Lab 3: FPGA-based Mental Binary Math Game

(ECE6370)

Raj Pramod Dasadia

.

1. **Introduction:**

   The game, mental binary math is a great way to have fun with math. It's a competitive game in which player's binary and decimal math is tested.

   Before the game begins, Authentication of a player is done by entering the last 4 digits of their ID,

   Each digit is entered at a time with a fixed set of 4 bit toggle-switches. Every time a digit's binary value is toggled, a load password switch is to be pressed, the process is repeated for all the 4 digits.

   Once the authentication is done, a green LED glows. If the authentication is not done, the red LED is on the game will not start and the players will not be able to load their inputs in their corresponding 7 segment until the authentication is done.

   After the authentication is done, a timer value needs to be set , since it is a two digit decimal counter, 8 toggle switches are used to set the timer. Once the timer is set Player has to press the Game start button one more time to load the timer value. The next game start button push begins the game. Once the timer starts counting down, RNG button is pressed to generate a random value

   Now Player has to analyze the decimal value of that hexadecimal number displayed and then guess a number so that the sum is 15 and then guess the binary equivalent of that number and switch player's set of 4 bit toggle switches and then press load button of to load a valid hexadecimal number to make the sum '15' and display the sum. If the challenge is completed successfully by the player, a green light is turned on, and the red light switches off.
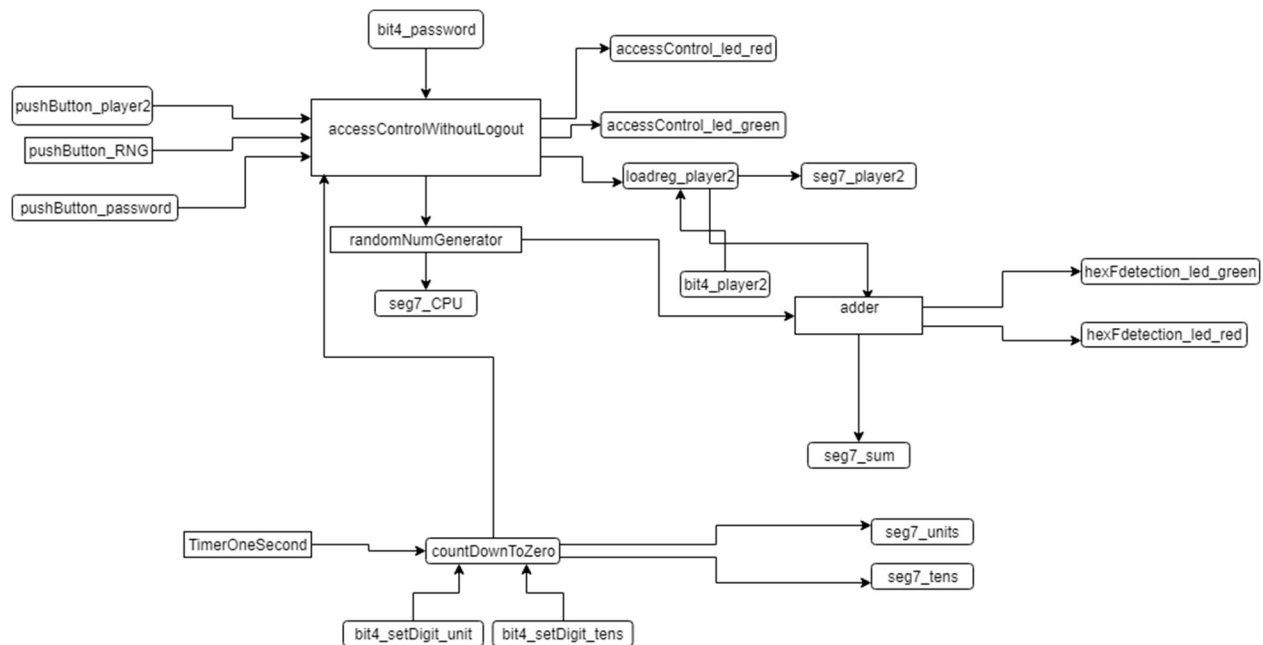
2. Top Module Architecture

Figure 1

The figure 1 above is the schematic of the top module of lab 2. It connects or wires different independent modules having independent functionalities to produce desired output.

The different modules used here are

a. accessControllerwithoutLogout
b. counter and randomNumGenerator (counter instantiated in the randomNumGenerator)
c. countdownToZero which instantiates modules – Timer1ms, countTo100, countTo10, Timer100ms, TimerOneSecond
d. loadReg
e. singlePulseButton
f. Seg7decoderDisplay
g. winDetector
h. adder

2.1 AccessControlWithoutlogOut

1. Functionality:
   Access Control provides authentication and security. Access controller is designed to accept 4 decimal digits (each 4 bit) one at a time, and checks the entered digits with the authenticated passcode of 4 decimal digits. If the passcode is correct, the push button from the external side is connected to the load register inside the module, thus providing the user an access to the actual functionality of the system. The push button from the external side is always set to 0, thereby providing no access to the user for the actual functionality of the system (Game). Actual access to the Push button to load Player's toggle switch input and RNG button is only provided when The game is in game start mode.

   The code is entered in the following fashion.
   The first digit is entered by switching the specific set of 4 bit toggle switches. As the first digit is entered, a load button is pushed to load the first digit in the module.
   Similarly all the remaining 3 digits are entered and when all the digits match the password saved in the module, access to the load register is granted to the push buttonsignal, thereby allowing the user to operate. As an indication a Green LED glows.

   If the password entered is incorrect, the user needs to enter the correct password again, in all the other times, the push button signal is zero and user is not granted an access. Red LED glows all the other times.

   Once the password is authenticated the timer for the game is to be set, this is done by using the 8 toggle switches dedicated for the game timer and once both the digit of the timer are set, push button for password is pressed again and the game enters in the game begin stage. Only in this stage the access control provides an access to the RNG and the pushLoad button for player 2's toggled 4 bit input

2. Module design
   2.1 One-Procedure Finite state Machine

s1
If load_password is 1 check the password with the passinput4bit, if incorrect set flag 0

s2
If load_password is 1 check the password with the passinput4bit, if incorrect set flag 0

s3
If load_password is 1 check the password with the passinput4bit, if incorrect set flag 0

s4
If load_password is 1 check the password with the passinput4bit, if incorrect set flag 0

s5
If flag is 0 go to s6 else go to s7

s6
go to s1

s7
passledred = 0 passledgreen = 1 go to s8

s8
enable_internal = 1 if load_password = 1 set reconfig and go to s9

s9
reconfig = 0 if load_password = 1 go to s10

s10
load_sig_RNG = pushButtonLoad_RNG load_sig_2 = pushButtonLoad2; enable <= 1;

load_password = 1

pause: set load_sig_RNG to 0 pushButtonLoad_rng = 1 enable = 0

laod_password = 1 watchdogTimer = 0

pause1:
enable_internal = 1 watchdogTimer = 0

timeout = 1

pause2:
Go to state 1

s11:
enable = 0 pushButtonLoad_RNG = 1 load_sig_2 = 0

s12:
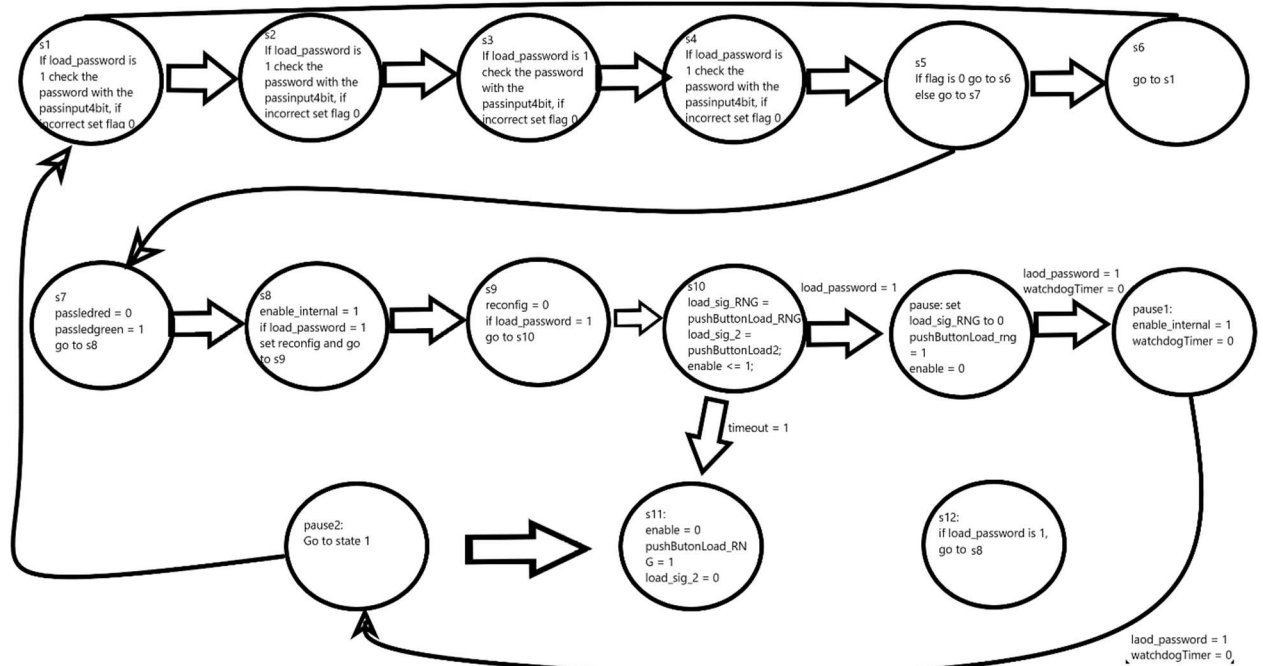if load_password is 1, go to s8

laod_password = 1 watchdogTimer = 0

Figure 2

2.1.1 Finite state machine

In figure 1, all the states and the functionality at each state is shown. Before entering the first state, the module is initialized by setting the value of load signal to the load_reg to 0 and the RNGPushButton as 1 so that in any case the user is not granted any access to the game or to load the numbers in the sum module until the state s10 which is the game begin stage.

State 1, State 2, State 3, State 4: if load_password signal is high, every 4 bit decimal input is checked in each stage, if the bit matches with the predefined digit, flag is not changed else flag is set to 0.

State 5: flag is compared, if it is 0, it is sent to stage 6 else it is sent to stage 7.

State 6: Unconditionally transitioned to state 1.

State 7: Unconditionally transitioned to state 8 which is the set timer state. Green LED for authenticatin is turned on and the red LED is turned off.
If the load_password button is pressed again, game moves to the next state.

State 8: Set reconfig so that timer value can be initialized. If load_password is 1 go to state 9

State 9: If load_password(game begin button) is 1 go to state 10.

State 10: If timeOut is 1, go to State 11 or if load_password = 1, go to Pause.

Pause: if load_password = 1 and watchdogtimer = 0 go to pause 1, else go to state 9.

Pause1: if load_password = 1 and watchdogtimer = 0 go to pause 2 state else go to state 8.

Pause2: Unconditionally go to state 1.

State 11: Set RNG to 1 and load_sig_2 = 0 so that the player can no longer toggle and input their input or use the RNG button. If load_password = 1, go to state 12(game over).

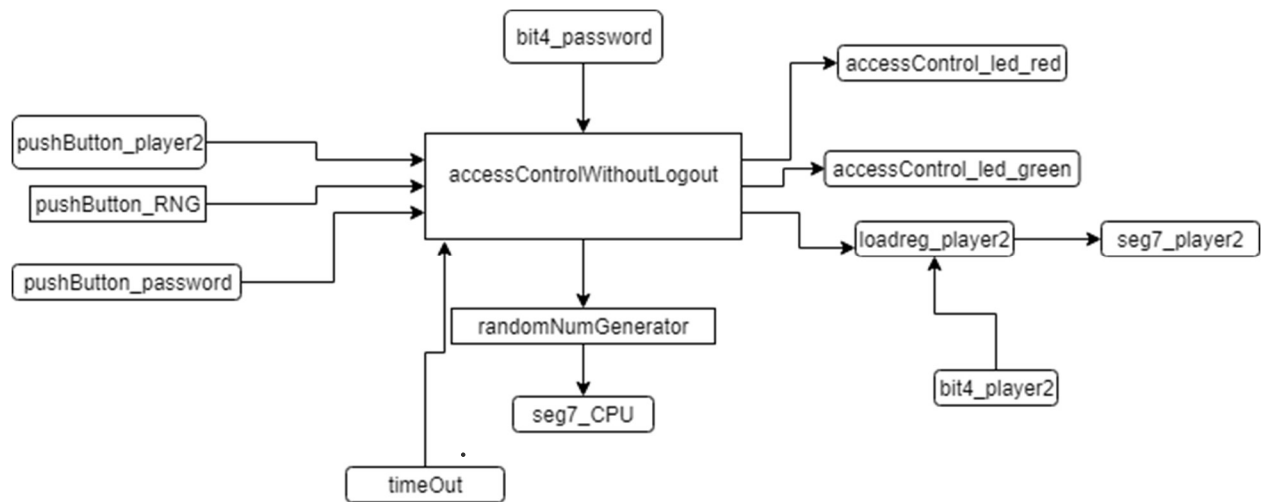State 12: If load_password is 1, go to State 8.

2.2 accessControl design:

Figure 3

3. Random Number Generator

Functionality: A Random number Generator is used to make the player compete with some random value generated by the processor.

This module makes use of a counter which starts counting the moment begin_count signal is set by the RNG module. It counts at every clock cycle and reinitializes when the value is hex F.

The module provides with an Output of Count whenever the button is released.

Since it counts every clock cycle, it is counts upto 50,000,000 times a second and can be called as a true random generator
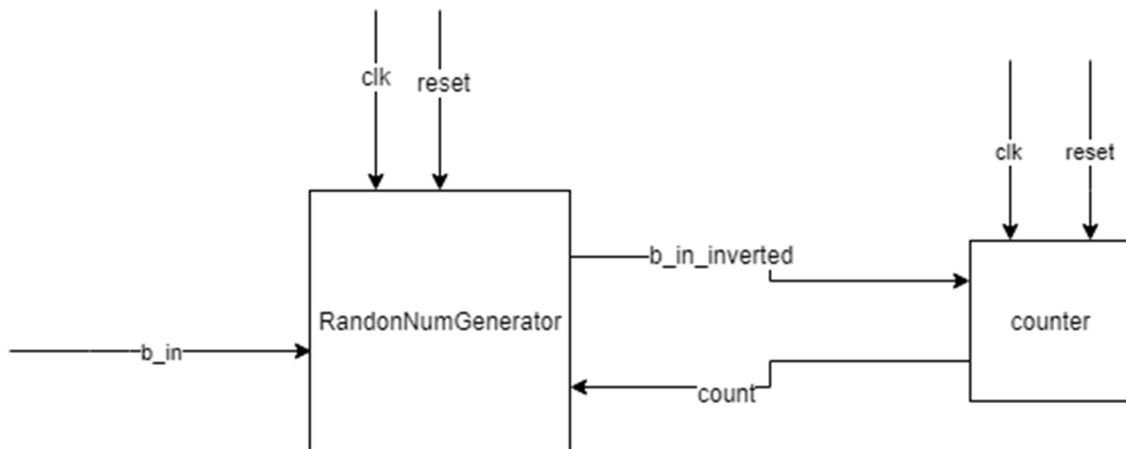

Figure 4

Signals

Input:

B_in: Unshaped button from the Board

B_in_inverted: unverted unshaped button

Output:

Count: counter value that counts upto hex F and reinitializes every clock cycle.
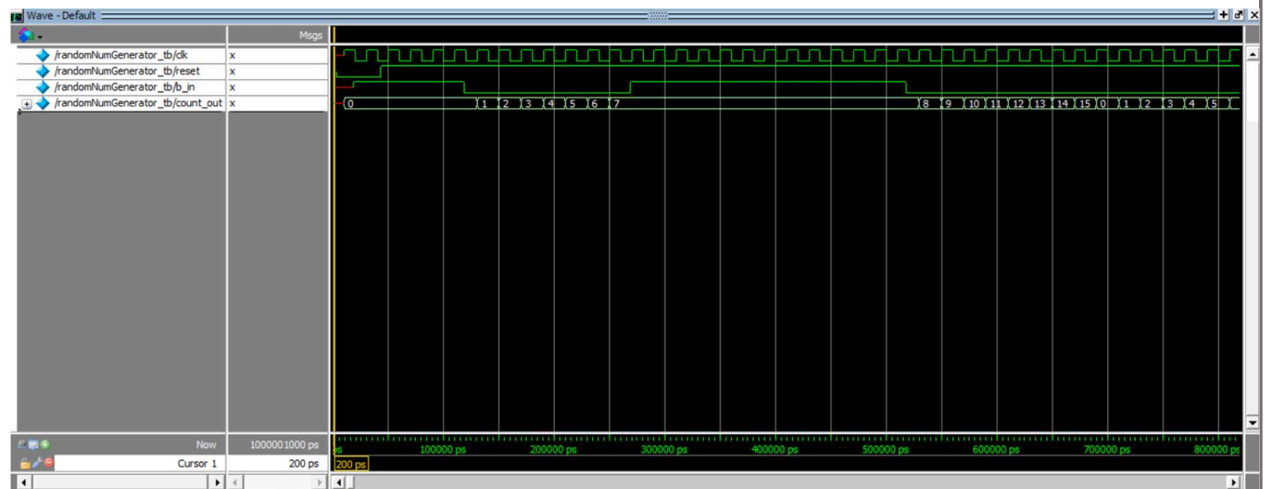
3.1 Simulation

Figure 5

It can be observed from the above simulation that when the button is pressed, the RNG module generates a random value by incrementing the value of the counter every second.

The counter reinitializes at every 15th count since we are counting until hex F. Since the value of the timer increments every second, the user has no control over the count and thus it can be considered as a true random number generator.

4. CountDownToZero:

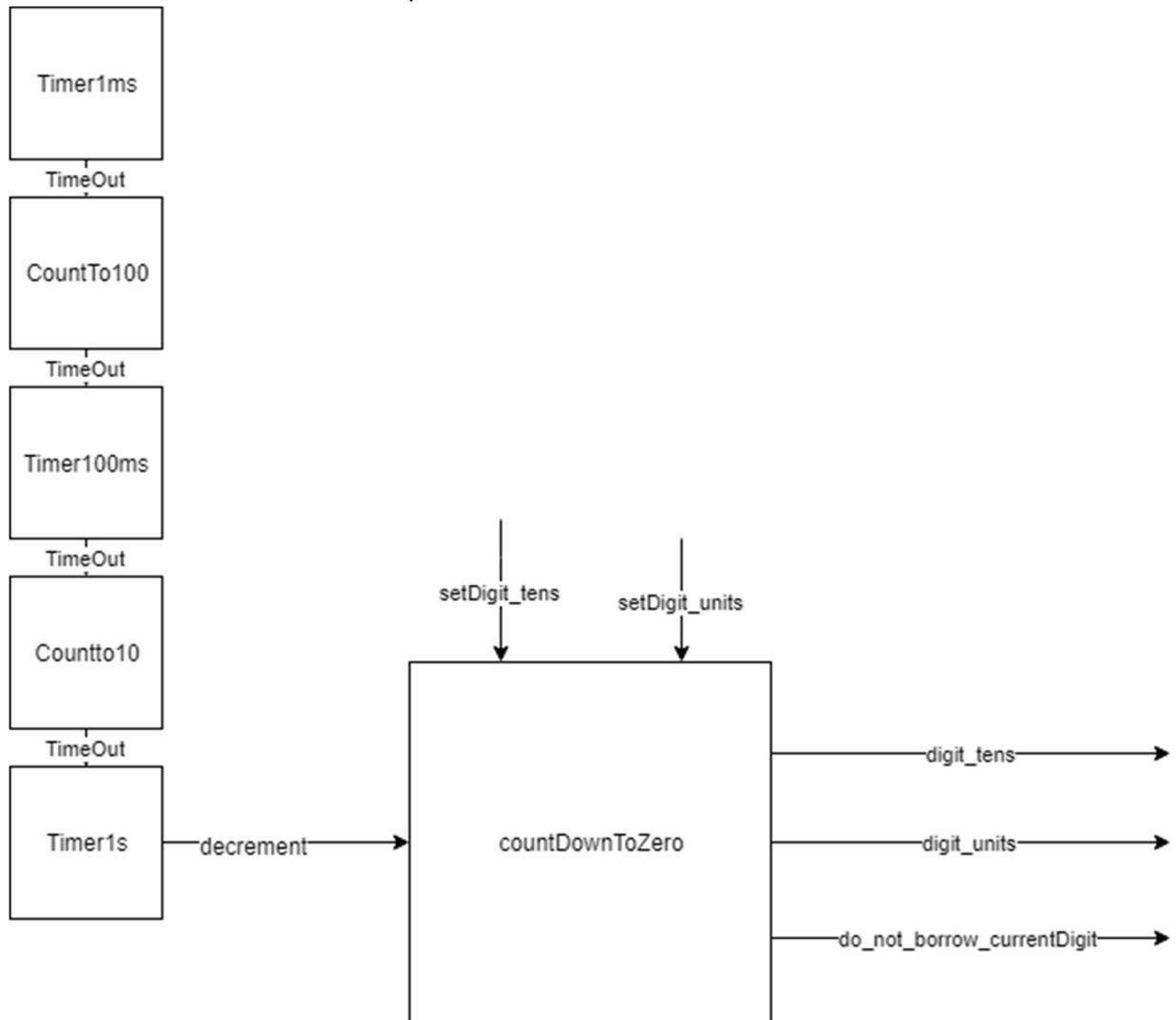This module instantiates a hierarchy of instantiated modules

Figure 6

The first module counts by 1 every cycle in a register and reinitializes the register at every 50,000[th] count and sends out a timeout pulse. Since the clock is 50Mhz, every 50,000[th] count is actually one milli second.

The output is supplied to other module which counts upto 100 every enable signal. This module is initialized by 100msTimer, so it sends a timeOut pulse every 100th millisecond.

The timeOut of Timer100ms is supplied to Count to 10 which sends an enable at every 10th count.

This module is instantiated by Timer1ms so the TimeOut pulse from this module is sent every 1 second. This tomeOut pulse is inputted to a timer module which is instantiated twice in countdownToZero.

The following is the Schematic the timer module instantiated twice by countDownTozero module
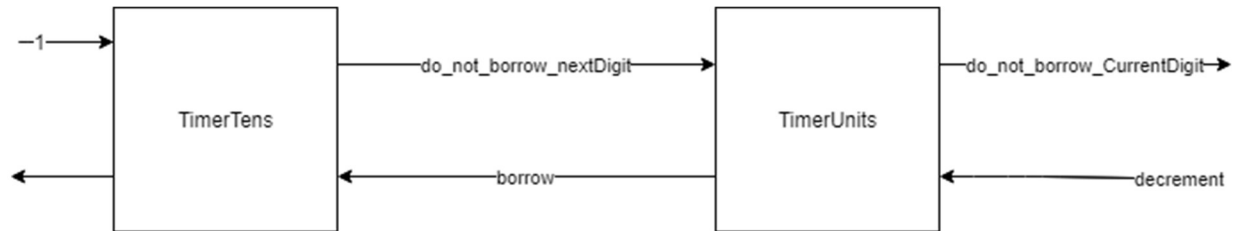
Figure 7

in TimerUnits, Timer1s sends decrement pulse every 1s. From the TimerUnits module, every time the digit reaches a value 0, a borrow signal is sent. The timerTens works on on the same logic. Whenever the value of the timerUnits is 0, the do_not_borrow_nextDigit is checked, if it is 0, a borrow signal is set and the digit in TimerUnits is set 9. Same logic is used in TimerTens with the decrement signal acting as borrow for TimerTens.
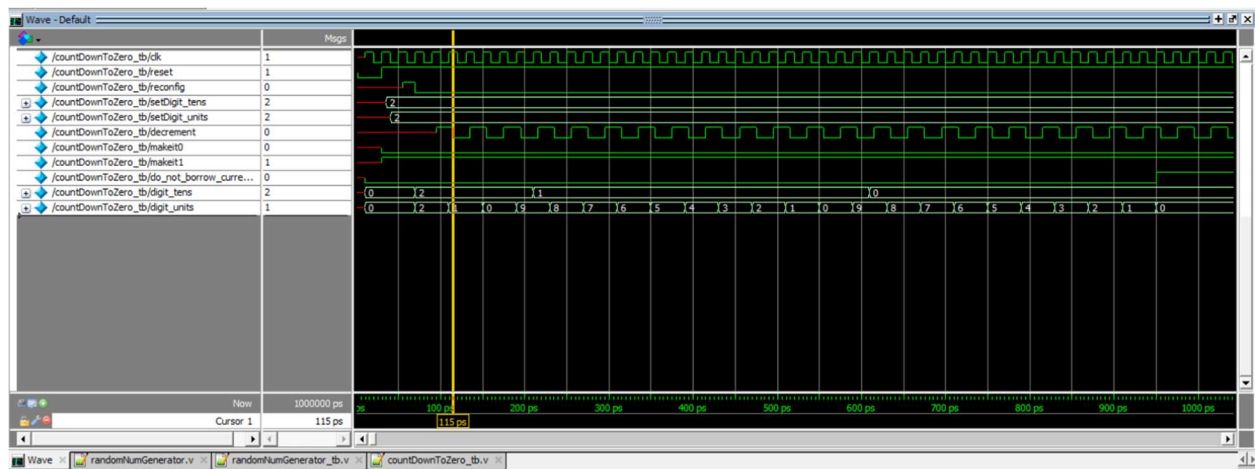
Figure 8

5. Loadreg

Module function: This module has two main functions,
The first function is the reset function, whenever reset is triggered(value is 1), the output of the module should be resetted to a default value of 0. When the value of the reset is 0, The other function comes into action. The other function is load, if the value of load is 0, the output should not change despite of any change in the input. The output of the previous case is stored in the register and it keeps displaying that. When the value of load is 1, the output is set the value of the latest input as the at the rising edge of the clock.

.   5.1 Module design

Figure 9

Figure 10

Figure 11

5.2 Simulation conclusion

we can observe that until reset is 1, the output value is 0, as the reset goes to 0, the next control signal for the output is load signal.

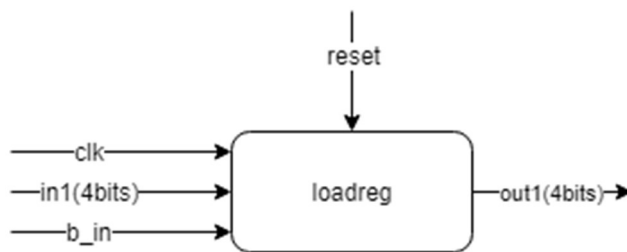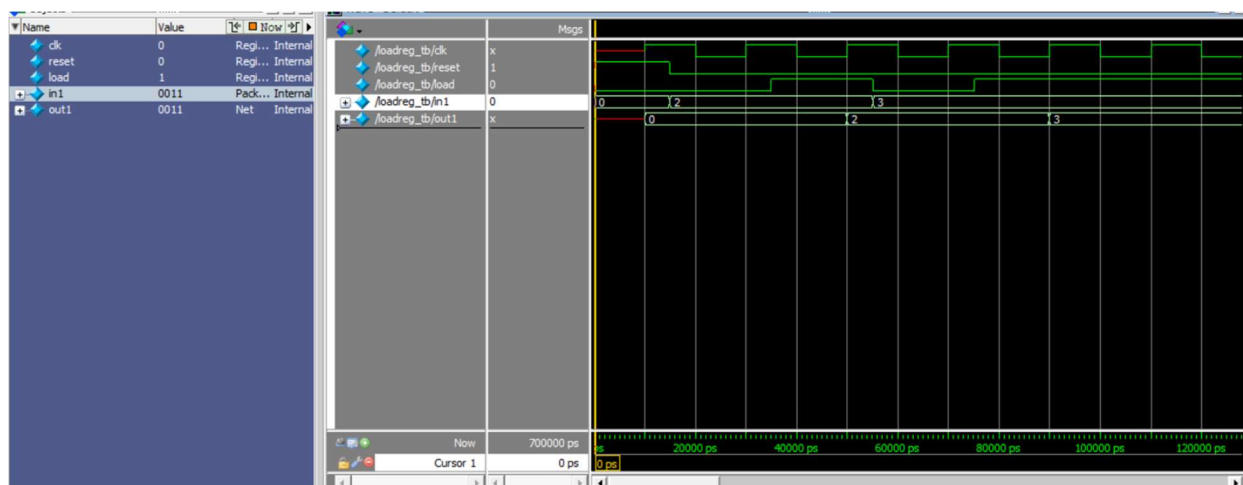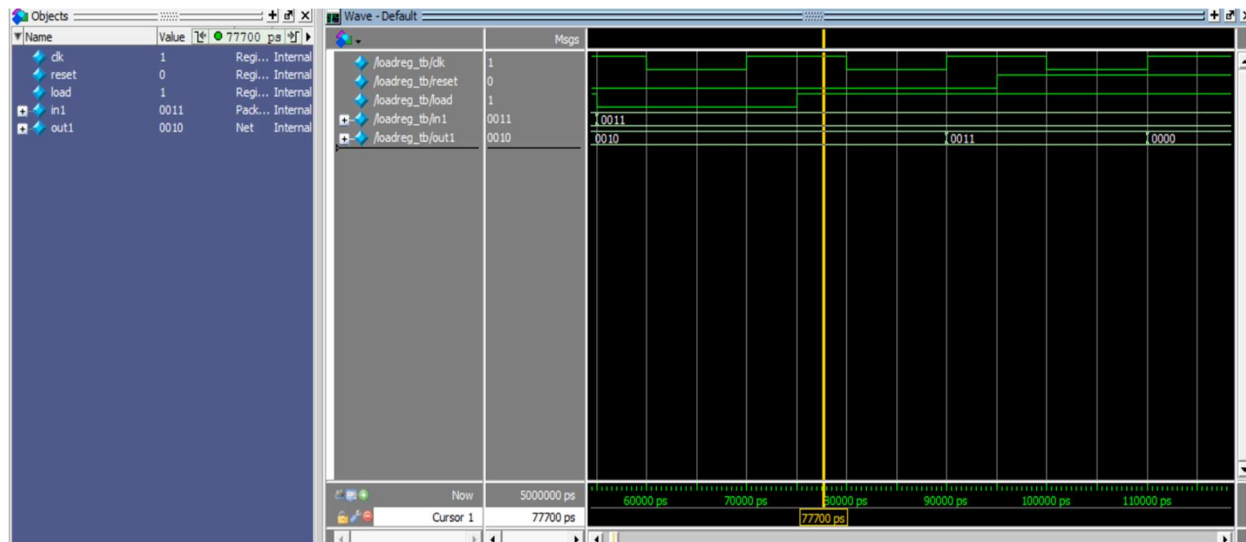It can be seen in the figure 2 that although the value of input is changed, output remains zero until the load is set to 1. When the load is set to 1, the output takes the value of the input at the next rising edge of the clock. So the output becomes 2 at the next rising edge of the clock. Again it is seen that although fourBitIn is set to 3, fourBitOut is not set to 3 until the load is set to 1. After it is set to 1, in the very next cycle's rising edge, fourBitOut is set to 3.

In figure 3 it is observed that again reset is set to 1 and despite of any value of load or fourBitIn, the value of fourBitOut goes to 0 at the very next rising edge of the clock cycle.

6.   Button Shaper
1.   Functionality:

The function of this module singlePulseButton is to generate a single output pulse whenever a button (active low) is pressed once. In a general case, whenever a button is pressed, it generates an output signal for as long as the button is pressed. For a high frequency clock, even if the button is pressed for a second, it generates a high output for around a million clock cycles. As a solution to this problem we implement this module. This kind of module is specifically used in sensitive applications where generating a single pulse is the most important criteria.

To implement this module we make a 3 stage finite state machine.

if B_in ==0   unconditional

State:Init
stay if B_in == 1

State: Pulse
Stay for a single cycle

State: wait
Stay if B_in == 0

if B_in ==1
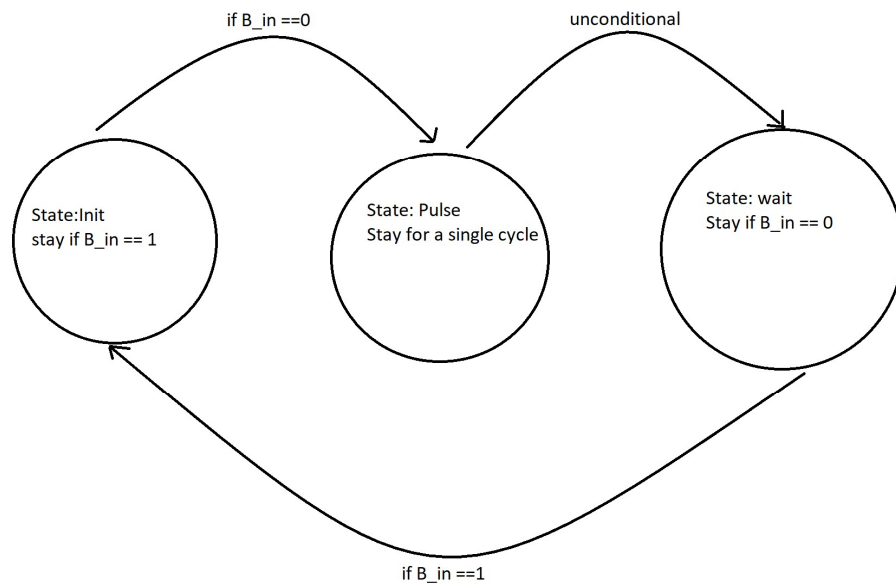
Figure 12

In the above state cycle it can be seen that when the button is not pressed i.e. B=1 (active low), the output remains zero.

As the button is pressed, i.e. b=0, the output s high for a single cycle since all the instructions are executed in a cycle, and then it unconditionally moves to the next state which is the wait state

Now, the output remains zero in the wait state even though the button is continuously pressed. As the user releases the button, i.e. B=1, it goes into initial state until the button is pressed again.

6.2 Button Shaper Design analysis

As seen in figure 2, the module has a single bit input and a clock input. The module can be further classified into two modules. Combinational logic, which uses the inputs and produces the output based on the given specific logic. The other module is a state register which maintains the states of the main module. The different states are s_init, s_pulse, s_wait.

S_init: As given above is the initial state, and every program starts with initial state since it resets before it starts. It stays in the reset state until the button is pressed.

S_pulse: the module stays in this state just for a single cycle and produces an output of 1 and then arrives unconditionally at the wait state.

S_wait. The program stays in this state until the button is continuously pressed and when it is released, the program again arrives in the initial state.

Figure 13

Signals:

1. B_in: 1 bit input signal indicating the status of the button.
2. State/ nextState: 2 bit registers, storing the value of the defined parameters, s_init = 0, s_pulse = 1, s_wait = 2
3. Pulse: single bit output.
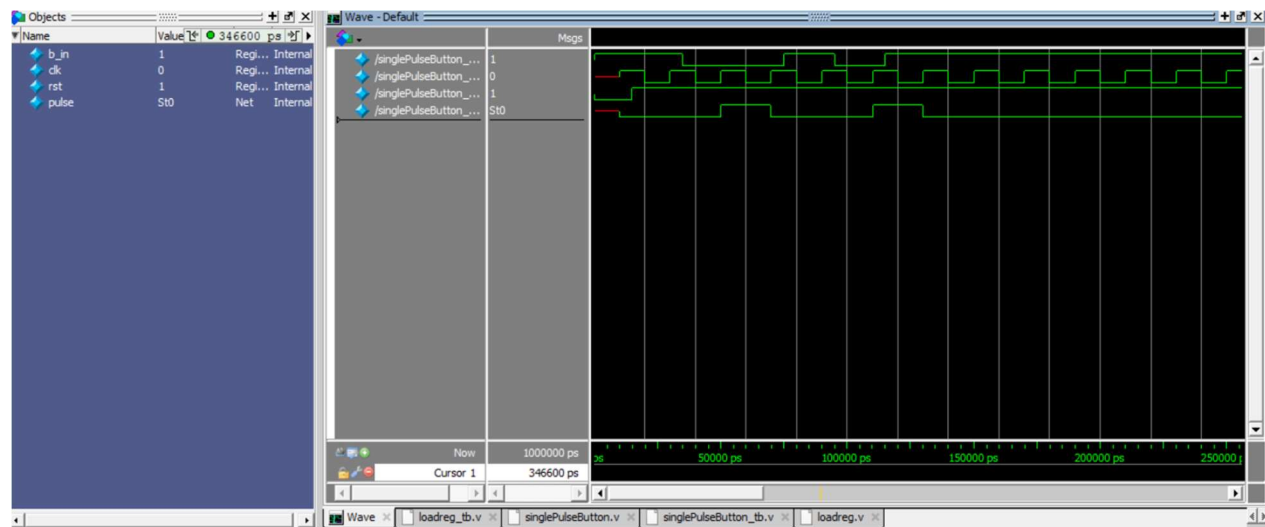
Figure 14

As seen in figure 3, initially the reset signal (rst) is low, and the system is resetted and the program is in initial state. The reset is released but the program stays in the initial state until the button is pressed. As the button is pressed, in the next rising edge of the cycle, pulse is 1 just for a single cycle.

The pulse gets low in the next cycle and the program is now in the wait state until the button is released and then the program goes into initial state again.

Simulation Conclusion:

It can be observed from the above figures that only when the button is pressed, we get a single cycle pulse, no matter how long the button is pressed again, the output remains zero after a single pulse. In order to generate another pulse, one has to press the button again.

7. Seven Segment Display
   7.1 Functionality:
   i. sevenSegDisplayDecoder: The function of this module is to convert the 4 bit hexadecimal input to a seven bit signal that corresponds to different segments of the display in such a way that the signal leads to the readable character in hexadecimal which is same as the 4 bit hexadecimal that is inputted.
   So each of the 4 bit hexadecimal character 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F corresponds to 7 bit output signal displaying the same redable hexadecimal character.

   ii. sevenSegDisplayDecoder.tlb: This module is a testbench to test the above module's working. In this module we store each of the 4 bit hexadecimal character in the input register and feed it as an input to the sevenSegDisplayDecoder module with a delay of 10 seconds to obtain the corresponding 7 bit output signal.

7.2 Block Diagram

Figure 15

7.3 Simulation

Figure 16

Figure 17
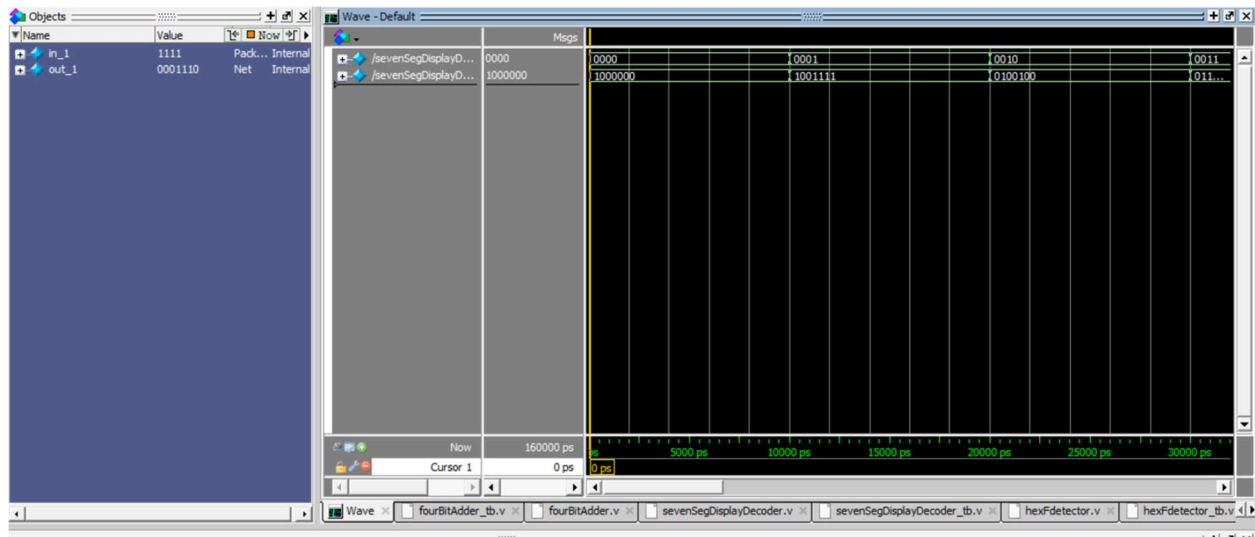
Figure 18
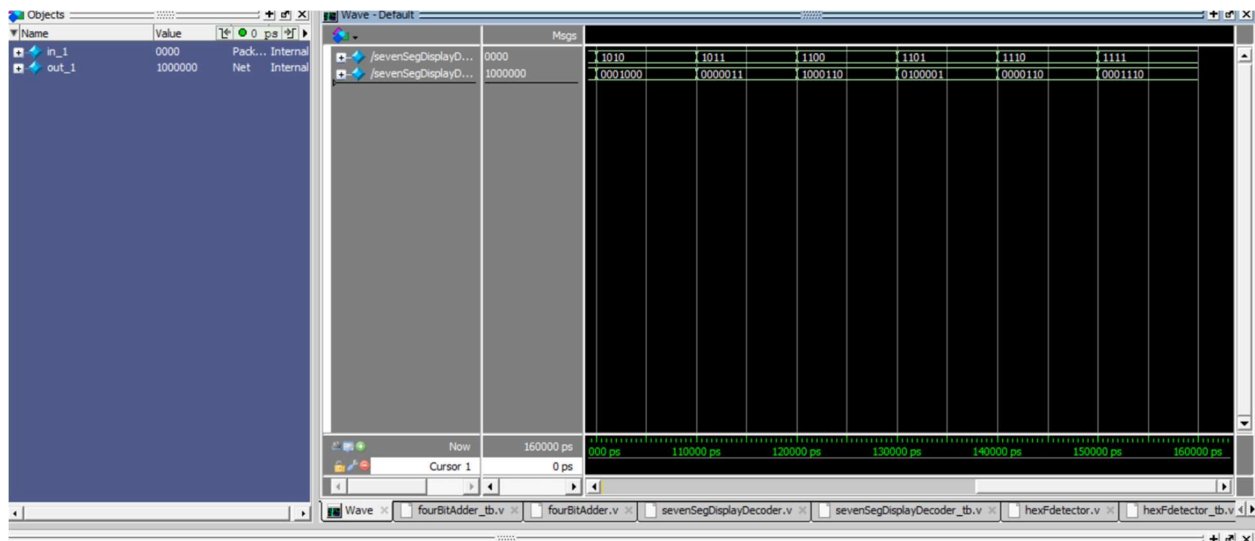
Figure 19

7.4 Result : As seen in figure 2,3,4,5 for each 4 bit hexadecimal input, we get a 7 bit output signal which generates readable same hexadecimal character in the 7 segment display. Note that the signal to turn on a segment is '0' and the signal required to turn off a segment is '1' due to current sinking property.
Each hexadecimal has it's unique set of 7 bits output signal.

8. adder

This module is a simple combinational module that accepts two 4 bit input and produces a sum of 4 bit output.

The max number that the adder can sum upto is a hex F, since the output is only 4 bits.
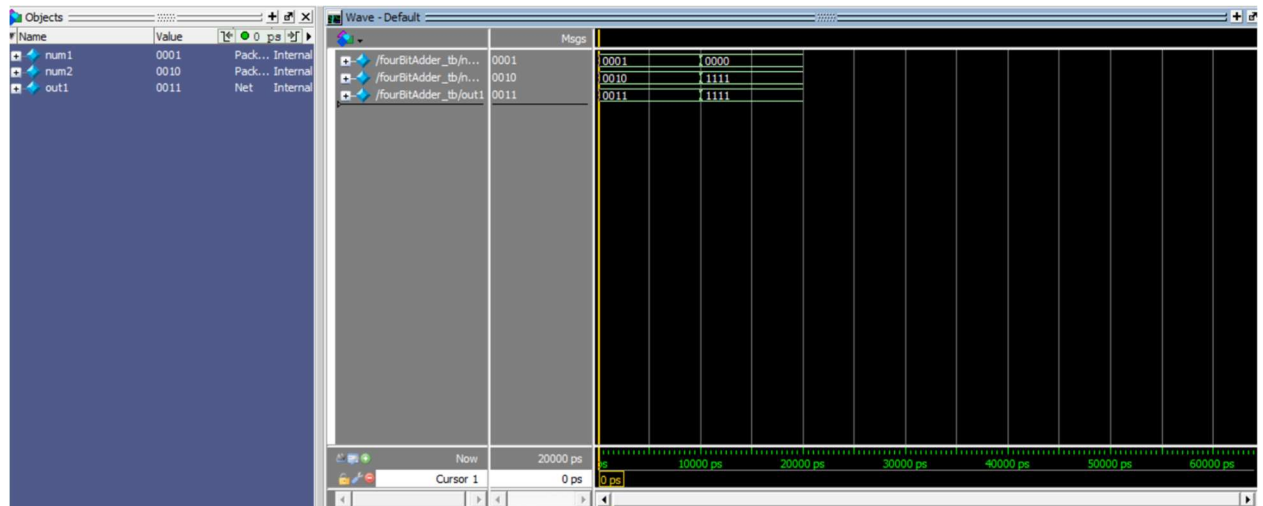
Figure 20

9. Video Demonstration

In the video, user authentication is demonstrated followed by examining edge cases. This is followed by testing of Pause feature, Resume feature, Timer reset feature, and Log out state. It also demonstrates working of RNG. This is followed by the timeout test where once the timer value is 0, player cannot access the game.

https://drive.google.com/file/d/1wHnB1C-RH695On3yr2_KbWsAksJ2e5Bg/view?usp=sharing

The second feature tests RNG and player load button, it also tests win detection which are the main feature of the game.

https://drive.google.com/file/d/1d704rGhWAFcy7wKIJbCNiBoDjpzsAyVL/view?usp=sharing

10. Conclusion

All the modules above have been tested individually and incorporated in the project and tested for their functionality in the project. All the functionality and the edge cases have been tested and works as expected in the project

Appendix

```
//4014
//testLED1, testLED2, testLED3, testLED4, testLED5 are just to test the states


module accessControlWithLogout (clk, reset, load_sig_RNG, load_sig_2,  load_password,
pushButtonLoad_RNG, pushButtonLoad2, passinput4bit, timeout, reconfig, enable, passredled,
passgreenled, testLED1, testLED2, testLED3, testLED4, testLED5);

input clk, reset;
input load_password;     //signal from button shaper
input [3:0] passinput4bit;
input pushButtonLoad_RNG, pushButtonLoad2 ;   //signal from button shaper
input timeout;

reg flag;
reg [3:0] state;
output load_sig_RNG, load_sig_2; //signal to the load register
output passredled, passgreenled;
output reconfig, enable;
reg reconfig, enable;
reg passredled, passgreenled;
reg load_sig_RNG, load_sig_2 ;
output reg testLED1, testLED2, testLED3, testLED4, testLED5;

parameter s1 = 0, s2 = 1, s3 = 2, s4 = 3, s5 = 4, s6 = 5, s7 = 6, s8 = 7, s9 = 8, s10 = 9, s11 = 10, s12
= 11;

always @ (posedge clk)

 begin
```

```verilog
                    if (reset == 0)
                     begin
                      flag <=1;
                      load_sig_RNG <= 1;
                      load_sig_2 <= 0;
                      passredled <= 1;
                      passgreenled <= 0;
                      state <= s1;
                     end
                    else
                     begin
                      case (state)

                       s1:
                        begin
                         if (load_password == 1)
                          begin
                           if ( passinput4bit == 4'b0100)
                                begin
                                 state <= s2;
                                 end
                                else
                                 begin
                                  flag <= 0;
                                  state <= s2;
                                 end
                           end
                                        else
                                 begin
                                  state <= s1;
                                 end
                          end

                       s2:
                        begin
                         if (load_password == 1)
                          begin
                           if ( passinput4bit == 4'b0000)
                                begin
                                 state <= s3;
                                 end
                                else
                                 begin
                                  flag <= 0;
```

```verilog
                    state <= s3;
                    end
                end
                    else
                      begin
                       state <= s2;
                      end
                end

            s3:
             begin
              if (load_password == 1)
               begin
                if ( passinput4bit == 4'b0001)
                    begin
                     state <= s4;
                     end
                    else
                     begin
                      flag <= 0;
                      state <= s4;
                     end
                end
                    else
                     begin
                      state <= s3;
                     end
                end

            s4:
             begin
              if (load_password == 1)
               begin
                if ( passinput4bit == 4'b0100)
                    begin
                     state <= s5;
                     end
                    else
                     begin
                      flag <= 0;
                      state <= s5;
                     end
                end
                    else
```

```verilog
                     begin
                      state <= s4;
                     end
                 end


             s5:
              begin
               if (flag == 0)              //password incorrect
                begin
                     state <= s6;
                end
                 else
                     begin
                      state <= s7;            //password correct
                     end
                 end


             s6:
              begin
                 flag <= 1;
                     state <= s1;
              end




             s7:
              begin
              passgreenled <= 1;
              passredled <= 0;
                     state <= s8;
              end

             s8:
              begin
                     testLED1 <= 1;
                      testLED5 <= 0;
                     if(load_password == 0)
                             begin
                                     state <= s8;
                                     reconfig <= 0;
                             end
                     else
                             begin
```

```verilog
                                reconfig <= 1;
                                state <= s9;
                        end
        end

        s9:
        begin
                testLED1 <= 0; testLED2 <= 1;
                reconfig <= 0;
                if(load_password == 0)
                        begin
                                state <= s9;
                        end
                else
                        begin
                                state <= s10;
                        end
        end

        s10:
        begin
                testLED2 <= 0; testLED3 <= 1;
                load_sig_RNG <= pushButtonLoad_RNG;                //the push button
from rng is equal to count begin
                load_sig_2 <= pushButtonLoad2;
                enable <= 1;
                if(timeout == 1)
                        begin
                                state <= s11;
                        end
                else
                        begin
                                state <= s10;
                        end
        end

        s11:
        begin
                testLED3 <= 0; testLED4 <= 1;
                enable <= 0;
                load_sig_RNG <= 1;
                load_sig_2 <= 0;
                if (load_password <= 0)
                        begin
```

```verilog
                                    state <= s11;
                            end
                    else
                            begin
                                    state <= s12;
                            end
            end

            s12:
            begin
                    testLED4 <= 0; testLED5 <= 1;
                    if (load_password <= 0)
                            begin
                                    state <= s12;
                            end
                    else
                            begin
                                    state <= s8;
                            end
            end
        default:
          begin
           state <= s1;
          end
     endcase
    end
 end
endmodule

//4014
module countDownToZero(clk, reset, reconfig, setDigit_tens, setDigit_units, decrement,
do_not_borrow_currentDigit, digit_tens, digit_units);//, makeit0, makeit1);
input clk, reset;
input reconfig;
input decrement;
input [3:0] setDigit_units;
input [3:0] setDigit_tens;
//input makeit0, makeit1;
output do_not_borrow_currentDigit;
output [3:0] digit_units;
output [3:0] digit_tens;
wire borrow;
wire do_not_borrow;
//parameter makeit0 = 0, makeit1 = 1;
```

```verilog
        wire makeit1 ;
        assign makeit1 = 1'b1;
        wire makeit0;
        timer TimerTens(clk, reset, reconfig, setDigit_tens, borrow, makeit1, do_not_borrow, makeit0,
        digit_tens);
        timer TimerUnits(clk, reset, reconfig, setDigit_units, decrement, do_not_borrow,
        do_not_borrow_currentDigit, borrow, digit_units);

        endmodule
        //Author 4014
        //counter module for rng


        module counter( clk, reset, begin_count, count);
        input clk, reset;
        input begin_count;
        output [3:0] count;
        reg [3:0] count;



        always @ (posedge clk)
        begin
                if (reset == 0)
                        begin
                                count <= 0;
                        end

                else if((count == 15) | (count > 15))
                        begin
                                count <= 0;
                        end
                else
                        begin
                                begin
                                        if (begin_count == 1 )
                                                begin
                                                        count <= count + 1;
                                                end
                                end
                        end
        end
        endmodule
```

```verilog
//Author 4014
//count to 10 module for 1ms module

module CountTo10(clk, reset, count, timeOut);
input clk, reset;
input count;
output timeOut;
reg timeOut;
reg [0:3] counter;
always @ (posedge clk)
begin

        if (reset == 0)
        begin
                timeOut <= 0;
                counter <= 0;
        end

        else
        begin
                if(count == 0)
                        begin
                                counter <= counter;
                                timeOut <= 0;
                        end
                else
                        begin
                        if(counter >10)
                                begin
                                        counter<=0;
                                end
                        else if(counter == 10)
                                begin
                                        counter <= 0;
                                        timeOut <= 1;
                                end

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

```
                    end
                end
    end
    endmodule



    //Author 4014
    //Count to 100 for 1 ms module
    module CountTo100(clk, reset, count, timeOut);
    input clk, reset;
    input count;
    output timeOut;
    reg timeOut;
    reg [0:6] counter;
    always @ (posedge clk)
    begin

            if (reset == 0)
            begin
                    timeOut <= 0;
                    counter <= 0;
            end

            else
            begin
                    if(count == 0)
                        begin
                                counter <= counter;
                                timeOut <= 0;
                        end
                    else
                        begin
                        if(counter >100)
                                begin
                                        counter<=0;
                                end
                        else if(counter == 100)
                                begin
                                        counter <= 0;
                                        timeOut <= 1;
                                end
                        else
                                begin
```

```
                                                timeOut <= 0;
                                                counter <= counter + 1;
                                    end
                        end
            end
end
endmodule




// 6370
// Homework 2
// Author: Raj Pramod Dasadia, 4014
// Adds two 4 bit numbers
module fourBitAdder (in1, in2, sum);

input [3:0] in1, in2;
output [3:0] sum;
reg [3:0] sum;
        always @ (in1, in2)
         begin

            sum = in1 + in2;

        end
endmodule




// 6370
// Homework 2
// Author: Raj Pramod Dasadia, 4014
// detects if the input is a binary 1111
module hexFdetector (hex_in, out_sigRed, out_sigGreen);

input [3:0] hex_in;
output out_sigRed, out_sigGreen;

reg out_sigRed, out_sigGreen;

        always @ (hex_in)
                if (hex_in == 4'b1111)
                  begin
                        out_sigGreen = 1; out_sigRed = 0;
```

```
                        end
                    else
                      begin
                             out_sigRed = 1; out_sigGreen = 0;
                      end

        endmodule




        // Lab 6
        // Author: Raj Pramod Dasadia, 4014
        // Top module to connect all the modules


        module LAB3_DASADIA_R ( clk, reset, bit4_player2, bit4_password, bit4_setDigit_units,
        bit4_setDigit_tens, pushButton_RNG, pushButton_player2, pushButton_password,
                          seg7_CPU, seg7_player2, seg7_sum,seg7_units, seg7_tens,
        hexFdetection_led_red, hexFdetection_led_green, accessControl_led_red,
        accessControl_led_green, testLED1, testLED2, testLED3, testLED4, testLED5);


        input clk, reset;

        input pushButton_RNG, pushButton_player2, pushButton_password;

        input [3:0] bit4_player2, bit4_password;
        input [3:0] bit4_setDigit_tens, bit4_setDigit_units;

        output hexFdetection_led_red, hexFdetection_led_green, accessControl_led_red,
        accessControl_led_green;

        output [6:0] seg7_CPU, seg7_player2, seg7_sum, seg7_units, seg7_tens;


        wire load_sig_player1, load_sig_player2, pulse_player2, pulse_password;
        wire [3:0] fourBitOut_player2, sum4bit, digit_tens, digit_units, count ;
        wire load_sig_RNG, reconfig, decrement, enable, timeout;


        output testLED1, testLED2, testLED3, testLED4, testLED5;
```

```
accessControlWithLogout accessControlWithLogout1 (clk, reset, load_sig_RNG,
load_sig_player2,  pulse_password, pushButton_RNG, pulse_player2, bit4_password, timeout,
reconfig, enable, accessControl_led_red, accessControl_led_green, testLED1, testLED2,
testLED3, testLED4, testLED5);

randomNumGenerator randomNumGenerator1 (clk, reset, load_sig_RNG, count);

fourBitAdder adder (count, fourBitOut_player2, sum4bit);

hexFdetector win_detector (sum4bit, hexFdetection_led_red, hexFdetection_led_green);


loadreg loadreg_player2 ( clk, reset, load_sig_player2, bit4_player2, fourBitOut_player2 );

sevenSegDisplayDecoder sevenSegDisplayDecoder_CPU (count, seg7_CPU);
sevenSegDisplayDecoder sevenSegDisplayDecoder_player2 (fourBitOut_player2, seg7_player2);
sevenSegDisplayDecoder sevenSegDisplayDecoder_sum (sum4bit, seg7_sum);
sevenSegDisplayDecoder sevenSegDisplayDecoder_tens (digit_tens, seg7_tens);
sevenSegDisplayDecoder sevenSegDisplayDecoder_units (digit_units, seg7_units);


singlePulseButton singlePulseButton_player2 (pushButton_player2, pulse_player2, clk, reset);
singlePulseButton singlePulseButton_password (pushButton_password, pulse_password, clk,
reset);

countDownToZero countDownToZero1(clk, reset, reconfig, bit4_setDigit_tens,
bit4_setDigit_units, decrement, timeout, digit_tens, digit_units);

TimerOneSecond TimerOneSecond1(clk, reset, enable, decrement);




Endmodule




// 6370
// Homework 4
// Author: Raj Pramod Dasadia, 4014
// load register with reset and clock trigger

module loadreg ( clk, reset, load, fourBitIn, fourBitOut );
```

```
input clk, reset, load;
input [3:0] fourBitIn;
output [3:0] fourBitOut;
reg [3:0] fourBitOut;

always @ (posedge clk)

begin
        if ( reset == 1)
         begin
          if ( load == 1)
            begin
             fourBitOut = fourBitIn;
            end
          end
         else if (reset == 0)
          begin
           fourBitOut = 4'b0000;
          end
end
endmodule




//Author 4014
//rng module instantiating counter module
module randomNumGenerator (clk, reset, b_in, count);
input clk, reset;
input b_in;
output [3:0] count;
assign b_in_inverted = ~b_in;

counter rng_counter (clk, reset, b_in_inverted, count);

endmodule




// 6370
// Homework 2
```

```verilog
// Author: Raj Pramod Dasadia, 4014
// Displays 4 bit binary into 7 segment display device
module sevenSegDisplayDecoder (fourbit_in, sevenbit_out);

input [3:0] fourbit_in;
output [6:0] sevenbit_out;
reg [6:0] sevenbit_out;

always @ (fourbit_in)
    begin
        case (fourbit_in)
            4'b0000:
            begin
                sevenbit_out = 7'b1000000;
            end
            4'b0001:
            begin
                sevenbit_out = 7'b1001111;
            end
            4'b0010:
            begin
                sevenbit_out = 7'b0100100;
            end
            4'b0011:
            begin
                sevenbit_out = 7'b0110000;
            end
            4'b0100:
            begin
                sevenbit_out = 7'b0011001;
            end
            4'b0101:
            begin
                sevenbit_out = 7'b0010010;
            end
            4'b0110:
            begin
                sevenbit_out = 7'b0000010;
            end
            4'b0111:
            begin
                sevenbit_out = 7'b1111000;
            end
            4'b1000:
```

```verilog
                begin
                        sevenbit_out = 7'b0000000;
                end
                4'b1001:
                begin
                        sevenbit_out = 7'b0010000;
                end
                4'b1010:
                begin
                        sevenbit_out = 7'b0001000;
                end
                4'b1011:
                begin
                        sevenbit_out = 7'b0000011;
                end
                4'b1100:
                begin
                        sevenbit_out = 7'b1000110;
                end
                4'b1101:
                begin
                        sevenbit_out = 7'b0100001;
                end
                4'b1110:
                begin
                        sevenbit_out = 7'b0000110;
                end
                4'b1111:
                begin
                        sevenbit_out = 7'b0001110;
                end
                default:
                begin
                        sevenbit_out = 7'b0000000;
                end
            endcase
        end
    endmodule


    //Author 4014
```

```verilog
//Button Shaper module
module singlePulseButton (b_in, pulse, clk, rst);

input b_in;
input clk, rst;

output pulse;

reg pulse;
reg [1:0] state, nextState;

parameter s_init = 0, s_pulse = 1, s_wait = 2;

always @ ( state, b_in)

        begin

        case (state)


          s_init:
           begin
            pulse = 0;
                  if (b_in == 1)
                   begin
                    nextState = s_init;
                   end
                  else
                   begin
                    nextState = s_pulse;
                   end
            end

          s_pulse:
           begin
             pulse = 1;
             nextState = s_wait;
           end

          s_wait:
            begin
                  pulse = 0;
                    if (b_in == 0)
                     begin
```

```
                    nextState = s_wait;
                  end
                else
                  begin
                    nextState = s_init;
                  end
              end

          default:
            begin
              nextState = s_init;
            end
        endcase
      end


  always @ (posedge clk)
      begin
        if (rst == 0)
          begin
            state <= s_init;
          end
        else
          begin
            state <= nextState;
          end
      end

endmodule




//Author 4014
//Digit clock module
module timer(clk, reset, reconfig, setDigit, decrement, do_not_borrow_nextDigit,
do_not_borrow_currentDigit, borrow, digit);
input clk, reset;
input reconfig;
input [3:0] setDigit;
input decrement;
input do_not_borrow_nextDigit;
output reg do_not_borrow_currentDigit;
output reg borrow;
```

```verilog
output reg [3:0] digit;

always @ (posedge clk)
        begin
                if(reset == 0)
                        begin
                                do_not_borrow_currentDigit <= 0;
                                borrow <= 0;
                                digit <= 4'b0000;
                        end
                else
                        begin
                                if(reconfig == 1)
                                        begin
                                                do_not_borrow_currentDigit <= 0;
                                                if(setDigit > 9)
                                                        begin
                                                                digit <= 4'b1001;
                                                        end
                                                else
                                                        begin
                                                                digit <= setDigit;
                                                        end
                                        end
                                else
                                        begin
                                                if(decrement == 1)
                                                        begin
                                                                if(digit == 4'b0000)
                                                                        begin

        if(do_not_borrow_nextDigit == 1)

                                                                                begin

        //digit <= 4'b0000;

        do_not_borrow_currentDigit <= 1;

                                                                                end
                                                                        else
                                                                        begin

        borrow <= 1;

        digit <= 4'b1001;
```

```
                                                        end
                                    end
                else
                    begin
                            borrow <= 0;
                            digit <= digit -
1;

                            if(digit == 1)
                                begin

            if(do_not_borrow_nextDigit == 1)

            begin

                    do_not_borrow_currentDigit <= 1;

            end
                                                        end

                                    end
                    end
                else
                    begin
                            digit <= digit;
                            borrow <= 0;
                    end
                        end
                    end
            end
    endmodule



    //Author 4014
    //1ms module counting upto 50,000
    module Timer1ms(clk, reset, enable, timeOut);
    input clk, reset;
    input enable;
    output timeOut;
    reg timeOut;
    reg [15:0] counter;
```

```verilog
always @ (posedge clk)
begin

        if (reset == 0)
        begin
                timeOut <= 0;
                counter <= 0;
        end

        else
        begin
                if(enable == 0)
                        begin
                                counter <=0;
                                timeOut <=0;
                        end

                else
                        begin
                                if (counter >50000)
                                        begin
                                                counter<=0;
                                        end
                                else if(counter == 50000)
                                        begin
                                                counter <= 0;
                                                timeOut <= 1;
                                        end

                                else
                                        begin
                                                timeOut <= 0;
                                                counter <= counter + 1;
                                        end
                        end
                end
        end
end
endmodule
```

```verilog
module Timer100ms(clk, reset, enable, timeOut);
input clk, reset;
input enable;
output timeOut;
//reg timeOut;
wire count;

Timer1ms Timer1ms_1(clk, reset, enable, count);
CountTo100 CountTo100_1(clk, reset, count, timeOut);

endmodule
```

```verilog
//Author 4014
//timer module instantiating digit counter twice for tens and hundred's place of counter
module TimerOneSecond(clk, reset, enable, timeOut);
input clk, reset;
input enable;
output timeOut;
//reg timeOut;
wire count1;

Timer100ms Timer100ms_2(clk, reset, enable, count1);
CountTo10 CountTo10_2(clk, reset, count1, timeOut);

endmodule
```

End of Design Document