Lab4-Bonus (Timer)

Lab 4: Timer, Hardware-based True Random Number Generator, ROM-based Game Access Control on FPGA

Sohail Shareef Mohammed

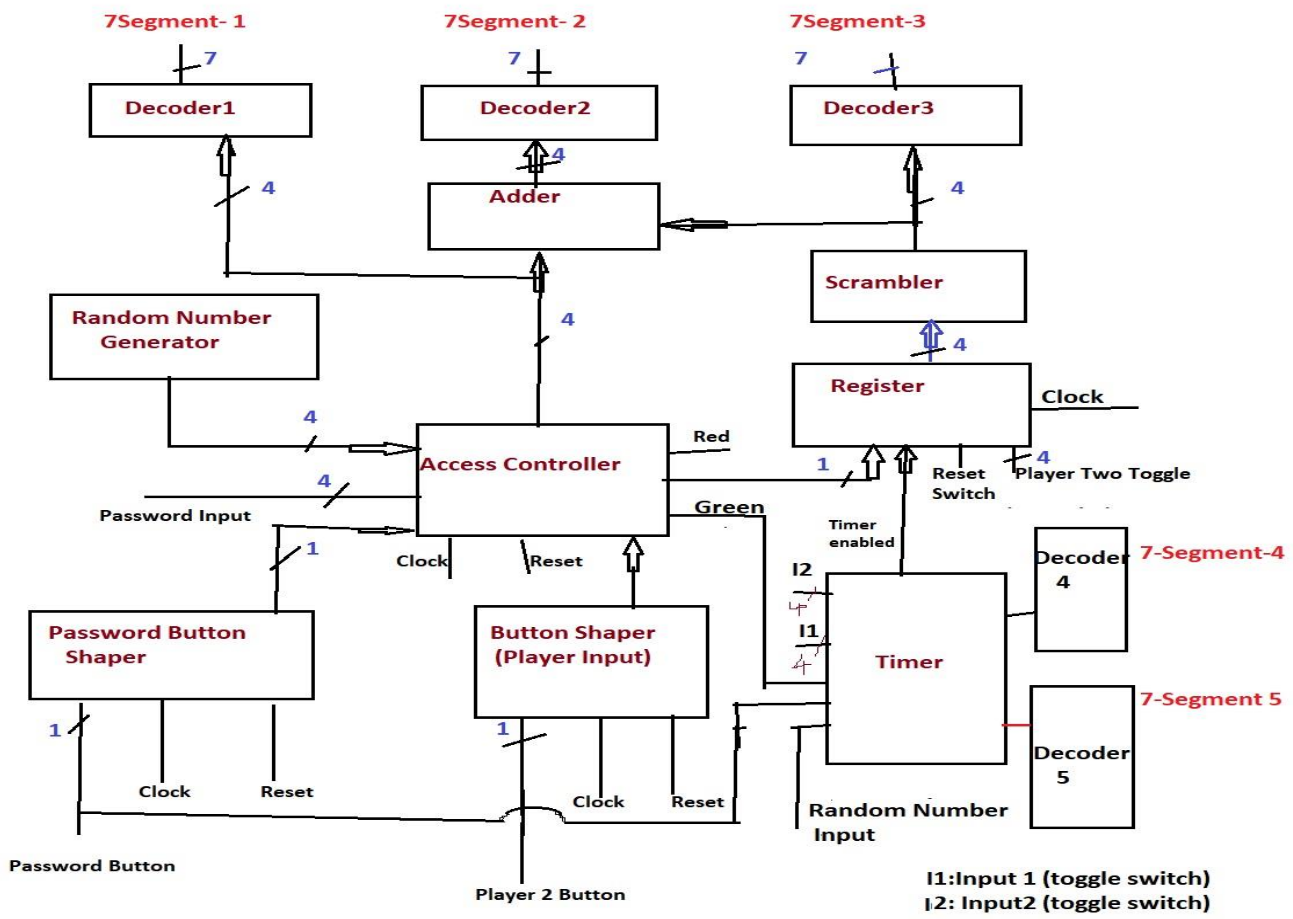
ECE6370

1. How it Works?

“Lab 4 Bonus” project introduces many changes to the previous Lab 4 project. Here we have implemented reconfigurable timer to the game using two 7-segment display devices, where each digit is updated with 4 toggle switches. The timer can be set after the board is authenticated and that is triggered to reload by using the password entering button. The timer is a BCD timer and any value over 9 will be loaded to the timer as 9. The maximum value of the timer is 99 seconds. The timer starts to count down after the random number generator button is pushed. When the timer value reaches 0, the inputs from Player 2 are no longer taken. To play the next round, the timer reload button should be pushed before the random number generator button is pushed.

2. Block Diagram

Overall Architecture of the Game :

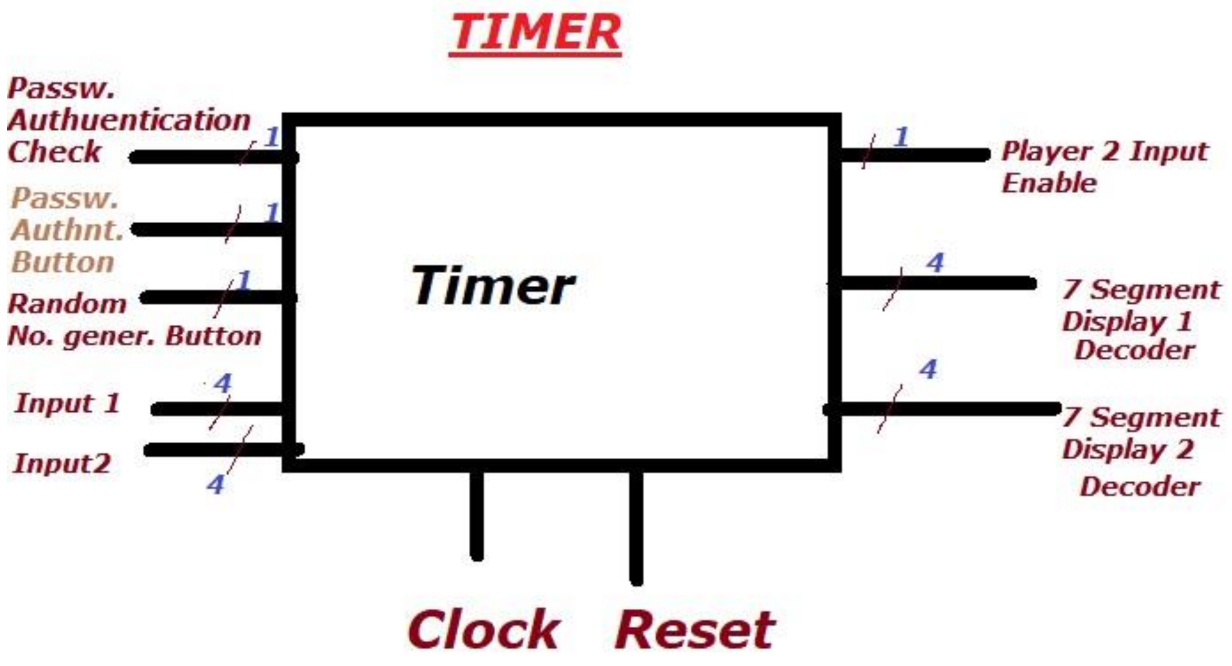


Timer, Hardware-based True Random Number Generator, ROM-based Game Access Control on FPGA

From the above Diagram it clearly Shows the flow and design of the architecture.

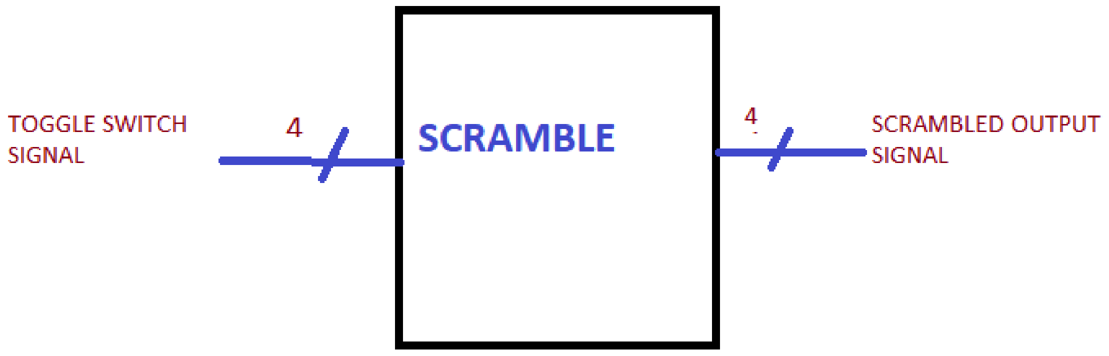
3. Verilog Modules (ModelSim)

3.0 Timer Module



This module count the number which user set till 0 .In this module first it check if pasw. is a authenticated and then on pressing the password authenticating button it will reload the toggle switch value on to counter. Once random number is pressed then it automatically will start to count. This can be reflected on two 7 segment display through decoders.

3.1 Scrambler Module

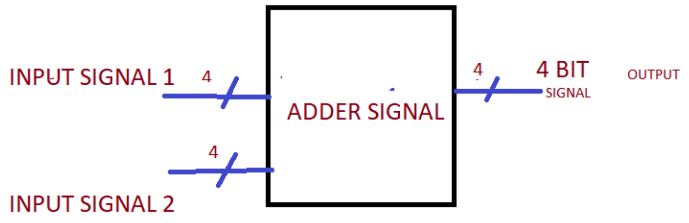


In this module we map every number between 0 and 15 to different number from 0-15 such that no two number has same number mapped. This module record all the mapping and it provide corresponding scrambled number of the requested number.

Mapping Index card for this particular example is as follows:

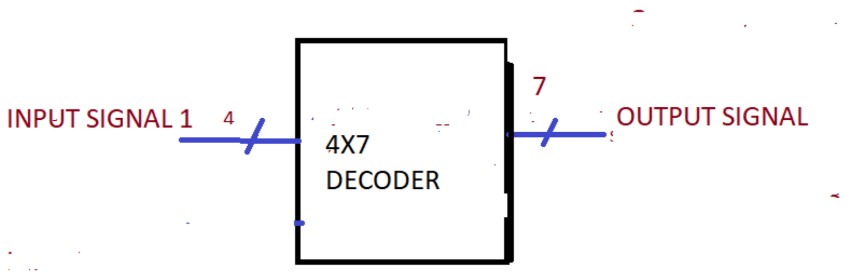
Unscrambled		Scrambled
0	→	D
1	→	7
2	→	4
3	→	0
4	→	F
5	→	8
6	→	1
7	→	C
8	→	6
9	→	E
A	→	3
B	→	A
C	→	5
D	→	2
E	→	9
F	→	B

3.2 ADDER



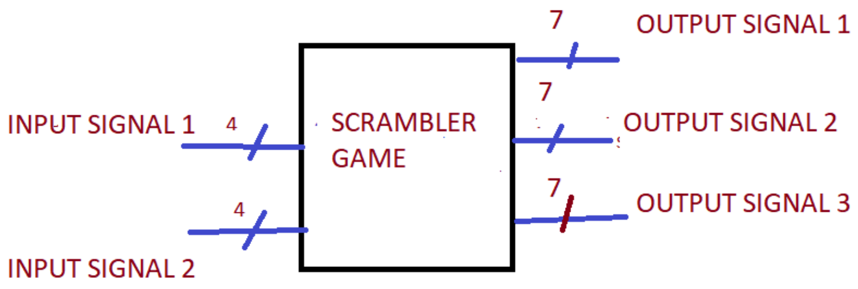
This module is used to find the sum of two 4 digit input signal. Here we are ignoring the carry bit. Output of the signal is 4 digit .One of the input signal is the output of scramble and other input is 1st Toggle switch.

3.3DECODER 4x7



This module convert the 4 digit signal into 7 bit signal that support 7 Segment display. This signal is very important to understand te nature and make the city green.

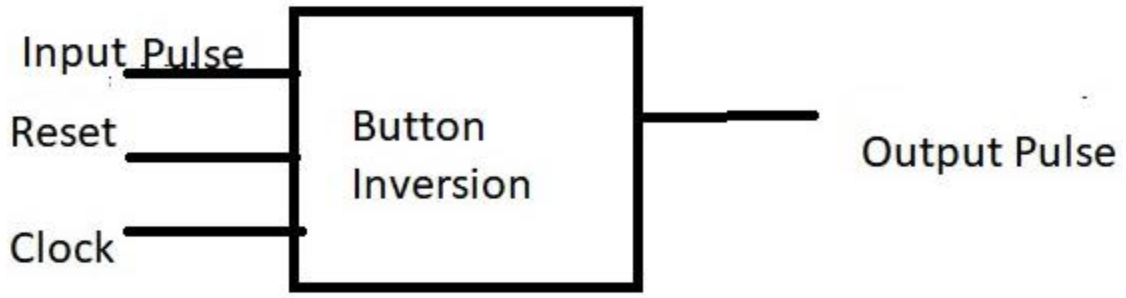
3.5 Scramble



This module implements adder, scramble and decoder modules. In this module it accepts 2 input signal and output signal are 3. This module is top module it implement the main game logic. where one user has to guess the scrambled number.

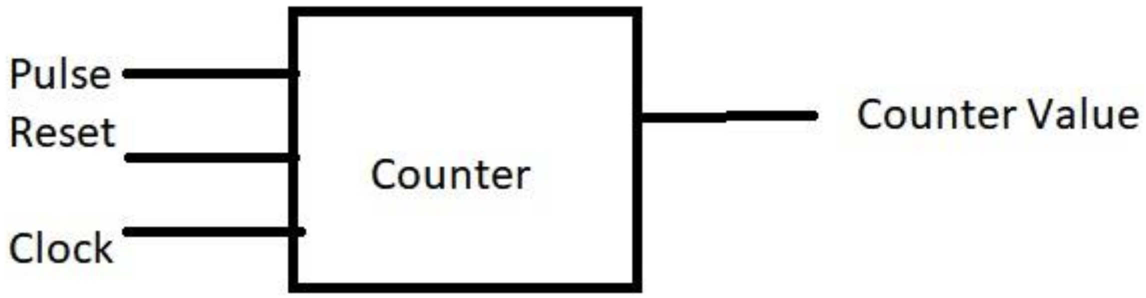
3.5 Button Inversion:

This helps is generating a low pulse whenever the push button is pressed and the reset is set. If it is pressed Pb_in_s =0 and if Pb_in_s =0 then Inv_sg_s=1 for one cycle called the single cycle pulse otherwise Inv_sg_s=0 and that means that button has not been pushed. Basically converts a low pulse to a high pulse.



This shows all possible input and output of Button Inversion Test bench used .

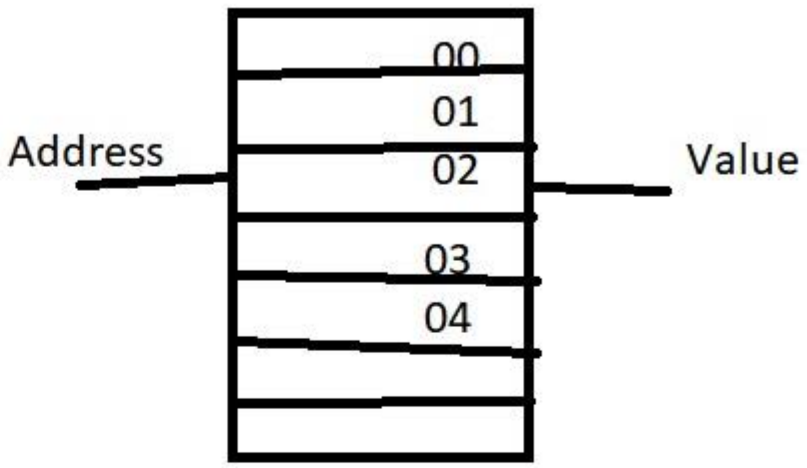
3.6 Counter:



This Module will start starts counting when active-low reset is de-asserted and rolls over to 0 when it reaches its maximum value of 4'hF. It will keep counting as long as it is provided with a running clock and reset is held high.

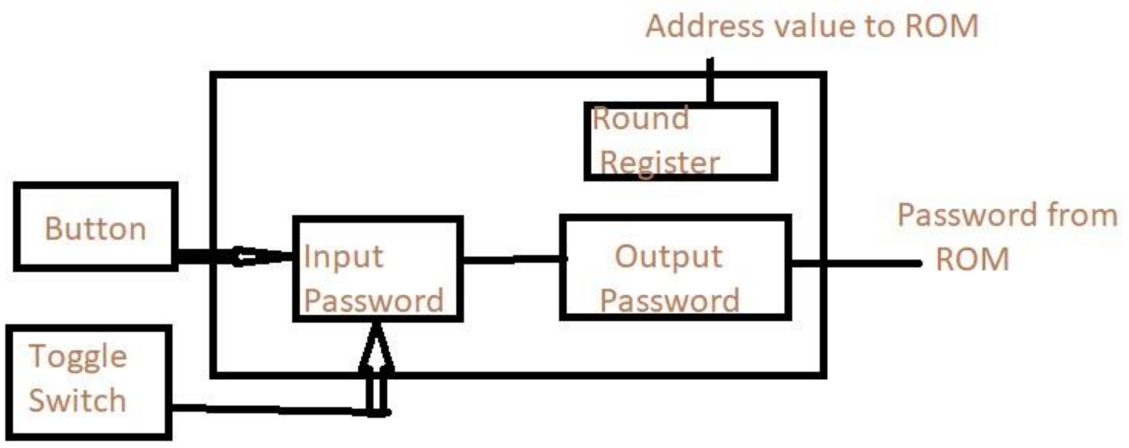
3.7 ROM Module:

This module will save the values at different addresses. We can access the values by giving the address as input



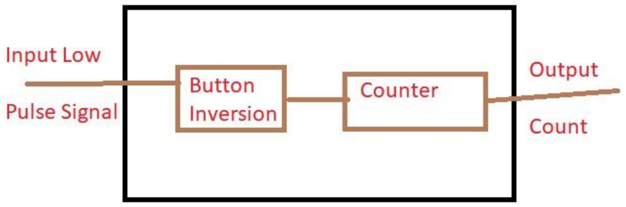
If we give the address value at the input we get the value saved at that address as the output. Here in the diagram shows the address values.

3.8 Access Module:



In this module when user press the button after resetting and setting the 4 bit password at each time ,the input password is compared with the password we got from the output password from ROM module. Thus we compare the input password with the password from ROM module here.

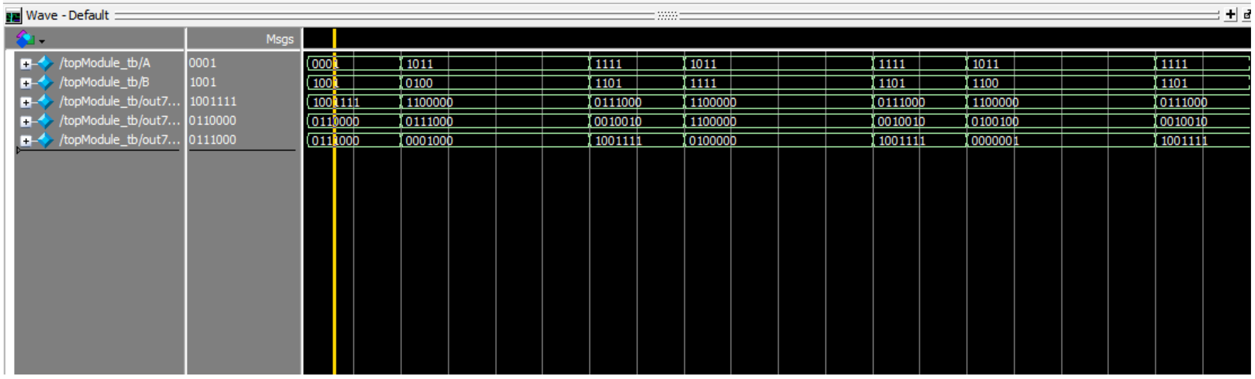
3.9 Random Number Generator



This module is combination of the button Inversion and Counter . Output of this module is the 4 bit Count value whereas the input is the signal.

Depending how long user press the button the module decides the counter values.

4. Waveforms

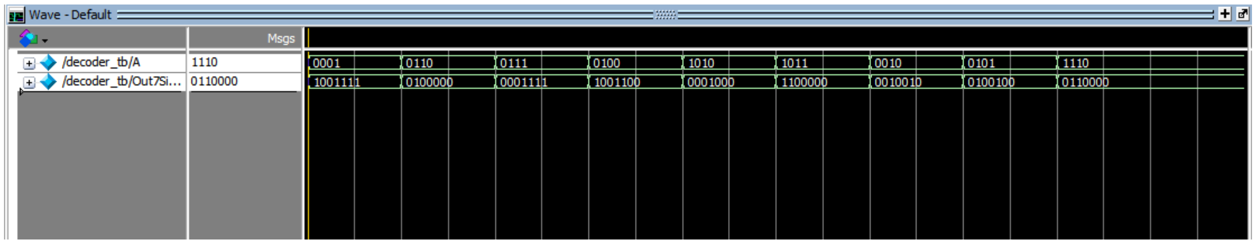


```
Wave - Default
                              Msgs
/topModule_tb/A        0001      0001     1011     1111     1011     1111     1011     1111
/topModule_tb/B        1001      1001     0100     1101     1111     1101     1100     1101
/topModule_tb/out7...  1001111   1001111  1100000  0111000  1100000  0111000  1100000  0111000
/topModule_tb/out7...  0110000   0110000  0111000  0010010  1100000  0010010  0100100  0010010
/topModule_tb/out7...  0111000   0111000  0001000  1001111  0100000  1001111  0000001  1001111
```

Figure 0: Adder module

As seen from waveform (Figure 1). In this graph the top module function applies all" scrambler top module" logics. From graph, for some of the examples we can see the output is F. this happen when we get sum 15 after addition of two signals. Getting "F" or 15 is the winning condition of this game.

Readings: 1001 + 0001 (Scrambled) => 1001 +0110 (Unscrambled) = 1111



```
Wave - Default
                              Msgs
/decoder_tb/A          1110      0001     0110     0111     0100     1010     1011     0010     0101     1110
/decoder_tb/Out7Si...  0110000   1001111  0100000  0001111  1001100  0001000  1100000  0010010  0100100  0110000
```

Figure 1: Decoder -Output Waveform

From above Waveform, we can see that 4-bit numbers are being decoded to 7-bit numbers. This 7-bit numbers are helpful to send to 7 segment displays .

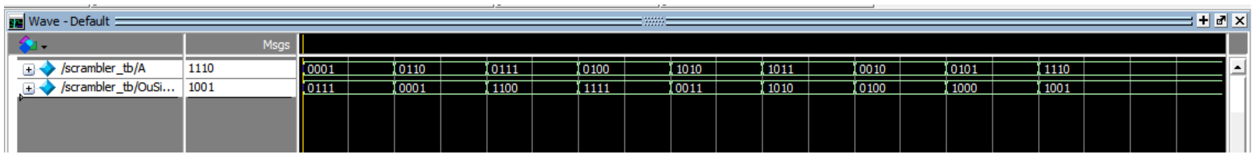


Figure 2: Scrambler -Output Waveform

From above waveform, we can see that every input of scrambler is mapped to unique new value in the output such that output range is in between 0 to 15.

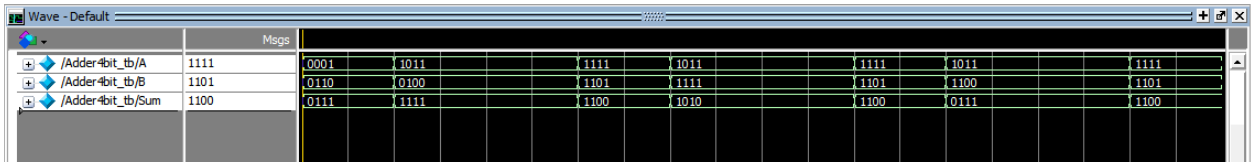


Figure 3: Adder -Output Waveform

From Above waveform, different sum result of the different 4 digit number can be observed .

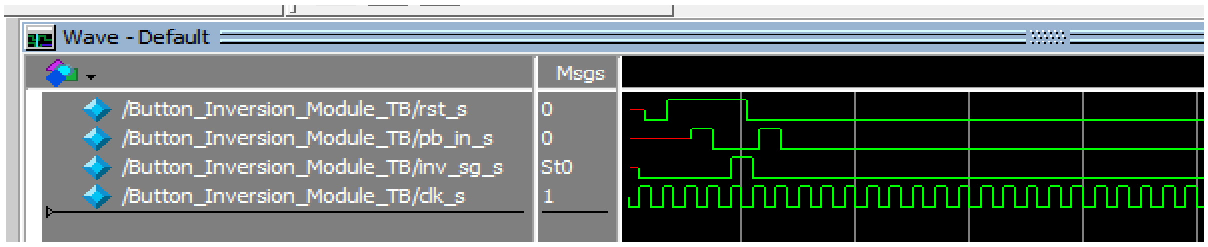

Wave - Default
Msgs
/Button_Inversion_Module_TB/rst_s 0
/Button_Inversion_Module_TB/pb_in_s 0
/Button_Inversion_Module_TB/inv_sg_s St0
/Button_Inversion_Module_TB/clk_s 1

Figure 4: Button Inversion Waveform

From Above waveform, It can be seen generating a low pulse whenever the push button is pressed and the reset is set.

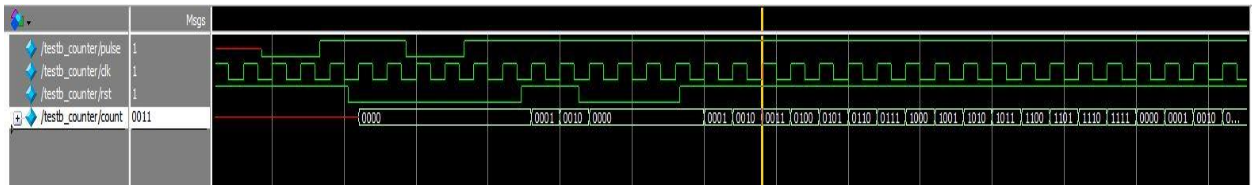


Figure 5: Counter Waveform

rom Above waveform, Counter starts incrementing when pulse =1 and reset=1. It will reset when reset is zero or incase when pulse is 0.

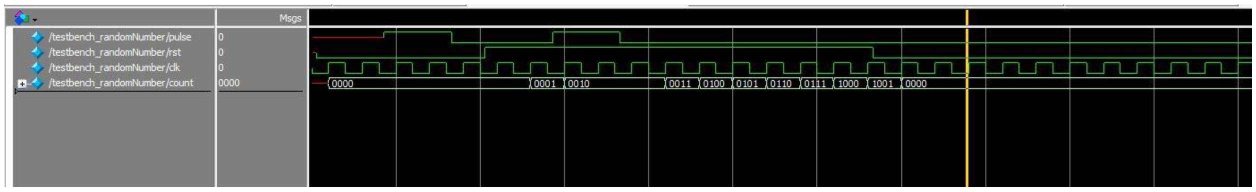

Msgs
/testbench_randomNumber/pulse 0
/testbench_randomNumber/rst 0
/testbench_randomNumber/clk 0
/testbench_randomNumber/count 0000
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
0000

Figure 6: Random Number Generator

This module shows that on low input pulse the input start incrementing once user release the button it stop counting and it will show the output as the counter number.

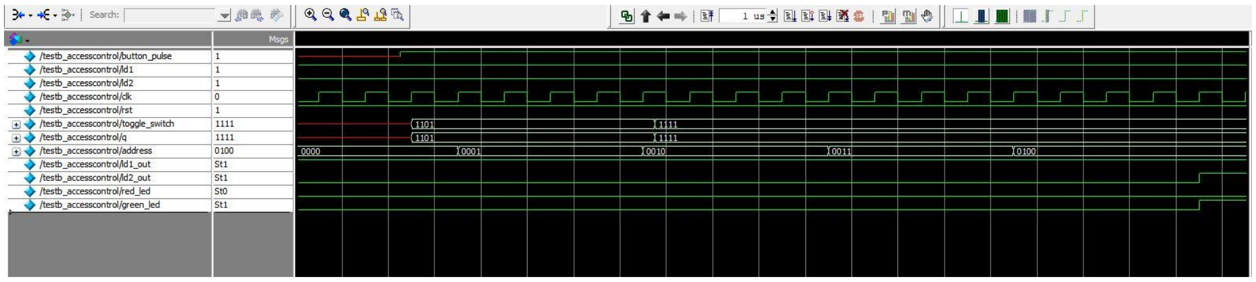

Search:
Msgs
1 us
/testb_accesscontrol/button_pulse 1
/testb_accesscontrol/ld1 1
/testb_accesscontrol/ld2 1
/testb_accesscontrol/clk 0
/testb_accesscontrol/rst 1
/testb_accesscontrol/toggle_switch 1111
/testb_accesscontrol/q 1111
/testb_accesscontrol/address 0100
/testb_accesscontrol/ld1_out St1
/testb_accesscontrol/ld2_out St1
/testb_accesscontrol/red_led St0
/testb_accesscontrol/green_led St1
1101
1111
1101
1111
0000
0001
0010
0011
0100

Figure 7: Access Control Waveform and ROM Password Module

This module will compare the password inserted by user and the password saved in the ROM module. Once the password is verified the Green Led will be ON and when password is not verified the red led is OFF.

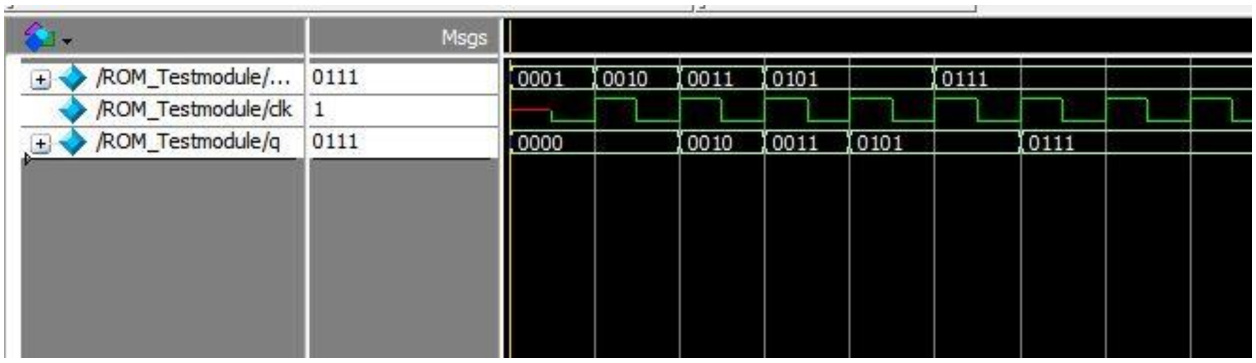


Figure 1: ROM Module

This module will verify the address and saved values .

5 And Now its Game Time

I asked my roommates to play this game. Tabish Kamran and Zishan Amlani played this game. They are master’s student in the field of Mechanical Engineering at UH. The Game rule remains the same and I have set 5 matches and whoever wins most number of Games wins the Games. The secret mapping indexes card is showed to them for very short time to go through it .

Unscrambled		Scrambled
0	→	D
1	→	7
2	→	4
3	→	0
4	→	F
5	→	8
6	→	1
7	→	C
8	→	6
9	→	E
A	→	3
B	→	A
C	→	5
D	→	2
E	→	9
F	→	B

The Game Rule remain same , first player will verify his 16 bit password on the board and once he verify the password the Green LED will light up and incase of not verified the red LED will be ON. Once password is verified the player 1 will press the button, depend on how long he press button It will show the random number using the button and then player2 test will be to remember the scrambled number correspond to this number and set other number such that sum of 1st random number and 2nd person unscrambled number will be displayed in 3rd 7 segment display. If the result is 15 second player wins the game.This game will continue until it comes to common understanding

6 FPGA Screen Shot

Password Verification:

First user has to verify 16 bit password by entering 4 bit each time using toggle switch. and once password is verified he must make sure Green LED is ON (Green LED represent the password is verified and RED led represent that password is incorrect)

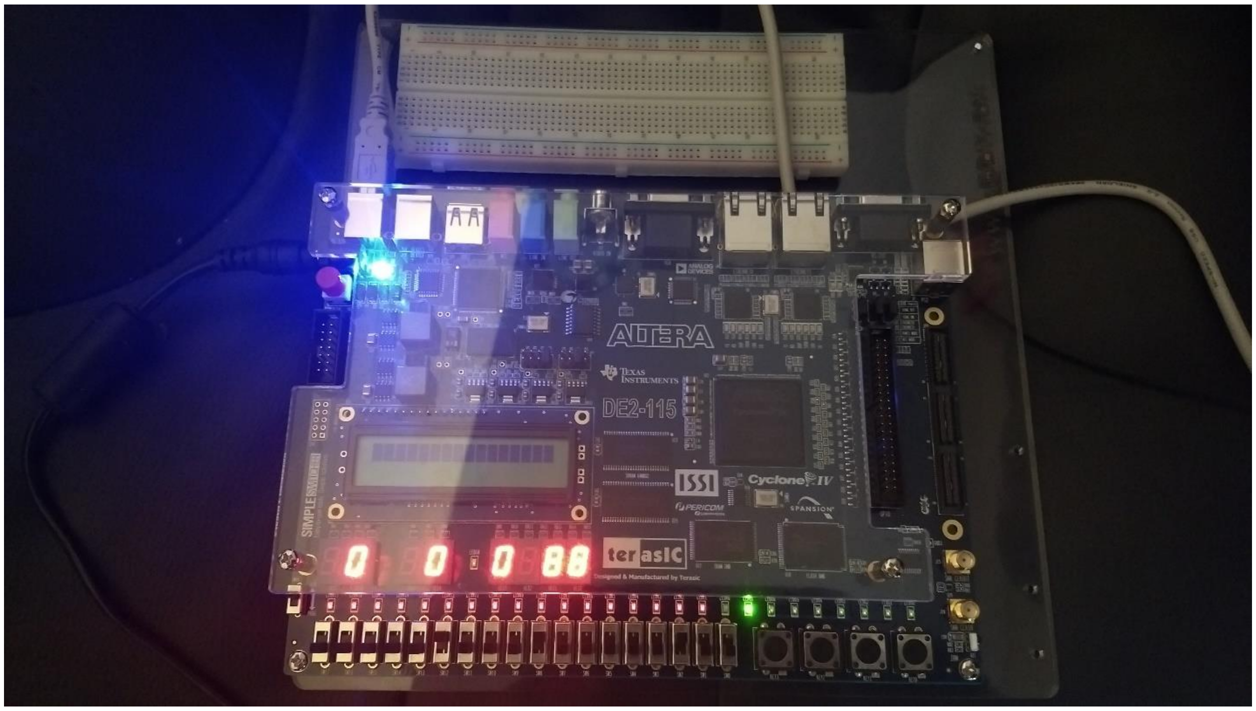


Figure 2: Green LED is ON (Password is verified.

Password not verified Incident:

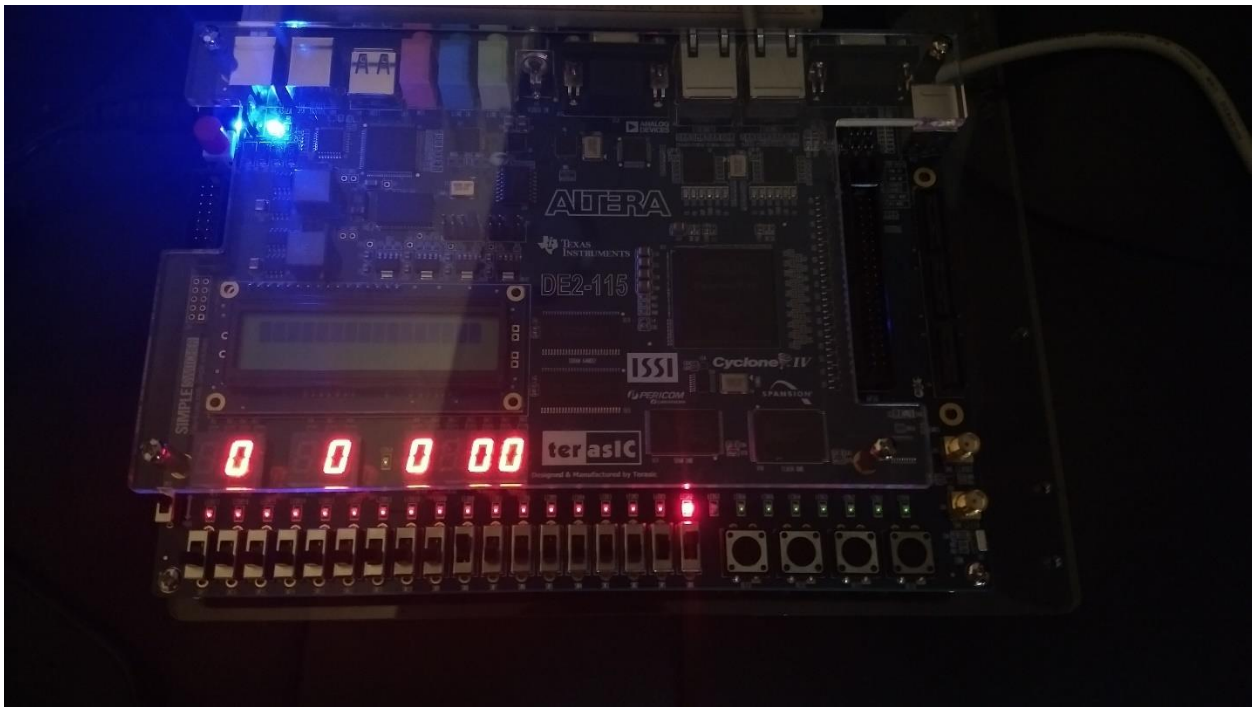
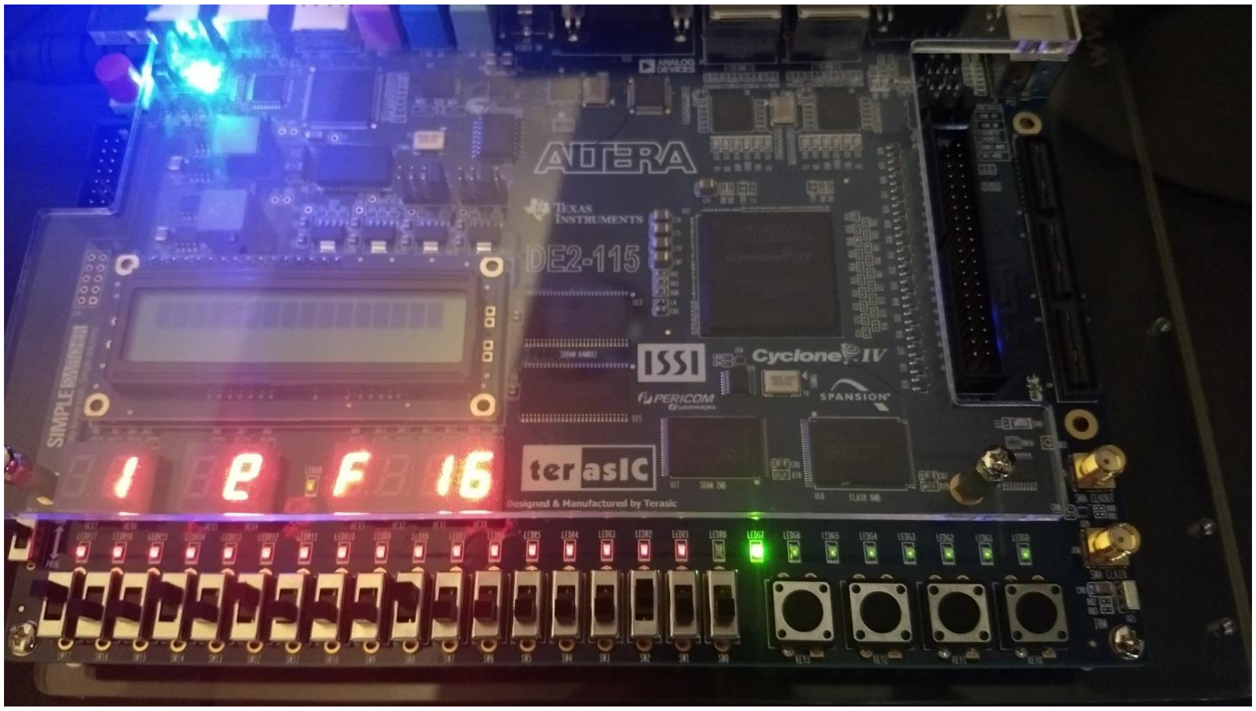


Figure 3: RED LED is ON (password is not verified)

Random Number Generation:

Once password is verified he must again press password verification button to reload the input on to timer and after that user can long press the random number generator button. Depending on how long he presses a random number will be generated and displayed on the 7 segments and also the timer will start to count.



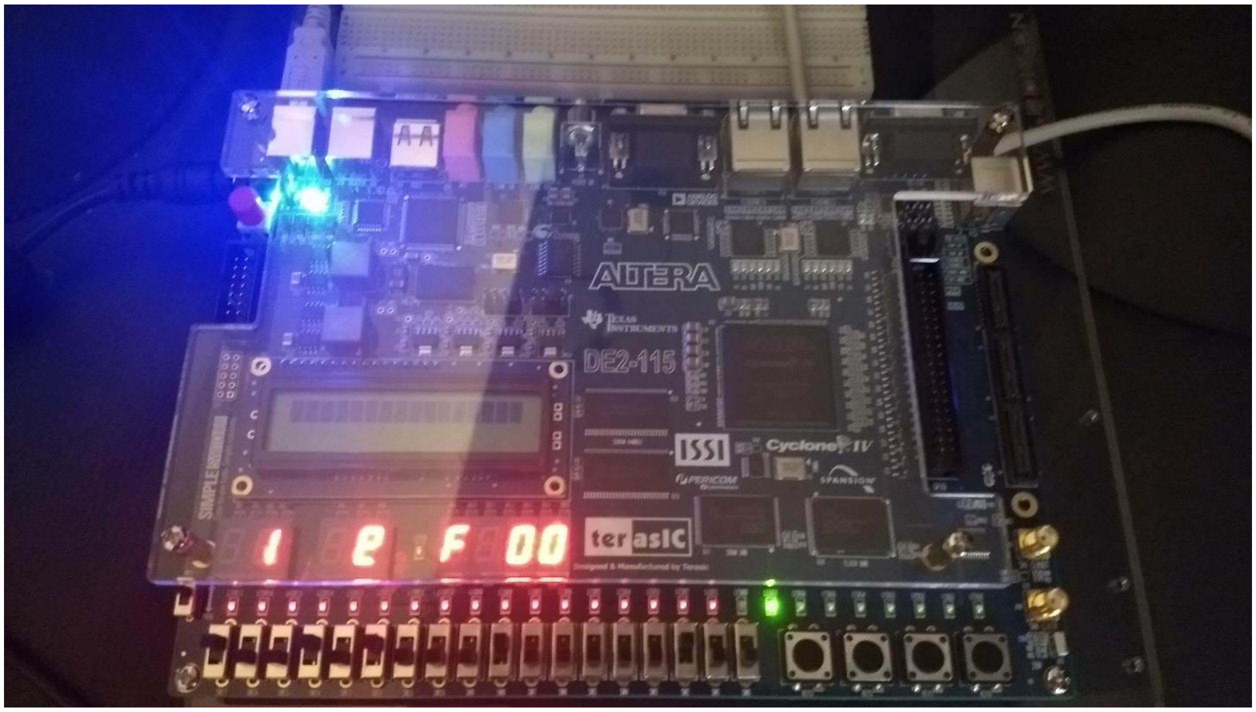
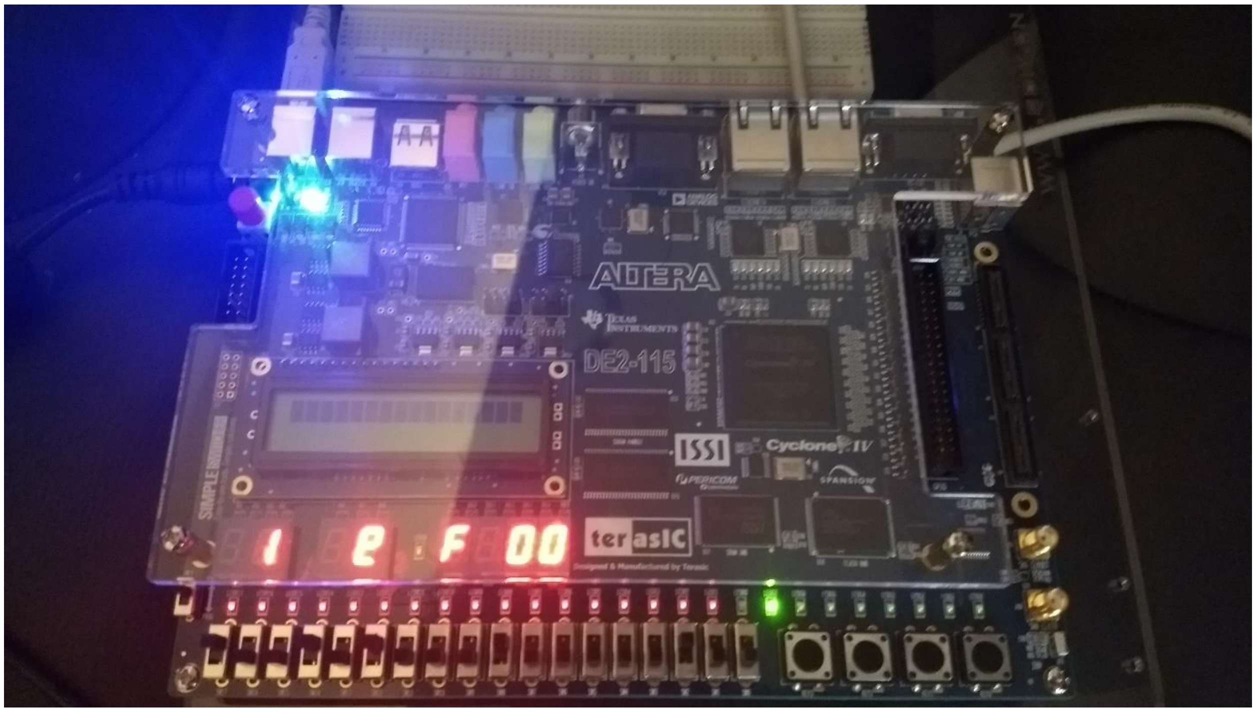


Figure 4: Random Number generation after long pressing button and timer initiation

Then player must now calculate how much number he must add to get sum of 15. But twist in the game is that user must give scrambled input of that number as input to the system. He has then to press the load button so that number get loaded on to the system.

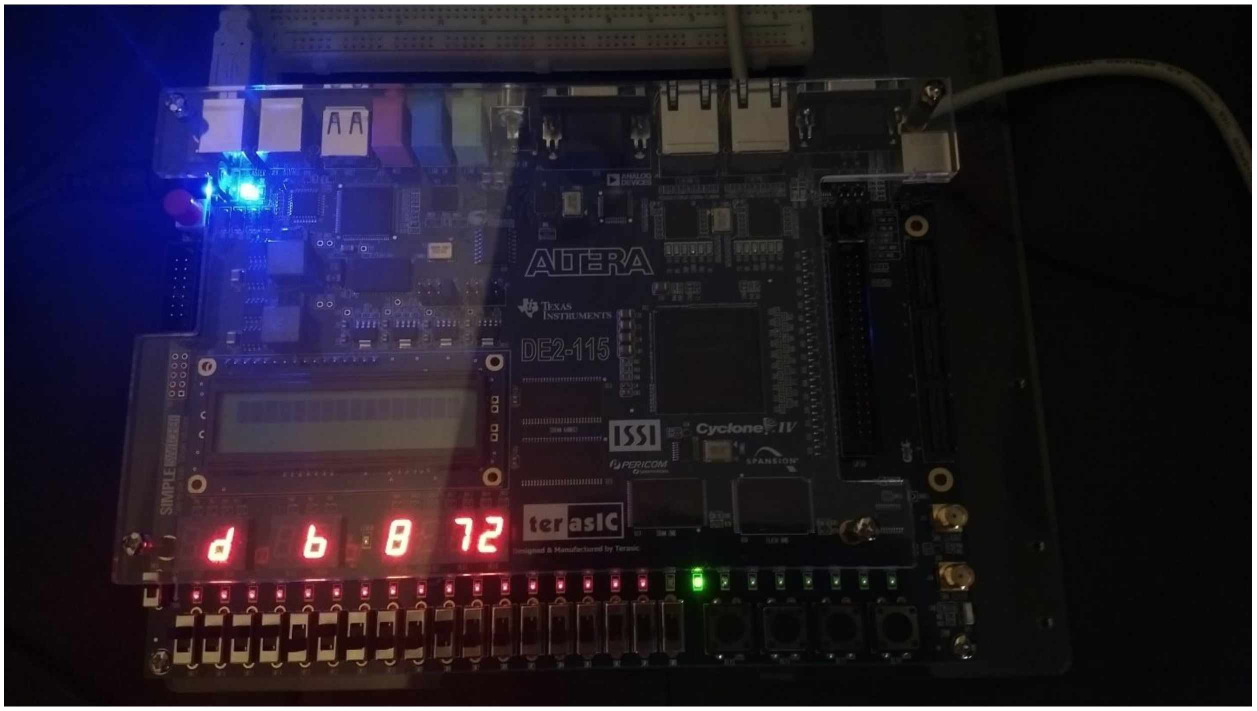
Example 1:

If we assume player one has pressed the random number button and that is displayed on 7 segments as 9 and now player user calculate that he need to add 5 more to get the result but he can't give 6 directly rather he has to give scrambled input of 5. If we assume he remember correct mapping number of 6 which was for example was 8 then upon selecting the scrambled number using toggle switches we get the sum of the result on the 7segment.



Example 2:

If we assume that 1st person was unsuccessful to assume the right scrambled number which can also be seen in the 7Segment as the sum value on the 7Segment won't reflect the "F" value In such cases 1st person will loose points as he loss the game and meanwhile 2nd person gets extra points as he won the game again.
This can also be shown in below picture



7 Appendix

Module Design:

Timer Module

```
module t_timer(led,button_pulse,pulse,clk,rst,input1,input2,timerOutt1,timerOutt2,splayer);
        input clk,rst,pulse,led,button_pulse;
        input [3:0]input1,input2;
    //output  reg [6:0] outSignal1,outSignal;
        output reg [3:0] timerOutt1,timerOutt2;
        output reg splayer;
        // reg [25:0] swire;
         reg [25:0] slow_clk;
         reg [3:0]timer1,timer2;
         reg flag=0;
```

```
 reg flag1=0;
always @(posedge clk)
        begin
                if(rst == 0)
                        begin
                                timer1<=0;
                                timer2<=0;
                                slow_clk <= 0;
                                flag=0;
                                flag1=0;
                                splayer=0;
                        end
                else
                        begin
                                if(timer1 == 4'b0000 && timer2 == 4'b0000) begin
                                        flag=0;
                                        flag1=0;
                                        splayer=0;
                                end
                                if(button_pulse==1 && led ==1) begin
                                        timer1<=input2;
                                        timer2<=input1;
                                        flag=1;
                                 end
                                if(pulse==1 && flag==1) begin
                                        flag1=1;
                                        splayer=1;
                                end
                                if(flag1==1) begin
if(timer1 > 4'b1001) begin
              timer1<= 4'b1001;
end
if(timer2 > 4'b1001) begin
              timer2<= 4'b1001;
end
if (slow_clk == 26'd50000000) begin
              if(timer1==4'b0000) begin
                            timer2 <= timer2 - 1'b1;
              end
                            timer1 <= timer1 - 1'b1;


        slow_clk<= 0;
 end
    else begin
        slow_clk <= slow_clk + 1'b1;
    end
if(timer1 > 4'b1001) begin
              timer1<= 4'b1001;
end
if(timer2 > 4'b1001) begin
              timer2<= 4'b1001;
end
end



 timerOutt1=timer2;
 timerOutt2=timer1;
//assign swire=slow_clk;
```

```
                end
        end

endmodule
```

Adder Module

```
//  ECE6370
// Author: Sohail Shareef  Mohammed - 5985
// Application : Adder Function calculate the sum of the two 4-bit numbers.
module full_add( a ,b ,sum );
output [4:0]sum ;
input [3:0]a ;
input [3:0]b ;
assign sum = a + b;
endmodule
```

Scramble Module

```
/  ECE6370
// Author: Sohail Shareef  Mohammed - 5985
// Application : This will map every number between 0 to 15 to someother unique number between 0 to
15. This mapping will be done in accordance to index card.
module scrambler(unScrambled,scrambled);

    input [3:0]scrambled;
    output [3:0]unScrambled;
    reg [3:0]unScrambled;

    always @(scrambled)
        begin
```

```
            case(scrambled)
                4'b0000: begin unScrambled =4'b1101;end
                4'b0001: begin unScrambled =4'b0111;end
                4'b0010: begin unScrambled =4'b0100;end
                4'b0011: begin unScrambled =4'b0000;end
                4'b0100: begin unScrambled =4'b1111;end
                4'b0101: begin unScrambled =4'b1000;end
                4'b0110: begin unScrambled =4'b0001;end
                4'b0111: begin unScrambled =4'b1100;end
                4'b1000: begin unScrambled =4'b0110;end
                4'b1001: begin unScrambled =4'b1110;end
                4'b1010: begin unScrambled =4'b0011;end
                4'b1011: begin unScrambled =4'b1010;end
                4'b1100: begin unScrambled =4'b0101;end
                4'b1101: begin unScrambled =4'b0010;end
                4'b1110: begin unScrambled =4'b1001;end
                4'b1111: begin unScrambled =4'b1011;end
                default: begin unScrambled =4'b0000;end
            endcase
        end
endmodule
```

Decoder Module

```
//  ECE6370
// Author: Sohail Shareef  Mohammed - 5985
// Application : This will convert 4-bit BCD to 7 bit binary number such that it is compatible to 7 segment
display
module decoder4to7(inpSignal,outSignal);
```

```
        input [3:0]inpSignal;
        output [6:0]outSignal;
        reg [6:0]outSignal;

        always @(inpSignal)
                begin
                        case(inpSignal)
                                4'b0000: begin outSignal =7'b0000001;end
                                4'b0001: begin outSignal =7'b1001111;end
                                4'b0010: begin outSignal =7'b0010010;end
                                4'b0011: begin outSignal =7'b0000110;end
                                4'b0100: begin outSignal =7'b1001100;end
                                4'b0101: begin outSignal =7'b0100100;end
                                4'b0110: begin outSignal =7'b0100000;end
                                4'b0111: begin outSignal =7'b0001111;end
                                4'b1000: begin outSignal =7'b0000000;end
                                4'b1001: begin outSignal =7'b0000100;end
                                4'b1010: begin outSignal =7'b0001000;end
                                4'b1011: begin outSignal =7'b1100000;end
                                4'b1100: begin outSignal =7'b0110001;end
                                4'b1101: begin outSignal =7'b1000010;end
                                4'b1110: begin outSignal =7'b0110000;end
                                4'b1111: begin outSignal =7'b0111000;end
                                default: begin outSignal =7'b0000000;end
                        endcase
                end
endmodule
```

Top Module (Scramble Game)

```
// ECE6370

// Author: Sohail Shareef 5985

//Top level game module with authentication

module
Lab4_Sohail_M(button1,toggle1,rst,clk,ld1,ld2,switch_num2,green_led,red_led,sevenseg_num1,sevens
eg_num2,sevenseg_sum);


 input[3:0]  switch_num2;

 input button1,ld1,ld2,rst,clk;

 input [3:0] toggle1;

 output[6:0] sevenseg_num1, sevenseg_num2, sevenseg_sum;

 output green_led, red_led;

 wire[3:0] unscrambled_num, sum,Q_num1,Q_num2,Q_sum;

 wire ld_1,ld_2,ld1_out,ld2_out,button_pulse;

 wire [3:0] address,q;

 wire [3:0] switch_num1;

 button_shaper b1(button1, button_pulse, clk, rst);

 //button_shaper ld1button(ld1,ld_1,clk, rst);

 button_shaper ld2button(ld2,ld_2,clk, rst);


 ROM_Top_Test rmt(address,clk, q);



 access_control a1(button_pulse, toggle1,ld1, ld_2, red_led, green_led, ld1_out, ld2_out, clk,
rst,address,q);




 unscramble unscramblenum2(switch_num2, unscrambled_num);
 randomNumb(ld1_out,rst,clk,switch_num1);


 load_register l1(switch_num1,Q_num1,ld1_out,clk,rst);

 load_register l2(unscrambled_num,Q_num2,ld2_out,clk,rst);



 adder addnums(Q_num1, Q_num2, sum);



 seven_seg sevensegnum1(Q_num1, sevenseg_num1);

 seven_seg sevensegnum2(Q_num2, sevenseg_num2);

 seven_seg sevensegnum3(sum, sevenseg_sum);

endmodule
```

Counter

```
module counter(pulse,clk,rst,count);

        input clk,rst,pulse;

        output reg [3:0] count;

        //reg [3:0] count;


        always @(posedge clk)

                begin

                        if(rst == 0)

                                begin

                                        count<=4'b0000;

                                end

                        else
                                begin
                                        if(pulse==1) begin
                                                count <=count+4'b0001;
                                        end

                                end
                end
endmodule
```

Button Inversion

```
// ECE6370
// Author: Sohail Shareef 5985
// This module is to create a button shapper ,which takes a low pulse
// to generate a single high pulse
module button_inv(button_push, button_pulse, clk, rst);
 //declaring inputs and outputs
 input button_push;
 output button_pulse;
 reg button_pulse;

 //clock and reset
 input clk,rst;

 //register to store state
 reg[2:0] state,state_next;
 parameter s_wait=0, state1=1;
```

```
always @ (posedge clk)
begin
 if(rst==0)
  button_pulse<=0;
 else
  if(button_push==0) begin
        button_pulse<=1;
      end
  else begin
      button_pulse<=0;
 end
end
endmodule
```

ROM Module:

```
`timescale 1 ps / 1 ps
// synopsys translate_on
module ROM_Top_Test (
              address,
              clock,
              q);

              input    [3:0]  address;
              input      clock;
              output  [3:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
```

```
`endif
	tri1	clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [3:0] sub_wire0;
	wire [3:0] q = sub_wire0[3:0];

	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
				.address_b (1'b1),
				.addressstall_a (1'b0),
				.addressstall_b (1'b0),
				.byteena_a (1'b1),
				.byteena_b (1'b1),
				.clock1 (1'b1),
				.clocken0 (1'b1),
				.clocken1 (1'b1),
				.clocken2 (1'b1),
				.clocken3 (1'b1),
				.data_a ({4{1'b1}}),
				.data_b (1'b1),
				.eccstatus (),
				.q_b (),
                                        .rden_a (1'b1),
                                        .rden_b (1'b1),
                                        .wren_a (1'b0),
                                        .wren_b (1'b0));
        defparam
                altsyncram_component.address_aclr_a = "NONE",
                altsyncram_component.clock_enable_input_a = "BYPASS",
                altsyncram_component.clock_enable_output_a = "BYPASS",
                altsyncram_component.init_file = "ROM_top.mif",
                altsyncram_component.intended_device_family = "Cyclone IV E",
                altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 16,
                altsyncram_component.operation_mode = "ROM",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a = "CLOCK0",
                altsyncram_component.widthad_a = 4,
                altsyncram_component.width_a = 4,
                altsyncram_component.width_byteena_a = 1;


endmodule
```

Access Module

```
// ECE6370
// Author: Sohail Shareef 5985
// This module is to create a access control to allow a validation of a 6 bit password
// by checking each bit and then produce a green light if all matches or red if
// any fail

module access_control(button_pulse, toggle_switch,ld1, ld2, red_led, green_led, ld1_out, ld2_out, clk,
rst,address,q);


 input button_pulse,ld1, ld2;

 input clk,rst;

 input [3:0]toggle_switch,q;

 output reg [3:0] address;

 output reg ld1_out, ld2_out;

 output reg red_led, green_led;


 reg flag;

 reg [2:0] state;

 parameter s_wait=0,s1=1,s2=2,s3=3,s4=4,s5=5,s6=6,s7=7,s8=8;



 always @ (posedge clk)

 begin

 if(rst==0) begin

   state<=s_wait;

   flag=1;

   address <= 4'b0000;

 end

 else

  begin

   case(state)


     s_wait:

      begin
 flag=1;
 red_led<=0;
 green_led<=0;
 ld1_out<=1;
 ld2_out<=0;
 address <= 4'b0000;
 if(button_pulse==1)
   state<=s1;
else
   state<=s_wait;
 end

s1: begin
        red_led<=0;
        green_led<=0;

        if(button_pulse==1) begin
                    address <= address+4'b0001;
                    state<=s2;
        end
        else begin
        state<=s1;
        end
        end

s2:
 begin
  state<=s3;
 end
 s3:
  begin
   state<=s5;
   end
 s5:
  begin
         if(toggle_switch!=q) begin
         flag=0;


         end
         if(address==4'b0100) begin
                 state<=s7;
         end
         else begin
         state<=s1;
         end
         end


s7:
  begin
   if(flag==1)begin
         red_led<=0;
   green_led<=1;
          if(ld1==1) begin
                 ld1_out<=1;
          end
          else begin
          ld1_out=0;
                 end
```

```
              end
              else begin
              red_led<=1;
          green_led<=0;
              end
              if(ld2==1)
          ld2_out<=1;
         else ld2_out<=0;
        end
      default:
        state<=s_wait;
     endcase
   end
end
endmodule
```

TEST BENCHES

Adder Test Bench

```
`timescale 1ns / 1ps

module Adder4bit_tb();

 // Inputs

 reg [3:0] A;

 reg [3:0] B;
 // Outputs

 wire [3:0] Sum;



 // Instantiate the Unit Under Test (UUT)

 initial begin

  // Initialize Inputs

  A    = 4'b0001;

  B    = 4'b0110;

  // Wait 100 ns for global reset to finish

  #0.1;

  // Add stimulus here
```

Scrambled Test Bench

```
`timescale 1ns / 1ps

module decoder_tb();
```

```
// Inputs

reg [3:0] A;

// Outputs

wire [6:0] Out7Signal;

decoder4to7 decodert (

		A,

		Out7Signal);

initial begin

 // Initialize Inputs


 A    = 4'b0001;#0.1;


 A    = 4'b0110;#0.1;
 A    = 4'b0111;#0.1;
A    = 4'b0100;#0.1;
 A    = 4'b1010;#0.1;
A    = 4'b1011;#0.1;
 A    = 4'b0010;#0.1;
A    = 4'b0101;#0.1;
 A    = 4'b1110;
```

```
end
endmodule
```

Decoder Test Bench

```
`timescale 1ns / 1ps

module decoder_tb();

 // Inputs

 reg [3:0] A;

 // Outputs

 wire [6:0] Out7Signal;



 // Instantiate the Unit Under Test (UUT)

 decoder4to7 decodert (

                A,

                Out7Signal);

 initial begin

  // Initialize Inputs
  A   = 4'b0001;#0.1;

  A   = 4'b0110;#0.1;
  A   = 4'b0111;#0.1;
A   = 4'b0100;#0.1;
  A   = 4'b1010;#0.1;
A   = 4'b1011;#0.1;
  A   = 4'b0010;#0.1;
A   = 4'b0101;#0.1;
  A   = 4'b1110;

  //;

 end
endmodule
```

Counter Test Bench :

```
// ECE6370
// Author: Sohail Shareef 5985
//testbench for counter
`timescale 1ns/100ps
module testb_counter();
 reg pulse, clk, rst;

wire [3:0] count;
```

```
counter cntr(pulse,clk,rst,count);

always
 begin
  clk <= 0;
  #10;
  clk <= 1;
  #10;
 end // Note: Procedure repeats
// Vector Procedure
 initial
  begin
#3 rst=1;
#40 pulse =0;
#40 pulse =1;
#20 rst=0;
#40 pulse =0;
#40 pulse =1;
#40 rst =1;
#40 rst =0;
#70 rst=1;



end
endmodule
```

Button Inversion:

```
// ECE6370
// Author: Sohail SHareef 5985
//test bench for button_shaper
`timescale 1ns/100ps
module testbench_inversion ();
 reg button_push, clk, rst;
 wire button_pulse;

button_inv tbtn(button_push, button_pulse, clk, rst);
 always
 begin
  clk <= 0;
  #10;
  clk <= 1;
  #10;
 end // Note: Procedure repeats
// Vector Procedure
 initial
  begin
  rst=1;
   button_push=1;
  @(posedge clk);
  @(posedge clk);
  button_push=0;
  @(posedge clk);
  @(posedge clk);
  button_push=1;
  @(posedge clk);
  @(posedge clk);
  @(posedge clk);
  @(posedge clk);
  @(posedge clk);
  rst=0;
  @(posedge clk);
  @(posedge clk);
  @(posedge clk);
  @(posedge clk);
  @(posedge clk);
  rst=1;
  button_push=0;
  @(posedge clk);
  button_push=1;
  @(posedge clk);
  @(posedge clk);
  button_push=0;
  end
endmodule
```

Top Module :

```
`timescale 10ns/100ps
module Test_overall();

reg[3:0]  switch_num2;
 reg button_pulse,ld1,ld2,rst,clk;
 reg [3:0] toggle_switch;
 wire[6:0] sevenseg_num1, sevenseg_num2, sevenseg_sum;
 wire green_led, red_led;
// wire [3:0]address,q;
```

```
Lab4_Sohail_M
lab(button_pulse,toggle_switch,rst,clk,ld1,ld2,switch_num2,green_led,red_led,sevenseg_num1,sevense
g_num2,sevenseg_sum);


always begin

             #5 clk =0;

             #5 clk =1;

end


initial begin


switch_num2 = 4'b1101;

 ld1=0;

 ld2=0;




rst=0;

 ld1=0;

 ld2=0;

rst=1;

 ld1=0;

 ld2=0;


 ld1=0;

 ld2=0;

#15 button_pulse=0;

#5 toggle_switch=4'b0001;


#15 button_pulse=0;
```

```
#5 toggle_switch=4'b0010;

#15 button_pulse=0;

/*#5 toggle_switch=1;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 toggle_switch=0;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 toggle_switch=1;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
```

```
#5 button_pulse=0;


#5 toggle_switch=0;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 toggle_switch=1;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;*/


#5 toggle_switch=4'b0011;


#15 button_pulse=1;


#5 toggle_switch=4'b0100;


#15 button_pulse=0;






end
```

```
endmodule
```

Random Number Test Bench :

```
// ECE6370
// Author: Sohail SHareef 5985
//test bench for button_shaper
`timescale 1ns/10ps
module testbench_randomNumber();
 reg pulse ;
 reg rst,clk;
 wire [3:0]count;

 randomNumb rnG(pulse,rst,clk,count);
 always
 begin
  clk <= 0;
  #10;
  clk <= 1;
  #10;
 end // Note: Procedure repeats
// Vector Procedure
 initial
  begin
#3 rst=0;
```

```
#40 pulse =1;
#40 pulse =0;
#20 rst=1;
#40 pulse =1;
#40 pulse =0;
#40 rst =1;
#40 rst =1;
#70 rst=0;
  end
endmodule
```

ROM MODULE TESTBENCH :

```
`timescale 10ns/100ps
module ROM_Testmodule();

reg      [3:0]  address;
         reg       clk;
         wire    [3:0]  q;



ROM_Top_Test rnd(
         address,
         clk,
         q);
always begin
         #5 clk =0;
         #5 clk =1;
```

```
end

initial begin

                address =4'b001;
 #10 address =4'b010;

#10 address =4'b011;
#10 address =4'b101;
#10 address =4'b101;
#10 address =4'b111;



end

endmodule
```

Access Control Test Bench :

```
// ECE6370
// Author: Sohail Shareef 5985
//testbench for accesscontrol
```

```
`timescale 1ns/100ps
module testb_accesscontrol();
 reg button_pulse,ld1, ld2;
 reg clk,rst;
 reg [3:0]toggle_switch,q;
 wire [3:0] address;
 wire ld1_out, ld2_out;
 wire red_led, green_led;


access_control acs(button_pulse, toggle_switch,ld1, ld2,
red_led, green_led, ld1_out, ld2_out, clk, rst,address,q);
always
 begin
  clk <= 0;
  #10;
  clk <= 1;
  #10;
 end // Note: Procedure repeats
// Vector Procedure
```

```
 initial
  begin
rst=0;
ld1=1;
ld2=1;
@(posedge clk);
rst=1;
@(posedge clk);
@(posedge clk);
@(posedge clk);
#15 button_pulse=1;
#5 toggle_switch=4'b1101;
 q=4'b1101;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
```

```
#5 toggle_switch=4'b1111;
 q=4'b1111;
#15 button_pulse=1;
@(posedge clk);
/*#5 toggle_switch=1;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 toggle_switch=0;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;
@(posedge clk);
```

```
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 toggle_switch=1;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 toggle_switch=0;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;
@(posedge clk);
```

```
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 toggle_switch=1;
@(posedge clk);
#15 button_pulse=1;
@(posedge clk);
#5 button_pulse=0;*/
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 toggle_switch=4'b1111;
 q=4'b1111;
#15 button_pulse=1;
@(posedge clk);
@(posedge clk);
@(posedge clk);
```

```
@(posedge clk);


@(posedge clk);
@(posedge clk);
#5 toggle_switch=4'b1111;
 q=4'b1111;
#15 button_pulse=1;
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
#5 ld1=1;
#5 ld2=1;
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
```

```
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);


end
endmodule
```