'b1111111; // nothing on
    endcase
end
end
endmodule

module clk_div3 (input clk, output reg slowerClk);
reg [31:0] counter;
always@(posedge clk) begin
if(counter == 25000000) begin//100000000 is threshold, it is close to 1 sec 6250000
slowerClk <= 1;//signal new clk signal
counter <= 0;
end
else begin
slowerClk <= 0;
counter <= counter+1;
end
end

endmodule
module clk_div2 (input clk, output reg slowerClk);
reg [31:0] counter;
always@(posedge clk) begin
if(counter == 25000000) begin//100000000 is threshold, it is close to 1 sec 6250000
slowerClk <= 1;//signal new clk signal
counter <= 0;
end
else begin
slowerClk <= 0;
counter <= counter+1;
end
end

endmodule

module clk_div (input clk, output reg slowerClk);
reg [31:0] counter;
always@(posedge clk) begin
if(counter == 100000000) begin//100000000 is threshold, it is close to 1 sec 6250000
slowerClk <= 1;//signal new clk signal
counter <= 0;
```

```
end
else begin
slowerClk <= 0;
counter <= counter+1;
end
end

endmodule

module clk_div4 (input clk, output reg slowerClk);
reg [31:0] counter;
always@(posedge clk) begin
if(counter == 20000) begin//100000000 is threshold, it is close to 1 sec 6250000
slowerClk <= 1;//signal new clk signal
counter <= 0;
end
else begin
slowerClk <= 0;
counter <= counter+1;
end
end

endmodule

module top(input clk, input btnR, btnC, btnL, btnD, btnU, input [15:0] sw, output [15:0] led,
output [3:0] an, output [6:0] seg);
wire busy, other;
wire [2:0] letter;
wire [3:0] p1, p2, sec1, sec2, min1, min2, sec11, sec22, min11, min22, num, p3, p4, h1, h2;
wire [6:0] points, points2, hs;
wire flash, flag;
wire [15:0] l1, l2;
action_selector DUT1 (.hs(hs),.clk(clk), .btnR(btnR), .letter(letter), .btnC(btnC), .btnL(btnL),
.btnD(btnD), .btnU(btnU), .points(points), .points2(points2), .sw(sw), .l(l1), .b(busy),
.flash(flash));
points DUT2 (.h1(h1), .h2(h2), .counter(points), .p1(p1), .p2(p2), .p3(p3), .p4(p4),
.counter2(points2), .hs(hs));
blink DUT3 (.clk(clk), .flash(flash), .l(l2));
multiplex DUT4(.clk(clk), .flash(flash), .l1(l1), .l2(l2), .led(led));
```

```
clock DUT5 (.sec11(sec11), .sec22(sec22), .min11(min11), .min22(min22),.sw(sw),.btnD(btnD),
.btnC(btnC), .clk(clk), .btnR(btnR), .flash(flash), .sec1(sec1), .sec2(sec2), .min1(min1),
.min2(min2));
an_controller DUT6 (.h1(h1), .h2(h2),.sec11(sec11), .sec22(sec22), .min11(min11),
.min22(min22),.clk(clk), .letter(letter), .flash(flash), .sec1(sec1), .sec2(sec2), .min1(min1),
.min2(min2), .p1(p1), .p2(p2), .p3(p3), .p4(p4), .sw(sw), .num(num), .an(an), .other(other));
seg_controller DUT7 (.num(num), .seg(seg), .other(other));
endmodule
```