CSE100 Lab 5

1 Lab Overview

In this laboratory we were tasked to develop a sequential circuit logic that ran a simple game. The game consisted on two players waiting a random amount of time for LEDs 7 and 8 to light up upon which they flip their appropriate switches (switches 0 and 15). If you flip your switch too early the other player gains a point. If you flip it before your opponent after the LEDs light up you get a point. Press btnU again to play another round but make sure the player switches are off. The board shows the results after each round. A cheat switch (switch 3) displays the remaining amount of time in the time counter. Press btnR to reset the entire system.

With this assignment we were expected to implement a state machine which switched from state to state depending on what stage of the game the system is found in. Most of the source modules were taken from previous labs. The new modules consisted of State_Machine and LFSR, and two new versions of count4L: counter4L (point counter) and Time_Counter. The following diagram illustrates the logic and structure of our system as it was taken from the course page of this assignment

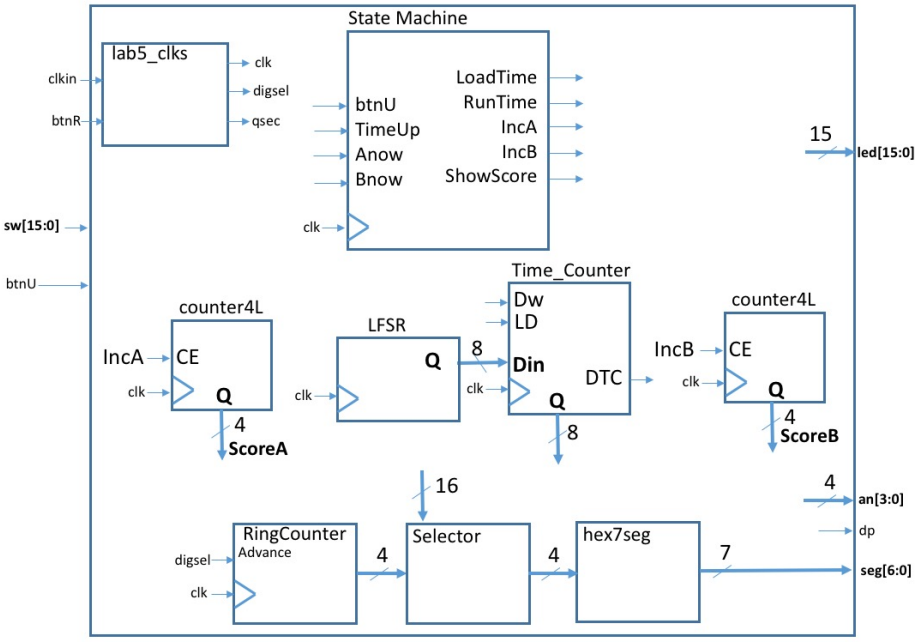


Figure 1: Taken from course online page for CSE 100.

1.1 State_Machine

The state machine for this assignment is meant to switch from states as the game progresses. Below are the following states (with a much in-depth description in the next section)

- Idle: The default state for the game where the score is showed and the game is ready to be played again. This is the initial state.
- Load: The system loads the corresponding semi-random number into the time counter. The state remains as the player presses btnU signal. Once the button is lifted up the time starts running upon the next clock cycle.
- Run: The time counter starts decreasing the loaded time every time the qsec signal goes high. This state remains as such until the timer runs out.
- TimeUp: Once the timer runs out this state lights up LEDs 7 and 8 and waits for a signal to be given by the players.
- AWins: State where A gets a point. Lasts only one clock cycle then jumps back to the Idle state.
- BWins: Same as AWins but instead gives B a point.
- A&BWin: A combination of AWins and BWins where both players get a point.

The following figure describes the behavior of this circuit:

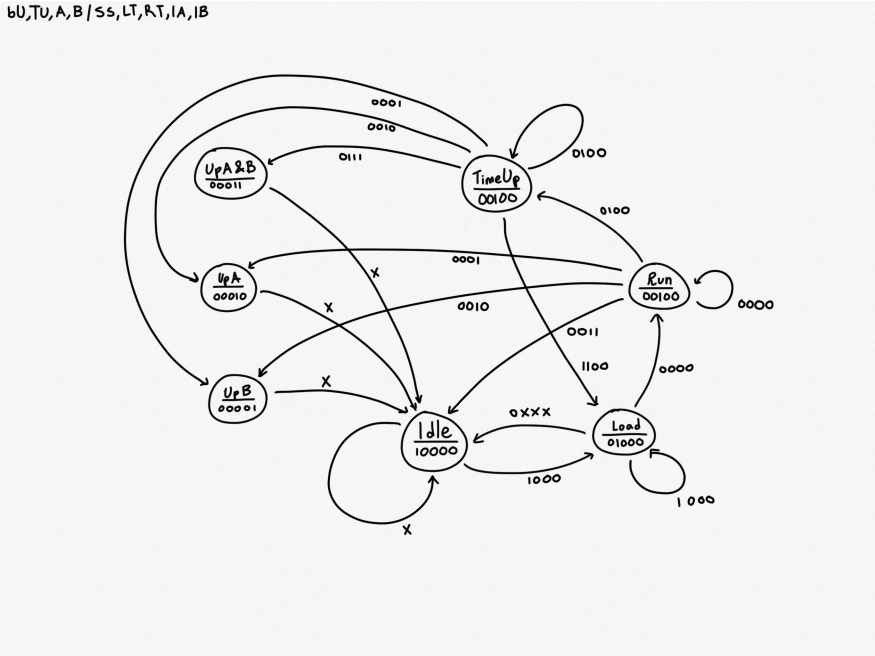


Figure 2: State Diagram for Lab5 State Machine.

1.2 LFSR

The Linear Feedback Shift Register (LFSR) module is a semi-random number generator which is dependent on the previous state and the current time of the system. This module outputs the random time to be loaded into the time counter whenever the Load state is up. The logic for this module was provided to us in the assignment description.

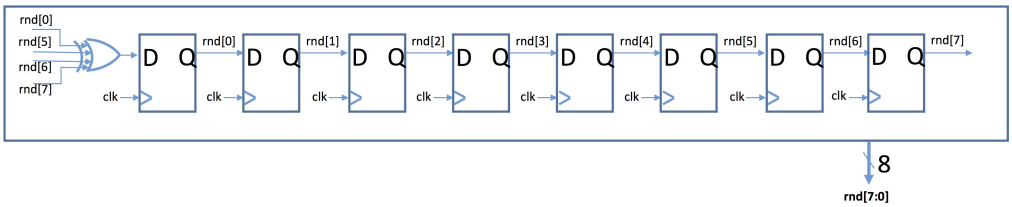


Figure 3: Taken from course online page for CSE 100.

1.3 Player Counter and Time Counter

These two modules were taken directly from the last laboratory assignment with some modifications to fit this assignment. The Time_Counter module now has 6 flip-flops storing the 8-bit 2-digit number (pre-pending to 0s to prevent the random time to be incredibly high). The signal for counting down is the qsec signal which is a slowed oscillating signal derived from clk, and the module can only count down until a DTC signal is given when time runs out. In the other hand, counter4L counts only upwards when the state machine provides either the IncA or IncB signals to increase A or B respectively.

1.4 Top_Module

The logic for the Top_Module was luckily provided to us by **Figure 1**, where all of the inputs and outputs are stated. The system uses a module lab5_clks.v to provide the signals cllk, digsel, and qsec. There are two player counters, each with inputs IncA/Incb, clk and output Q[7:0]. The LFSR module provides the random number which goes as input Din to the Time_Counter module, which itself outputs Q[15:0] (remaining time) and the DTC signal (TimeUp). The outputs from the two playter counters and Time_Counter are sent through a Ring_Counter, Selector, and hex7seg modules to be displayed in the 7-segment display (as explained in previous labs). Finally, the LEDs and AN values are connected to out values to be sent to the Basys3 board.

2 Implementation and Logic

For this section we will only focus on the State_Machine as the new/altered modules have their logic already given to us and that most other modules are taken from previous lab assignments. Also, since the most complex part of the assignment is the State Machine we will focus on its logic and implementation. The implementation of these states will be inspired on **Figure 2** which shows our State Diagram

The format for explaining our inputs will be given in the form XXXX where X are btnU, TimeUp, A, and B respectively.

- Idle: This state remains until 1000 is given. It then switches to the Load state. Any undefined state transitions in our state diagram will be set to the Idle state just in case the system finds itself in an unexpected state. In this state the signal ShowScore is set to high, all others to low.
- Load: When 1000 is provided in Idle state the state machine remains in Load until the inputs change. If 0000 then the system goes into Run, else it goes into Idle for a forbidden input. Both player switches have to be low for the round to start. The signal LoadTime is set to high in this state to allow the Time_Counter to load the time value provided by the LFSR. It is important to mention that any state can jump to the Load state if 1100 or 1000 are provided.
- Run: Once the Load state receives the 0000 inputs it goes into the Run state. Here the signal RunTime is set to high which allows qsec to count down the value in Time_Counter. If 0100 then the state changes to TimeUp; if 0010 or 0101 the state changes to BWins; if 0001 or 0110 the state changes Awins; 0011 (both players lose) and any other inputs send it to the Idle state.
- TimeUp: In this state no outputs are set to high. Since the DTC signal provided by Time_Counter sets TimeUp to high it also lights up LEDs 7 and 8, so this state has no need for any output to be set to high. This state awaits until either player to flip their switches (or a Load or forbidden input is given). As in the Run state, input 0110 switches to the AWins state and input 0101 switches to the BWins state. If 0111 is given then the state switches to A&BWins which can only happen this way.
- AWins: IncA is set to high and the state switches to Idle on the next clock cycle.
- BWins: IncB is set to high and the state switches to Idle on the next clock cycle.
- AWins: Both IncA and IncB are set to high and the state switches to Idle on the next clock cycle.

This logic leads to the following state transition table

		B	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		A	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
		TU	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
P.S.		btnU	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
A	Idle	N.S.	A	A	A	A	A	A	A	A	B	A	A	A	B	A	A	A
B	Load		C	A	A	A	A	A	A	A	B	A	A	A	B	A	A	A
C	Run		C	E	F	A	D	F	E	G	B	A	A	A	B	A	A	A
D	TimeUp		-	-	-	-	D	F	E	G	B	-	-	-	B	A	A	A
E	A wins		-	-	-	-	A	A	A	A	B	-	-	-	B	A	A	A
F	B wins		-	-	-	-	A	A	A	A	B	-	-	-	B	A	A	A
G	A & B win		-	-	-	-	A	A	A	A	B	-	-	-	B	A	A	A

Observe that certain input values are considered impossible or unexpected (denoted -). These inputs will set the state to Idle to avoid errors.

The following table explains the outputs given by each state

		z				
P.S.		SS	LT	RT	IA	IB
A	Idle	1	0	0	0	0
B	Load	0	1	0	0	0
C	Run	0	0	1	0	0
D	TimeUp	0	0	0	0	0
E	A wins	0	0	0	1	0
F	B wins	0	0	0	0	1
G	A & B win	0	0	0	1	1

These transitions and outputs for each input combination lets us deduce our output logic equations in a one hot encoding format.

$$D_0 = A = (\neg b|A|B|T)\&(Q_0|Q_4|Q_5|Q_6|Q_1\&(A|B|T)|Q_2\&(b\&(T|A|B)|(\neg b\&\neg T\&A\&B))|Q_3((\neg b\&\neg T)|b\&(A|B)))$$

$$D_1 = B = b\&\neg A\&\neg B,\ D_2 = C = \neg(b|T|A|B)\&(Q_1|Q_2)$$

$$D_3 = D = \neg(b|\neg T|A|B)\&(Q_2|Q_3),\ D_4 = E = \neg b\&((\neg T\&\neg A\&B\&Q_2)|(T\&A\&\neg B\&(Q_2|Q_3)))$$

$$D_5 = F = \neg b\&((\neg T\&\neg B\&A\&Q_2)|(T\&B\&\neg A\&(Q_2|Q_3)))$$

$$D_6 = G = (\neg b\&T\&A\&B)\&(Q_2|Q_3)$$

$$SS = Q_0,\ LT = Q_1,\ RT = Q_2,\ IA = S_3|S_5,\ IB = S_4|S_5$$

The previous output and next-state logic equations were used to implement the state machine using 7 flip-flops.

For the implementation of this equations in our Verilog project a style decision was made to reduce its complexity and number of gates. By setting $S_0 = \neg(Q_1|Q_2|Q_3)$ instead of the previous S_0 we managed to simplify the project without compromising functionality. Although, this new logic provided certain clock cycles were the system was in two states at once, but only for one cycle. Since the clock cycles very fast it is negligible to consider possible errors. This method also achieves the desired trait that S_0 functions as the default state, which is also activated on unexpected sequences.

3 Conclusion and Comments

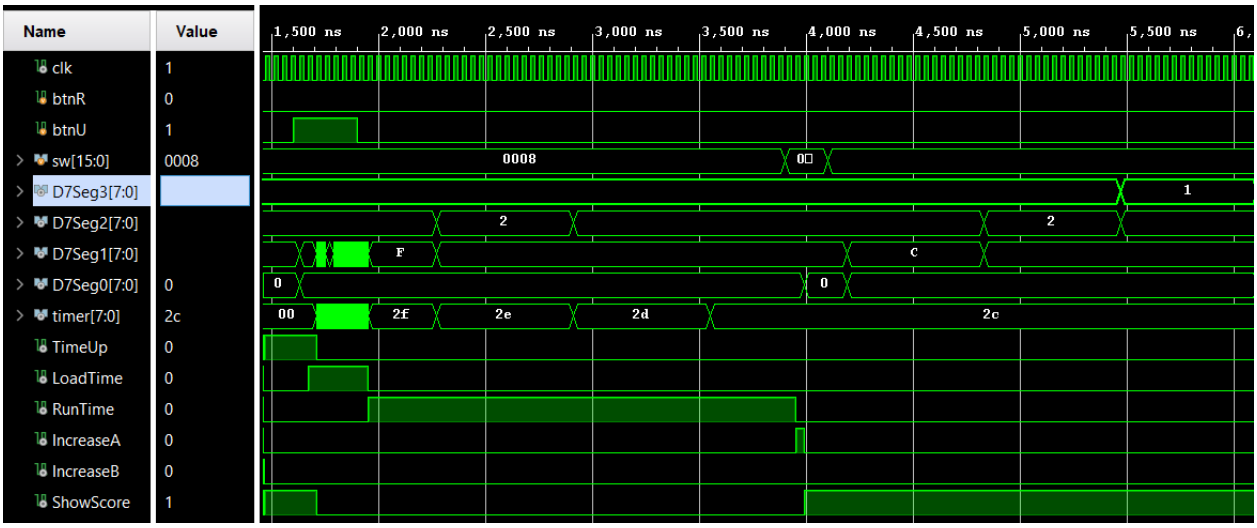
This assignment required a different type of analysis and computation: the creation of a state machine. The tools and techniques taught in class led way into creating the appropriate state machine. The main difficulty of this assignment was to account for all corner cases and all unexpected inputs, as well as determine the logic for D_0, which as showed before is a very complex equation. Since we had to create our own test benches there was an added level of difficulty but a freedom which allowed us to play around with out input and output signals. Finally, even though the state table and functions show streamlined and reduced equations and quantity of states, several processes and computations were made to simplify it significantly.

Appendix

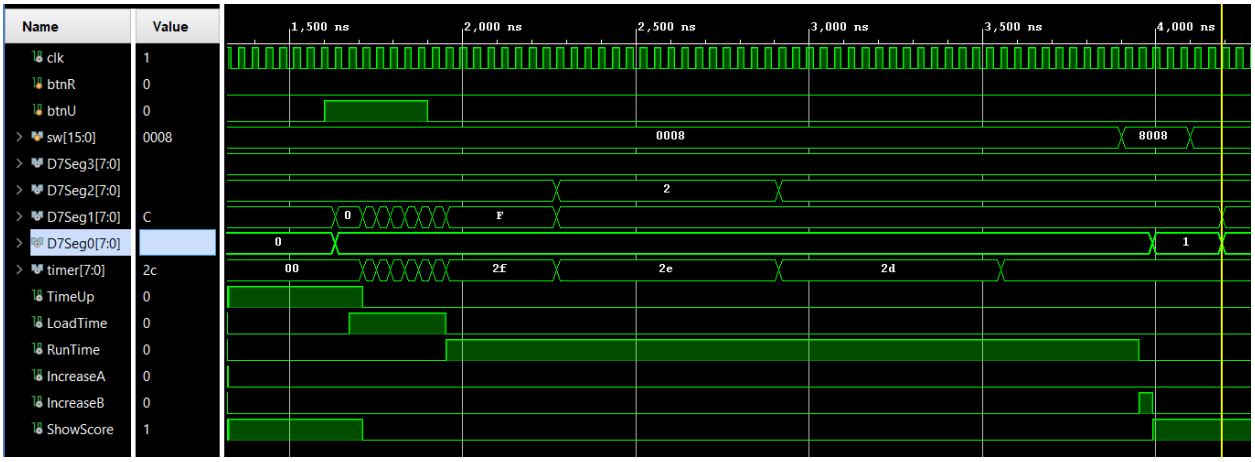
Waveform Viewer Screenshots

(The state output variables are shown as part of the simulation as well as the timer value)

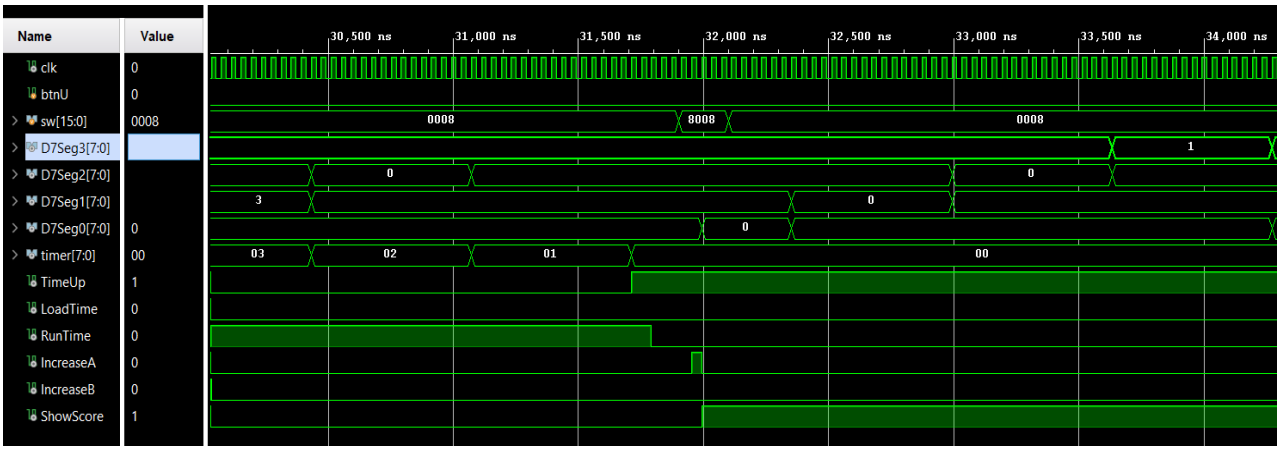
Player A winning by Player B flipping the switch before the green light.



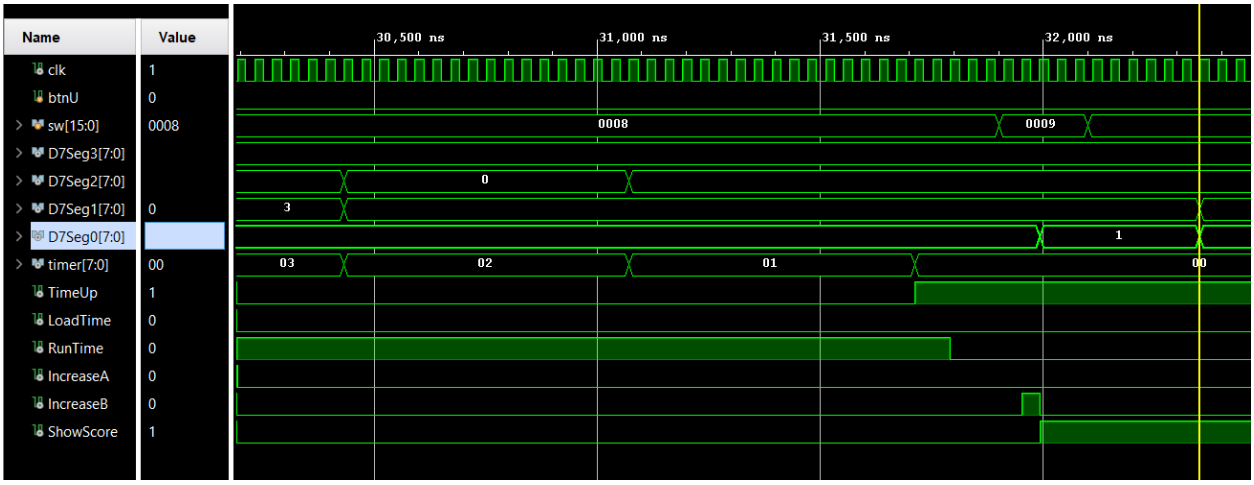
Player B winning by Player A flipping the switch before the green light.



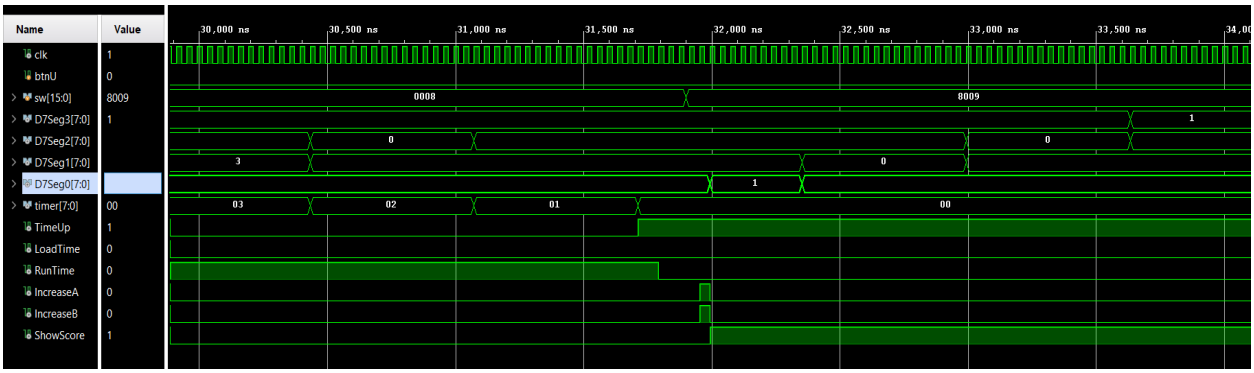
Player A winning by being first after the green light.



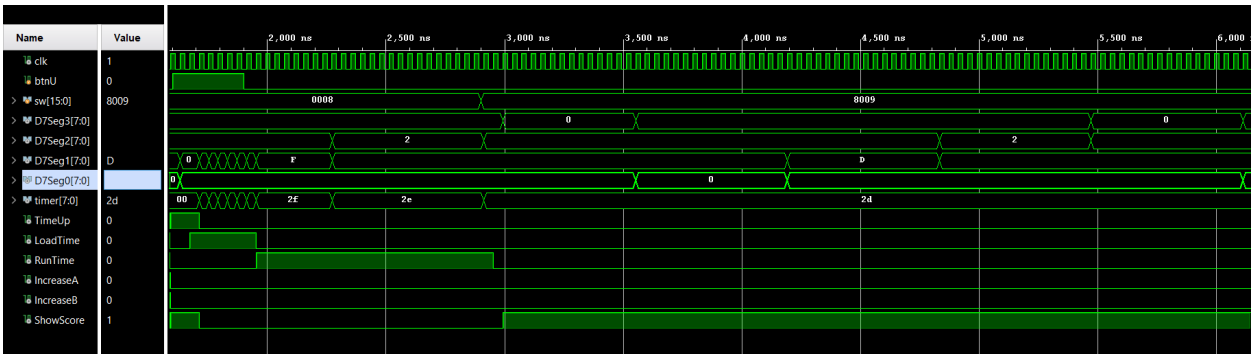
Player B winning by being first after the green light.

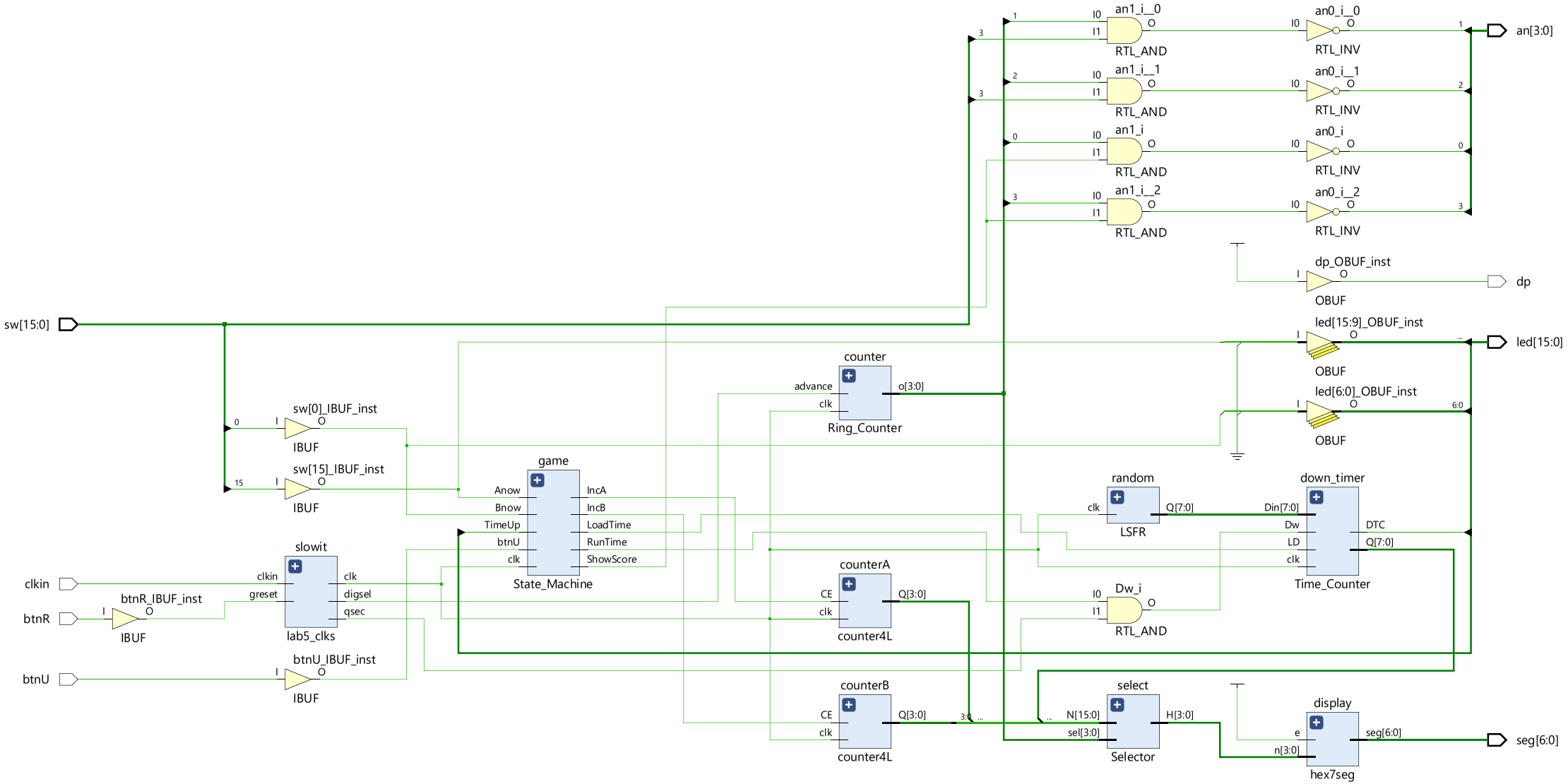


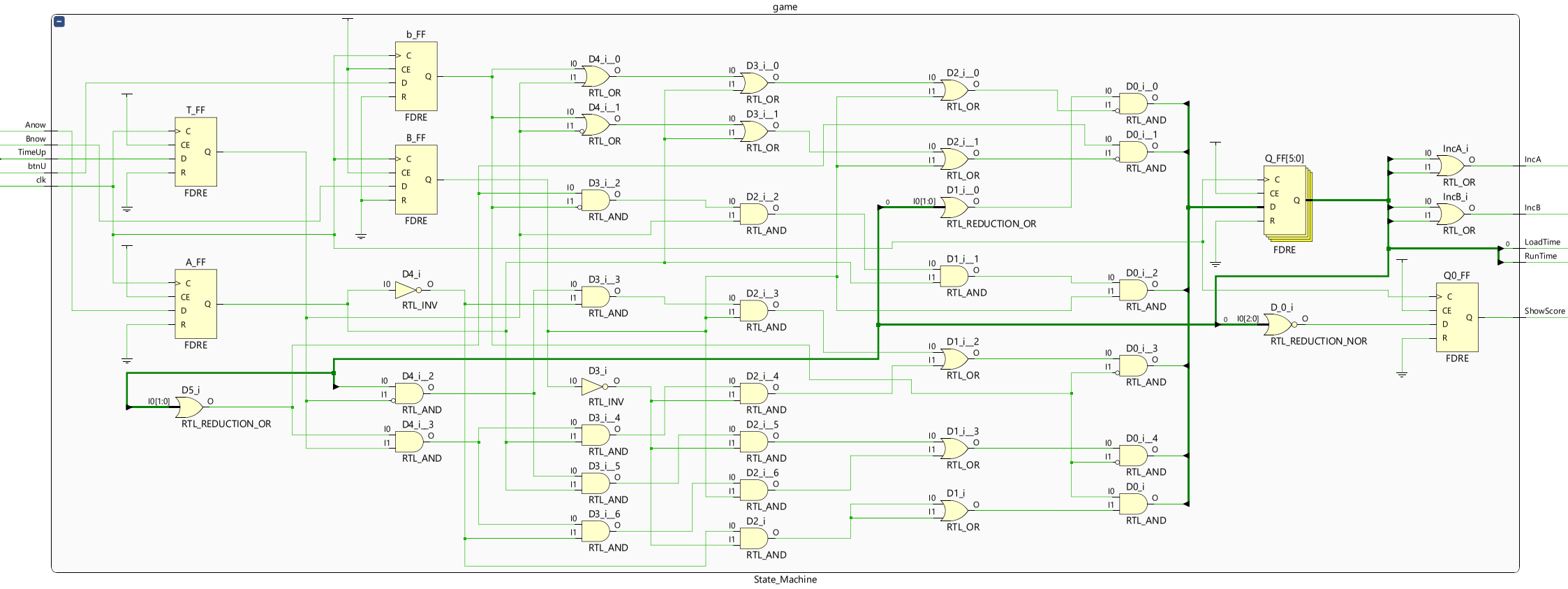
Players A and B both winning by flipping their switches at the same time after the green light.

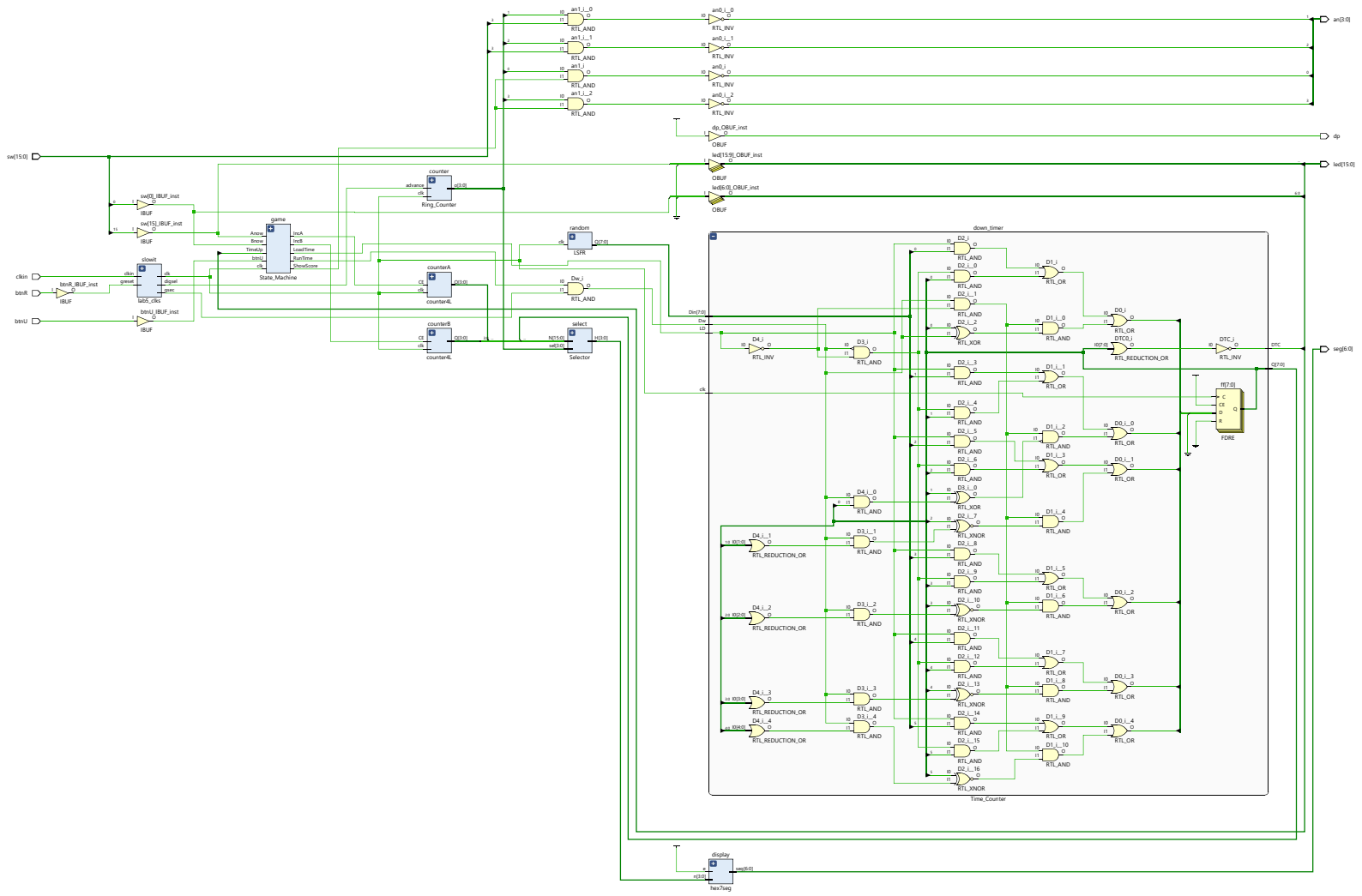


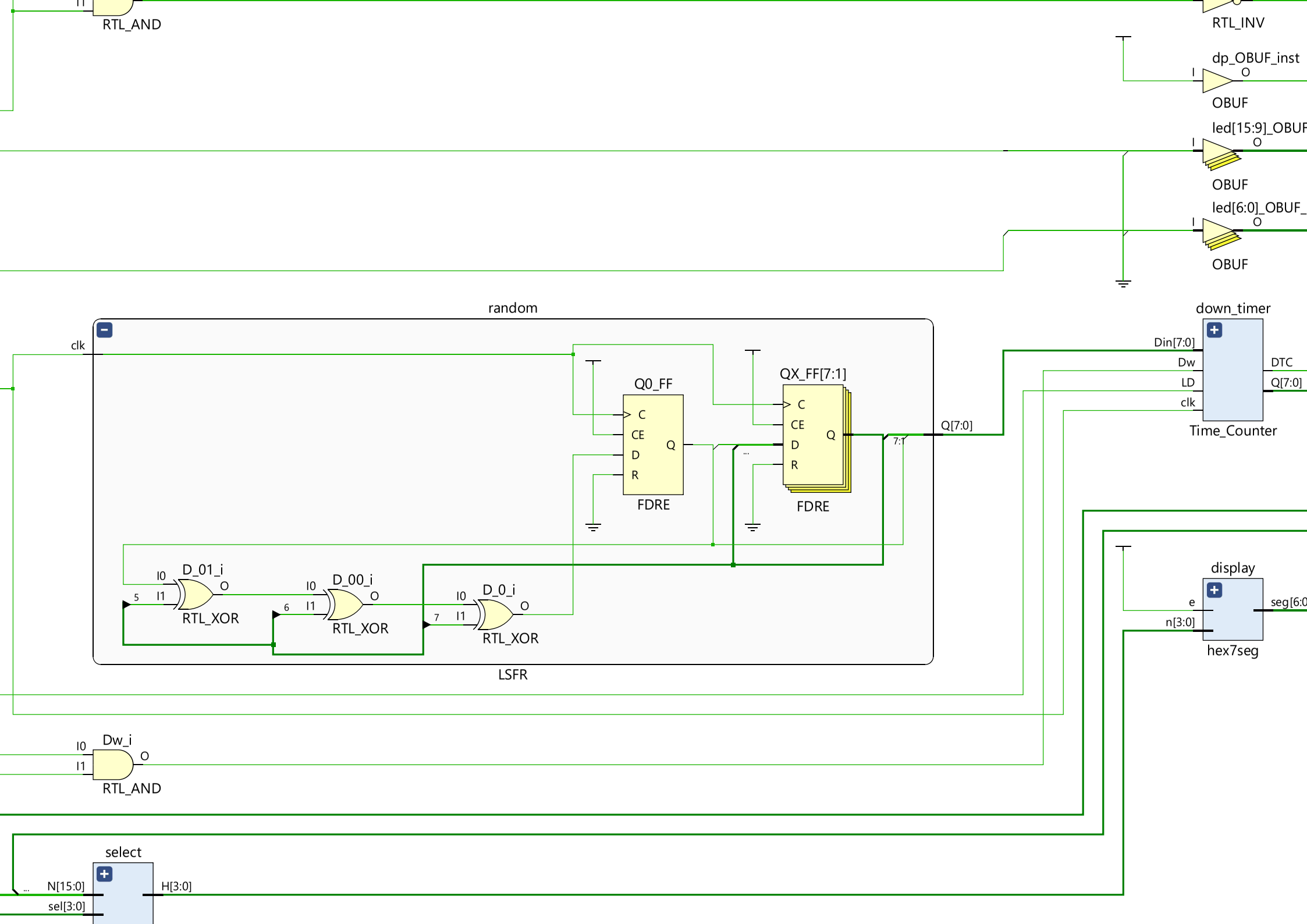
Players A and B both losing by flipping their switches at the same time before the green light.

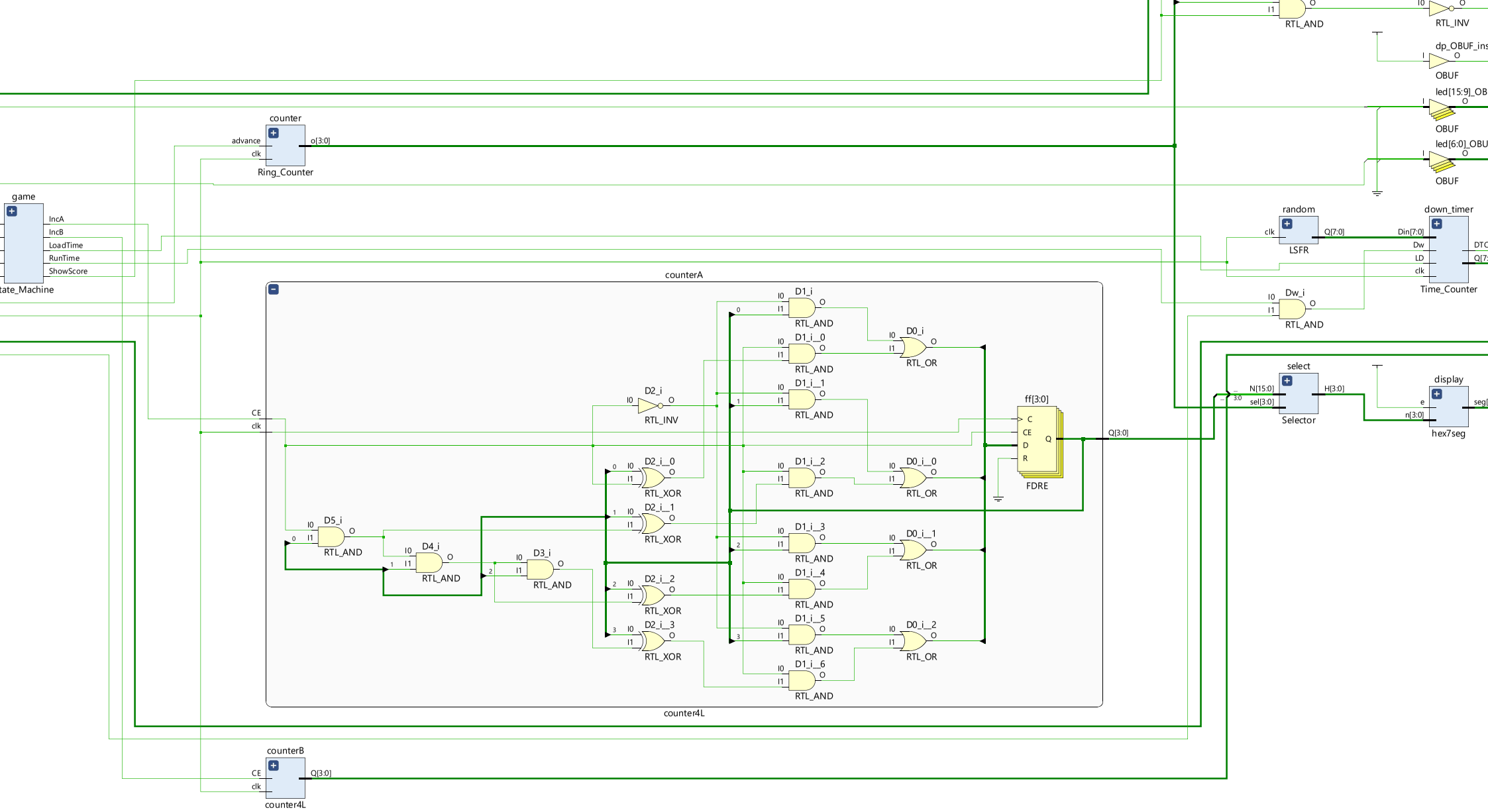



an1_i__0
RTL_AND
an0_i__0
RTL_INV
an[3:0]
an1_i__1
RTL_AND
an0_i__1
RTL_INV
an1_i
RTL_AND
an0_i
RTL_INV
an1_i__2
RTL_AND
an0_i__2
RTL_INV
dp_OBUF_inst
OBUF
dp
sw[15:0]
led[15:9]_OBUF_inst
OBUF
led[15:0]
led[6:0]_OBUF_inst
OBUF
counter
advance
clk
o[3:0]
Ring_Counter
sw[0]_IBUF_inst
IBUF
sw[15]_IBUF_inst
IBUF
game
Anow
Bnow
TimeUp
btnU
clk
IncA
IncB
LoadTime
RunTime
ShowScore
State_Machine
random
clk
Q[7:0]
LSFR
down_timer
Din[7:0]
Dw
LD
clk
DTC
Q[7:0]
Time_Counter
slowit
clkin
greset
clk
digsel
qsec
lab5_clks
clkin
btnR
btnR_IBUF_inst
IBUF
counterA
CE
clk
Q[3:0]
counter4L
Dw_i
RTL_AND
btnU
btnU_IBUF_inst
IBUF
counterB
CE
clk
Q[3:0]
counter4L
select
N[15:0]
sel[3:0]
H[3:0]
Selector
display
e
n[3:0]
seg[6:0]
hex7seg
seg[6:0]


game
Anow
Bnow
TimeUp
btnU
clk
T_FF
FDRE
A_FF
FDRE
b_FF
FDRE
B_FF
FDRE
D4_i
RTL_INV
D5_i
I0[1:0]
RTL_REDUCTION_OR
D4_i_2
RTL_AND
D4_i_3
RTL_AND
D4_i_0
RTL_OR
D4_i_1
RTL_OR
D3_i_2
RTL_AND
D3_i_3
RTL_AND
D3_i
RTL_INV
D3_i_4
RTL_AND
D3_i_5
RTL_AND
D3_i_6
RTL_AND
D3_i_0
RTL_OR
D3_i_1
RTL_OR
D2_i_2
RTL_AND
D2_i_3
RTL_AND
D2_i_4
RTL_AND
D2_i_5
RTL_AND
D2_i_6
RTL_AND
D2_i
RTL_AND
D2_i_0
RTL_OR
D2_i_1
RTL_OR
D1_i_0
I0[1:0]
RTL_REDUCTION_OR
D1_i_1
RTL_AND
D1_i_2
RTL_OR
D1_i_3
RTL_OR
D1_i
RTL_OR
D0_i_0
RTL_AND
D0_i_1
RTL_AND
D0_i_2
RTL_AND
D0_i_3
RTL_AND
D0_i_4
RTL_AND
D0_i
RTL_AND
Q_FF[5:0]
FDRE
D_0_i
I0[2:0]
RTL_REDUCTION_NOR
IncA_i
RTL_OR
IncB_i
RTL_OR
Q0_FF
FDRE
IncA
IncB
LoadTime
RunTime
ShowScore
State_Machine


an1_i_0
RTL_AND
an0_i_0
RTL_INV
an[3:0]
an1_i_1
an0_i_1
an1_i
an0_i
an1_i_2
an0_i_2
dp_OBUF_inst
dp
sw[15:0]
led[15:9]_OBUF_inst
led[15:0]
led[6:0]_OBUF_inst
OBUF
counter
Ring_Counter
sw[0]_IBUF_inst
sw[15]_IBUF_inst
IBUF
game
State_Machine
random
LSFR
down_timer
clkin
btnR_IBUF_inst
btnU
btnU_IBUF_inst
slowit
counterA
counterB
counter4L
Dw_i
select
Selector
Dw
LD
clk
D4_i
D3_i
D2_i
D1_i
D0_i
RTL_XOR
RTL_XNOR
RTL_OR
RTL_REDUCTION_OR
DTC_i
DTC
Q[7:0]
ff[7:0]
FDRE
Time_Counter
display
hex7seg
seg[6:0]


RTL_AND
RTL_INV
dp_OBUF_inst
I
O
OBUF
led[15:9]_OBUF
OBUF
led[6:0]_OBUF_
OBUF
random
clk
Q0_FF
C
CE
D
R
Q
FDRE
QX_FF[7:1]
FDRE
Q[7:0]
7:1
D_01_i
D_00_i
D_0_i
I0
I1
O
5
6
7
RTL_XOR
RTL_XOR
RTL_XOR
LSFR
down_timer
Din[7:0]
Dw
LD
clk
DTC
Q[7:0]
Time_Counter
display
e
n[3:0]
seg[6:0]
hex7seg
Dw_i
RTL_AND
select
N[15:0]
sel[3:0]
H[3:0]


counter
advance
clk
o[3:0]
Ring_Counter
game
IncA
IncB
LoadTime
RunTime
ShowScore
tate_Machine
counterA
CE
clk
D5_i
D4_j
D3_i
D2_i
D2_i__0
D2_i__1
D2_i__2
D2_i__3
D1_i
D1_i__0
D1_i__1
D1_i__2
D1_i__3
D1_i__4
D1_i__5
D1_i__6
D0_i
D0_i__0
D0_i__1
D0_i__2
RTL_AND
RTL_INV
RTL_XOR
RTL_OR
ff[3:0]
C
CE
D
R
Q
FDRE
Q[3:0]
counter4L
counterB
CE
clk
Q[3:0]
counter4L
RTL_AND
RTL_INV
dp_OBUF_ins
OBUF
led[15:9]_OB
OBUF
led[6:0]_OBU
OBUF
random
clk
Q[7:0]
LSFR
down_timer
Din[7:0]
Dw
LD
clk
DTC
Q[7:
Time_Counter
Dw_i
RTL_AND
select
N[15:0]
sel[3:0]
H[3:0]
Selector
display
e
n[3:0]
seg[
hex7seg

```
`timescale 1ns / 1ps

module Top_Module(
    input clkin,
    input btnR,
    input btnU,
    input [15:0] sw,
    output [15:0] led,
    output [3:0] an,
    output dp,
    output [6:0] seg
    );

    wire [15:0] Q;
    wire clk, digsel, qsec, Dw;
    wire [3:0] ScoreA, ScoreB, sel, H;
    wire [7:0] rnd;
    wire TU, LT, RT, IA, IB, SS;

    lab5_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk),
.digsel(digsel), .qsec(qsec));

    counter4L counterA(.CE(IA),.clk(clk),.Q(Q[15:12]));
    counter4L counterB(.CE(IB),.clk(clk),.Q(Q[3:0]));

    assign Dw = RT & qsec;

    LSFR random(.clk(clk),.Q(rnd));
    Time_Counter
down_timer(.clk(clk),.LD(LT),.Dw(Dw),.Din(rnd),.DTC(TU),.Q(Q[11:4]
));
    State_Machine
game(.btnU(btnU),.TimeUp(TU),.Anow(sw[15]),.Bnow(sw[0]),.LoadTime(
LT),.RunTime(RT),.clk(clk),.IncA(IA),.IncB(IB),.ShowScore(SS));

    Ring_Counter counter(.clk(clk), .advance(digsel), .o(sel));
    Selector select(.sel(sel), .N(Q), .H(H));
    hex7seg display(.n(H), .seg(seg), .e(1));

    assign an[0] = ~(sel[0] & SS);
```

```
    assign an[1] = ~(sel[1] & sw[3]);
    assign an[2] = ~(sel[2] & sw[3]);
    assign an[3] = ~(sel[3] & SS);

    assign dp = 1;
    assign led[6:1] = 6'b000000;
    assign led[14:9] = 6'b000000;
    assign led[7]  = TU;
    assign led[8]  = TU;
    assign led[0]    = sw[0];
    assign led[15]   = sw[15];
endmodule
```

```
`timescale 1ns / 1ps

module State_Machine(
    input btnU,
    input TimeUp,
    input Anow,
    input Bnow,
    input clk,
    output LoadTime,
    output RunTime,
    output IncA,
    output IncB,
    output ShowScore
    );

    wire [6:0] D;
    wire [6:0] Q;
    wire b, A, B, T;

    FDRE #(.INIT(1'b1)) A_FF (.C(clk), .CE(1'b1), .D(Anow),
.Q(A));
    FDRE #(.INIT(1'b1)) B_FF (.C(clk), .CE(1'b1), .D(Bnow),
.Q(B));
    FDRE #(.INIT(1'b1)) b_FF (.C(clk), .CE(1'b1), .D(btnU),
.Q(b));
    FDRE #(.INIT(1'b1)) T_FF (.C(clk), .CE(1'b1), .D(TimeUp),
.Q(T));

    assign D[0] =  ~|Q[3:1];                    // Idle
    assign D[1] =  b&(~A&~B|~A&~B);             // Load
    assign D[2] =  (|Q[2:1]) & ~( b| T|A|B);    // Run
    assign D[3] =  (|Q[3:2]) & ~( b|~T|A|B);    // Time Up
    assign D[4] =  (|Q[3:2]) &~b & T & A & B;   // Win Tie
    assign D[5] =  ((Q[2]  &~T &~A & B) |       // A wins
                   (|Q[3:2]& T & A &~B)) & ~b;
    assign D[6] =  ((Q[2]  &~T & A &~B) |       // B wins
                   (|Q[3:2]& T &~A & B)) & ~b;

    assign RunTime   = Q[2];
    assign LoadTime  = Q[1];
```

```
    assign ShowScore = Q[0];
    assign IncA      = Q[4] | Q[5];
    assign IncB      = Q[4] | Q[6];

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(D[0]),
.Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q_FF[5:0] (.C({6{clk}}), .CE({6{1'b1}}),
.D(D[6:1]), .Q(Q[6:1]));

endmodule
```

```
`timescale 1ns / 1ps

module Time_Counter(
    input Dw,
    input LD,
    input [7:0] Din,
    input clk,
    output [7:0] Q,
    output DTC
    );

    wire [7:0] D;
    assign D[0] =   (LD &  Din[0])|
       (~Dw & ~LD & Q[0]) |
       (Dw & ~LD & (Q[0] ^ Dw));
    assign D[1] =   (LD &  Din[1]) |
       (~Dw & ~LD & Q[1]) |
       (Dw & ~LD &~(Q[1] ^ (Dw & Q[0])));
    assign D[2] =   (LD &  Din[2]) |
       (~Dw & ~LD &  Q[2]) |
       ( Dw & ~LD & (Q[2]^~(Dw & |Q[1:0]))) ;
    assign D[3] =   (LD &  Din[3]) |
       (~Dw & ~LD &  Q[3]) |
       ( Dw & ~LD & (Q[3]^~(Dw & |Q[2:0]))) ;
    assign D[4] =   (LD &  Din[4]) |
       (~Dw & ~LD &  Q[4]) |
       ( Dw & ~LD & (Q[4]^~(Dw & |Q[3:0]))) ;
    assign D[5] =   (LD &  Din[5]) |
       (~Dw & ~LD &  Q[5]) |
       ( Dw & ~LD & (Q[5]^~(Dw & |Q[4:0]))) ;
    assign D[7:6] = 0;

    FDRE #(.INIT(1'b1) ) ff[7:0] (.C({8{clk}}), .R(),
.CE({8{1'b1}}), .D(D[7:0]), .Q(Q[7:0]));

    assign DTC = ~(|Q[7:0]);

endmodule
```

```
`timescale 1ns / 1ps
module counter4L(
    input CE,
    input clk,
    output [3:0] Q
    );

    wire [3:0] D;

    assign D[0] =
       (~CE &  Q[0]) |
       ( CE & (Q[0]^ CE));
    assign D[1] =
       (~CE &  Q[1]) |
       ( CE & (Q[1]^ (CE & Q[0])));
    assign D[2] =
       (~CE &  Q[2]) |
       ( CE & (Q[2]^ (CE & Q[0] & Q[1])));
    assign D[3] =
       (~CE &  Q[3]) |
       ( CE & (Q[3]^ (CE & Q[0] & Q[1] & Q[2])));

    FDRE #(.INIT(1'b0) ) ff[3:0] (.C({4{clk}}), .R(),
.CE({4{CE}}), .D(D), .Q(Q));


endmodule
```

```
`timescale 1ns / 1ps

module LSFR(
    input clk,
    output [7:0] Q
    );

    wire [7:0] D;

    assign D[0] =   Q[0]^Q[5]^Q[6]^Q[7];
    assign D[1] = Q[0];
    assign D[2] = Q[1];
    assign D[3] = Q[2];
    assign D[4] = Q[3];
    assign D[5] = Q[4];
    assign D[6] = Q[5];
    assign D[7] = Q[6];

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(D[0]),
.Q(Q[0]));
    FDRE #(.INIT(1'b0)) QX_FF[7:1] (.C({7{clk}}), .CE({7{1'b1}}),
.D(D[7:1]), .Q(Q[7:1]));

endmodule
```