Lab 5: Bak Bak Bak Bak!

Gavin Chen

CE 100 UCSC Spring 2018

Description:

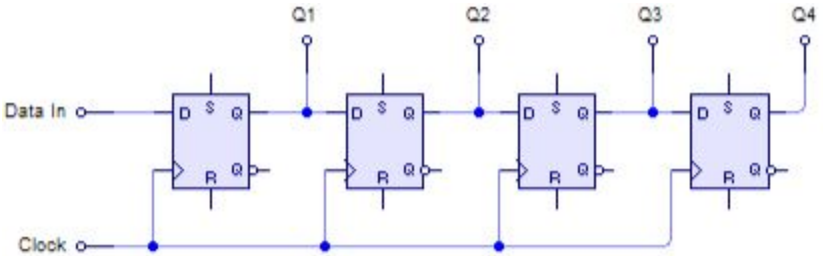
This lab familiarized us with two concepts: state machines and combinational logic. We implemented a quick game that transitioned between states, and interacted with the user with onboard buttons and switches. All of the logic was computed as a function of the current state and the given input, generating a value that corresponded to different actions or transitions to implement the game.

Methods:

The aim of the lab was to implement a combinational logic version of the game chicken. There were several parts to the lab consisting of auxiliary components to interface with the state machine. Since the state machine would ideally only coordinate the use of components through the use of combinational logic, its output pins simply activated different units of the hardware.

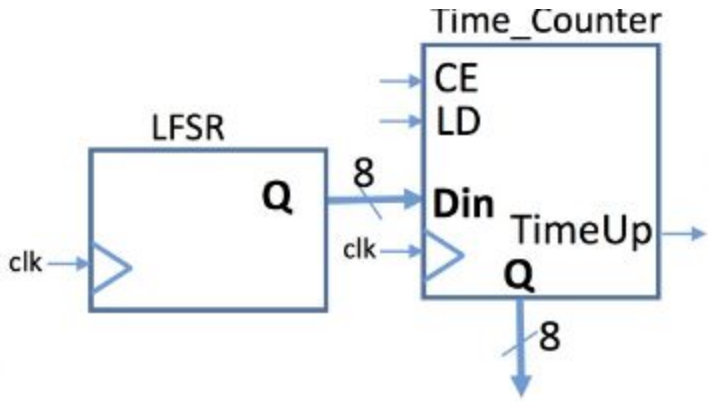
LED shifter:

We began with the shifter that was used to indicate a countdown sequence. The hardware was simply a shift register with a width of 16 bits to correspond to 16 LEDs. An example of the circuit drawn from wikipedia is shown to the right. A series of 16 flip flops were cascaded together, and at the start all flip flops were initialized to 0 and their clocks and reset lines tied together. The input of the leftmost flop was then wired to VCC to feed in a steady stream of 1's, and the enable pins were all combined together and set as an input into the module. The result was a 16'bit shift register that grew 1's in step with the input line, and which could be reset in a clock cycle through use of the shared reset line.



4-bit counter:

The next component we focused on was a 4-bit counter with the ability to count up or down given an input to the corresponding line. The logic implemented in the counter was the exact same as that implemented in the previous lab, only modified to 4 bits. The resulting block had the ability to count up or down given an external input, and was synchronized to the rest of the circuits through use of a clock. Individual counters were used to keep track of each players score. Single inputs from the combinational logic could then increment the respective winners counter per round.



Time counter:

2 such counters were then cascaded together to create an 8-bit counter to keep track of the score. The up/down count lines from the first counter was tied into inputs of the subsequent one. Clear, load, and a 16-bit output bus were added to the counter to interface with other modules. Finally, a time up line was added which

was simply the inversed OR of all the bits in the outputs. Through ORing the bits the output was driven high only in the event the counter had finished i.e. reached all zeros. A randomness functionality was then added to load in constantly changing start values. A Linear Feedback Shift Register was created by cascading 8 flip flops together and initializing the first to a 1. Outputs of flops 0, 5, 6, and 7 were then XORed together and fed back into flop 0. The resulting output formed an 8 bit bus with changing values, which was then interpreted as a load in value for the counter. To set an upper bound on possible load in values the MSB of the output was set permanently to zero. The resulting bus was fed into the counter as a loadable input. Load, reset, and time up were all sent to interface with the state machine, while the output bus was wired to the display. The resulting module was then reduced to a time counter which advanced and loaded in new values on signal from the state machine, producing an 8 bit number and a signal to specify when the timer has expired. A rough schematic from the lab manual is presented to the upper left.

Overhead module:

An overhead module was created as a buffer between the user input and input to the state machine. It consisted of 4 instances of a feedback flip flops. In each instance, an external input was fed into a flip flop, and the resulting output was then ORed back in as input. The result was an output that would stay on given a single transient event. 2 instances were used as an intermediaries for the button up/down inputs corresponding to player moves. Since only the first pressing of the button was taken as a transition event, it made the state machine logic simpler to keep respective button input high given a single event. The same logic was applied in reverse to make the flashing effect for the winner. Since the state machine only output the winner for a single cycle, that output was input into feedback flip flops that kept the winner bit high until reset on the next round. The overall module took in 4 external input and was synchronized to a clock and had a reset option. The resulting 4 outputs were then fed either into the state machine or to the segment display.

Segment display:

The same components from the previous lab were used to implement the segment display: a ring counter, 16-bit field selector, and a 7 segment display converter. All components were imported from the previous lab and implemented in the same fashion. The difference in operation during this lab was in the logic on the anode connection of the segment display. Logic was implemented to display digits only when specified by the state machine.

State Machine:

The state machine was implemented using 8 distinct states with 21 different transitions in a hybrid Mealy/Moore format. To make the combinational logic simpler, one-hot encoding was used to maintain the state.The behaviour of the state machine depended on 5 inputs shown to the left. Given the current state and an input event, states were changed in accordance with the game. A table of the transition logic for the 8 states is presented in the table below. Logic statements were assigned to wires and fed into the D input of the respective flip flop. All flops were enabled on each clock cycle

Input	Description
btnC	Begin game
btnU	Player 1 input
btnD	Player 2 input
Led	Finished final countdown
Tup	Game time is up

and state changes took place exactly one cycle after the combinational logic. Finally, given these states and their transitions, the output of the state machine was calculated as a combination of Mealy and Moore state machine logic. The output pin logic is also presented below. Given this state machine, we gained the ability to control the hardware modules through selective pin signals.

State	Flip flop	Input logic
Idle	Q0	Q0 btnC+Tup(Q2+Q5+Q6+Q7)
Count	Q1	Q0 btnC+Q1Led
Poll	Q2	Q1 Led+Q2 btnU btnD Tup
P1 winning	Q3	btnU btnD Tup(Q3+Q2)
P2 winning	Q4	btnU btnD Tup(Q4+Q2)
P1 won	Q5	Q3 Tup+Tup(Q5+Q4 btnUbtnD)
P2 won	Q6	Q4 Tup+Tup(Q6+Q3 btnU btnD)
Both won	Q7	Tup(Q7+Q2 btnU btnD

Output pin	Logic
Reset	Q0 btnC
Show score	Q0
Increment up	Tup(Q5+Q7)
Increment down	Tup(Q6+Q7)
Load time	Q0btnC
Show time	Q0 btnC+Q1Led
Run time	Q2+Q3+Q4+Q5+Q6+Q7

Top module:

The final step was combining all pieces together in the top module to interface with each other as well as with the on board hardware. Nearly all components took input from the state machine and output data that signaled a transition, with the notable exception of the player score keepers.

All button input was first first buffered through the overhead module, except for the center button input. The LED and Tup input came from the LED shifter and time counter, respectively. Given these inputs the state machine knew the status of the overall circuit, and controlled the inputs to coordinate the game. The reset and increment pins were connected to the overhead module, as well as the scoreboard counters. Runtime and Load time were connected into the timer module as needed, and Show time was connected to the enable pin of the Led shifter. Since showing and flashing the score and displaying the count time were dependent on the segment display anode, those functions were implemented with direct logic in the top module. First a half second oscillating signal was created by using an inverting flip flop that switched on quarter seconds, meaning it was high for every half second. It was then decided that the best way to implement these functions was by assigning the anode associated with the game time and score to logic involving the pins. The following was derived for the flashing and time display:

Flashing score: $an[n] = Showscore \ \& \ ring[n] \ \& \ (1 \oplus (flash \ \& \ halfsec))$

Show time: $an[n] = Showtime \ \& \ ring[n]$

Where an[n] represented the segments for score and time, i.e. (3,0) and (2,1) respectively. Finally a clock module was added and all components were synchronized.

Results:

Shifter:

The shifter module to control the LED's worked as expected. By having the LEDs shift by one bit per quarter second, the led stage persisted for 16 led cycles i.e. 4 seconds. By having the last led control the reset state of all flip flops, it also helped to maintain the led module as a self contained component. With only one input to let the leds run, and one output to signify its completion, the design and implementation with the state machine was much simpler and straight forward.

Scoreboard (counter):

The scoreboard 4 bit counter also only took in a single input from the state machine, making it a much simpler design. Since there was no need to reset the counter, only an input to increment the counter was needed for operation. This made it somewhat more difficult to implement the player win state. Since the increment counter and flash winner were both essentially outputs of the same type, only 2 output bits were used for both flashing and incrementing the players. It was decided that a single on signal for the winning player was easier to implement than a constant signal. Therefore the output of the state machine could be directly connected to the counter.

Overhead module:

The module was needed to conform input with the state machine logic. Since the button input was registered only once, a module was created to keep the button state high for the duration of the game, if pressed. The same idea was applied to the output controlling the flash winner functionality. Since a winner line was only driven high for a single cycle, a memory flip flop was implemented to keep the flash line high to conform with the other component logic.

Time counter:

The time counter combined the random input generator and counter into a single module to make use simple. The module took in a run line that enabled countdown and stopped if it was ever low. The load input was used to keep track of the ability to input a fresh random value when needed for the next round. Finally the time and timeup outputs were wired to the display and state machine respectively. Having the behaviour of the time counter controlled by single input lines made its behaviour easily manageable by the state machine.

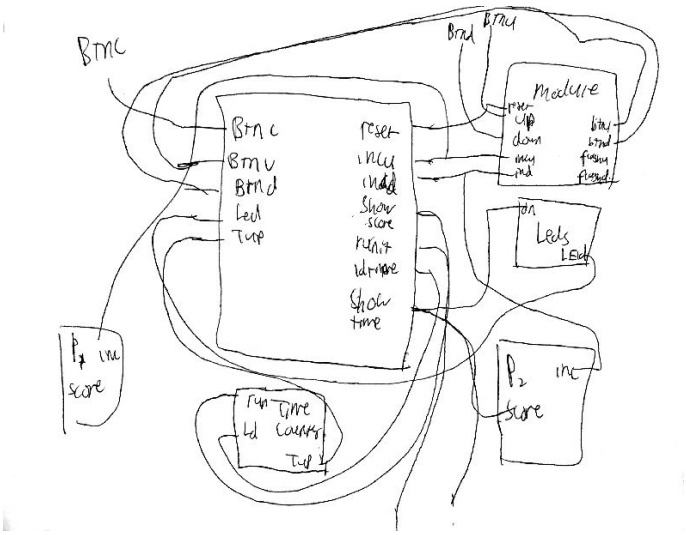
Segment display:

The display module to interface with the board was already created in a previous lab. In this case, the only difference in behaviour was with the ability in flashing and showing different numbers. The logic for the segment display was unchanged, the only thing changed to show the digits was in the anode activation. The logic and output for the digits were constantly being computed, however the only difference was in whether the anode corresponding to the digit was activated or not.

State Machine:

The state machine consisted of 8 states. It began in the idle state, which displayed the score of both players and activated the load time function of the time counter, remaining in that state while the

center button was not pressed. From there it could only transfer to countdown on the event the center button was pressed. The state machine stayed in the countdown state while the last LED in the countdown was not activated. While in countdown the state machine output high for the showtime pin, keeping the time on screen and the leds shifting. Upon completion of the LEDs the state transferred to polling, outputting a reset signal to the flip flop module to reset the flash and button conditions in the transfer. This began the game. The great game was played while time was not up or there was no winner. The machine remained in the polling state while buttons were low and time had not expired. When time expired it transferred directly back to idle with no winner output. If the up or down button was pressed it transferred to the p1 winning or p2 winning state, respectively. When both buttons were pressed at once control was transferred to the both state. In the p1 winning state it remained until the down button was pressed or time had expired. In the case of a button press it transferred to the p2 won state. In the case of a time expire it transferred to the p1 won state. The same logic was implemented in the p2 winning state, when the up button was pressed it transferred to the p1 won state, or when time expired it transferred to the p2 won state. In the p1 won state an increment signal for p1 was sent and the machine transferred control back to the idle state. In the p2 won state an increment signal was sent for p2 and control also sent to idle. For the case where both won the increment signal for both players was sent and control sent back to the start state. From there the process repeated until the world froze.



Top level:

A schematic of the top level is presented to the left. No logic apart from the anode activation was implemented in the top level. The top level existed mainly as a way to connect all components together in the correct fashion. Nearly all components required feedback into the state machine for the overall hardware to function properly. The only components that weren't fed back into the state machine were the score counters. The module module provided buffered connection into the state machine to maintain state, and transformed signal pulses into constant output. The main role of all components were to serve as wrapper modules for the logic implemented in the state machine, so that input and output could be managed separately from the combinational logic. The top level worked well in the end, after several iterations.

Simulation and testing:

We created several testing modules to observe the output of each component before connecting. The first module tested was the state machine. Assuming expected behaviour from the components, we tested that the machine went through the correct logic and states. There were some errors when attempting to choose the winner, especially between switching states, however the issue was resolved. The next modules to be tested were the interface module, score module, LED shifter, and time counter. All

modules needed slight adjustment to the logic before full functionality. However the desired behaviour was eventually created.

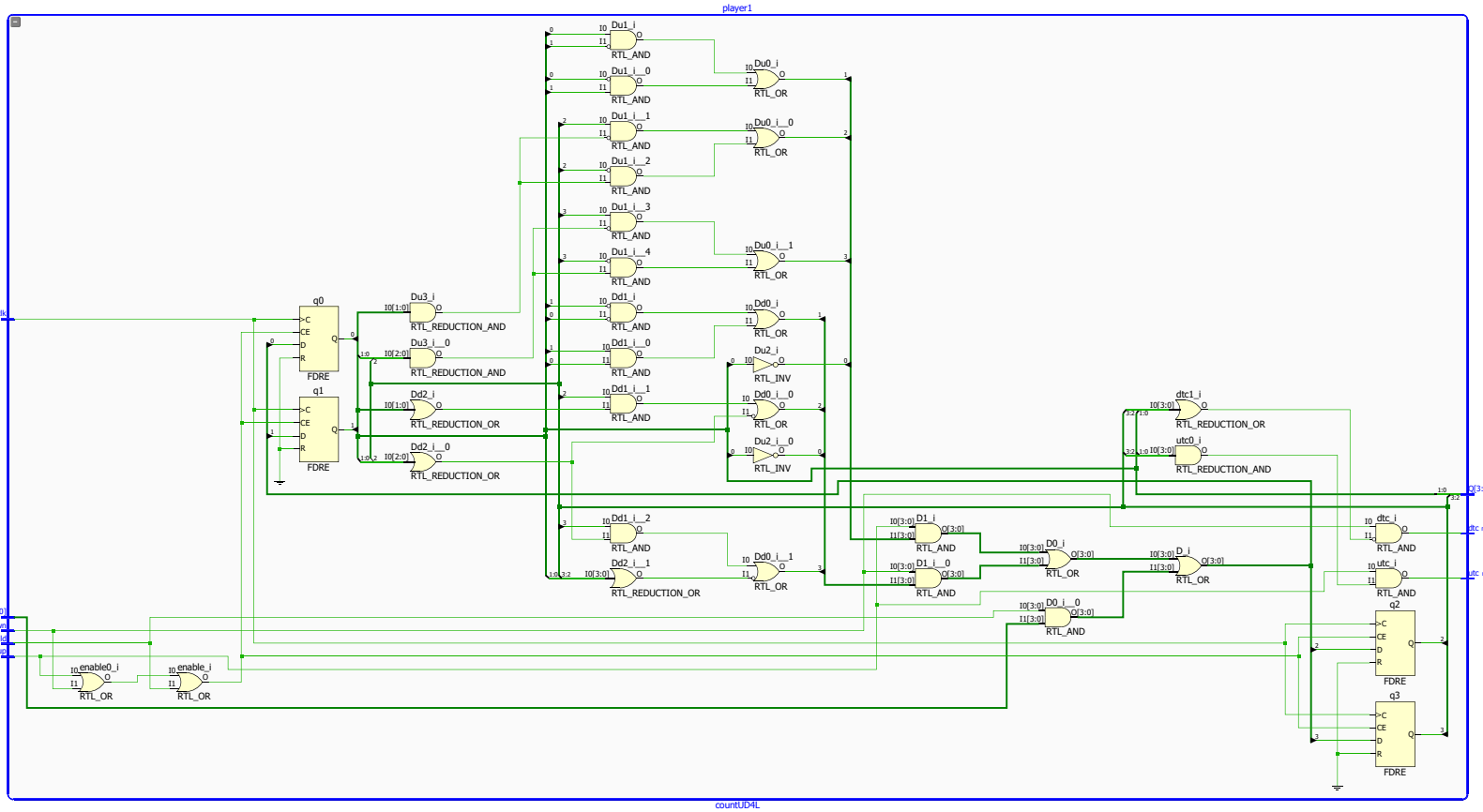
Conclusion:

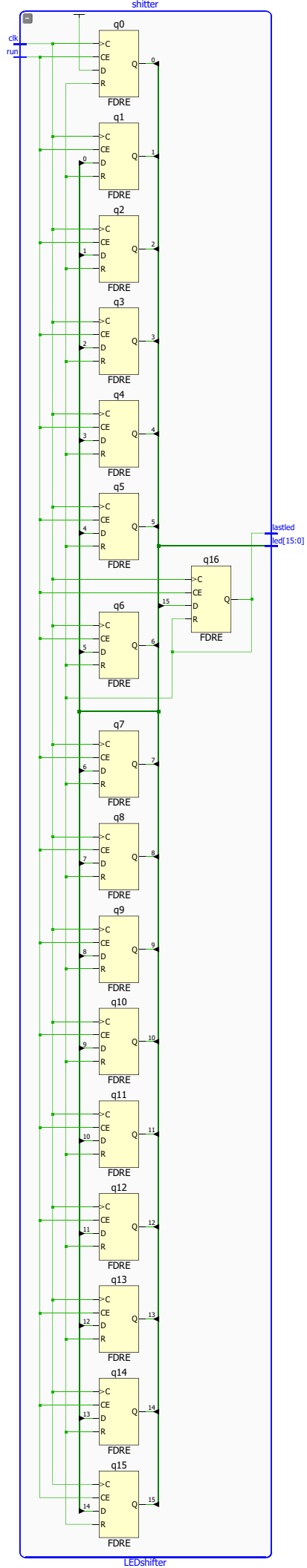
The lab was interesting and we got a good understanding of state machines. We got familiar with implementing combinational logic and creating components that could interface with the combinational logic. In the end we got the lab to work after a lot of work, and gained a game to play for years to come with family and friends yielding much fun and laughs!

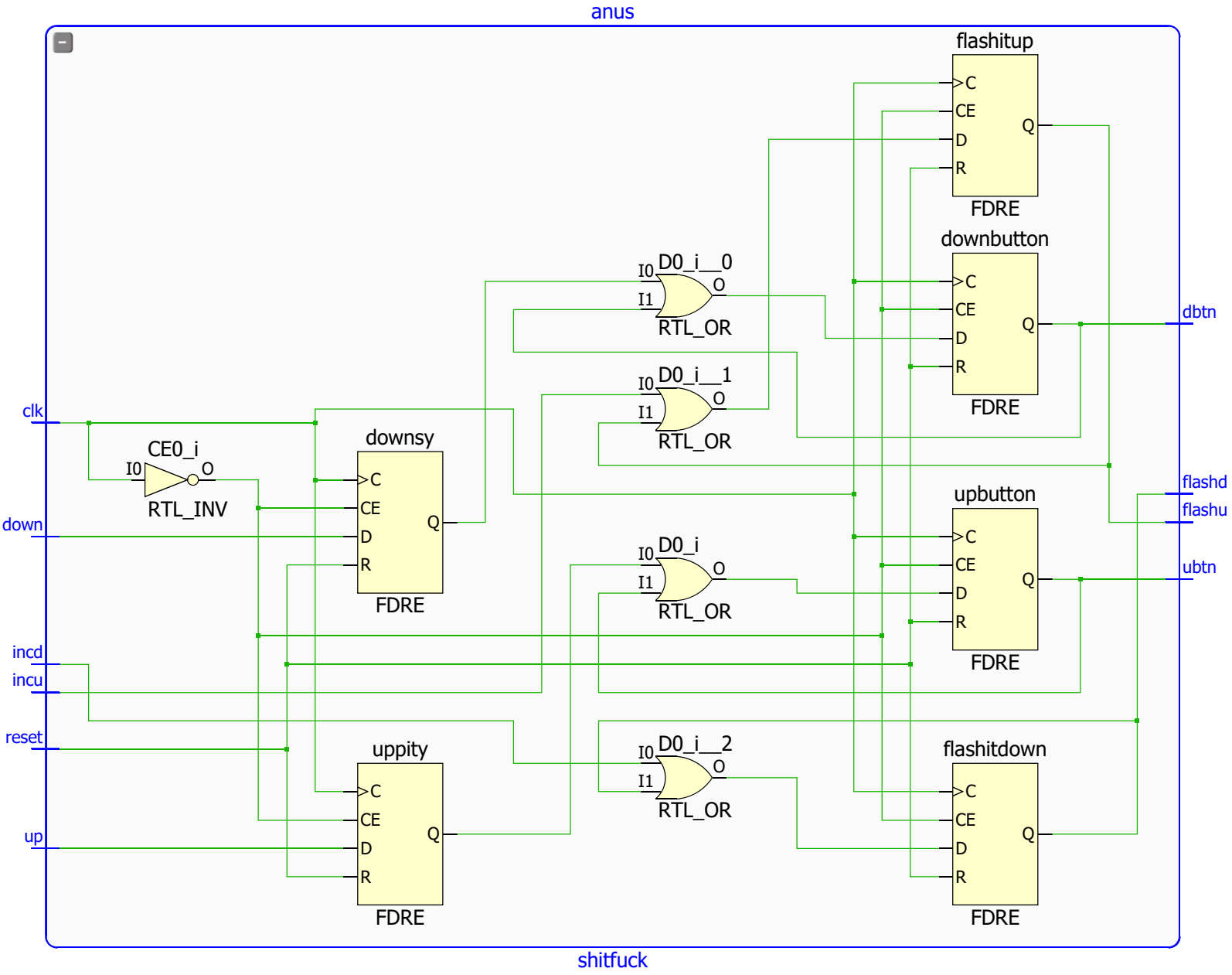
Works cited

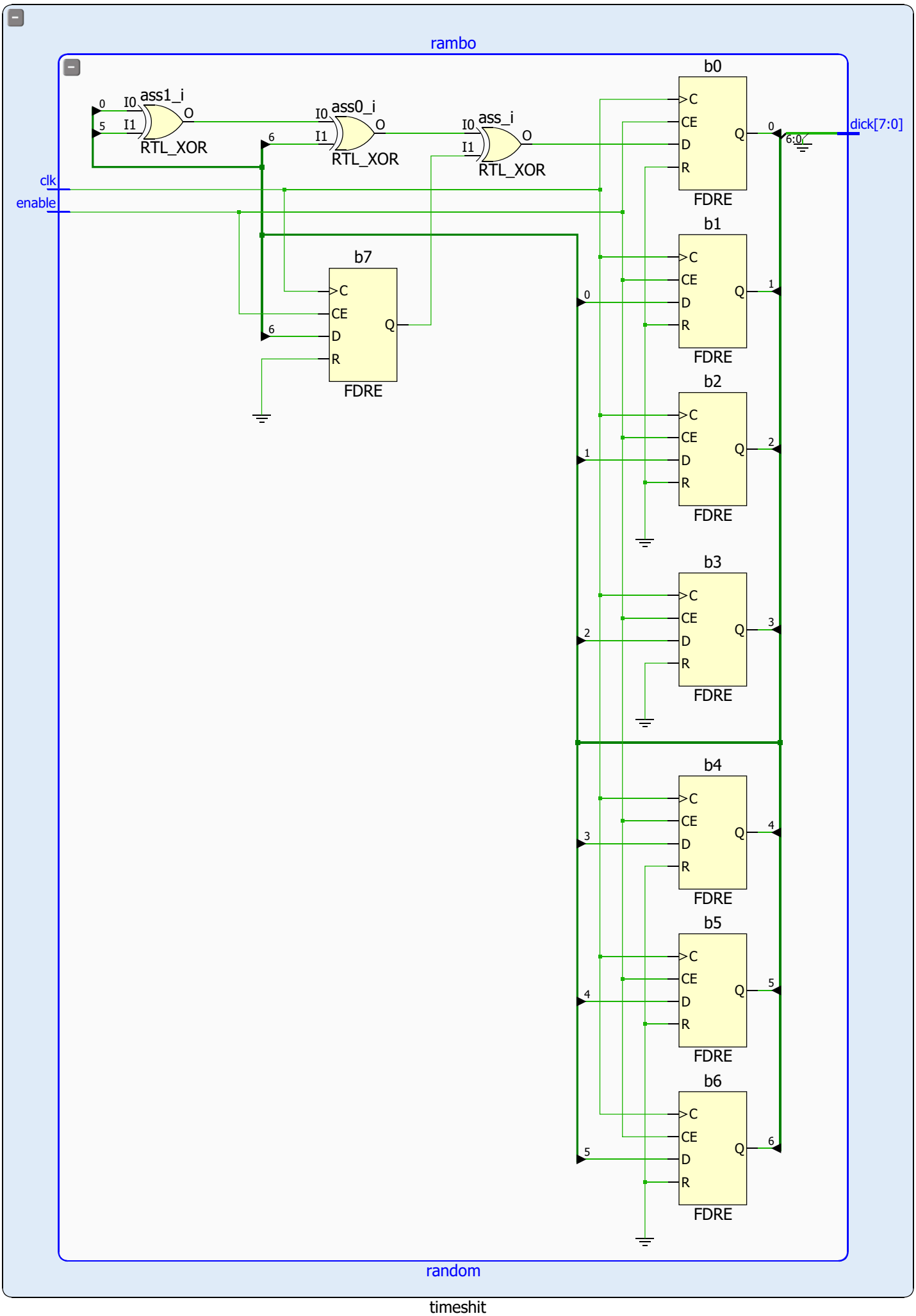
Shift Register image: https://en.wikipedia.org/wiki/Shift_register
Time counter image: UCSC CMPE100 Spring 2018 Lab 5 manual

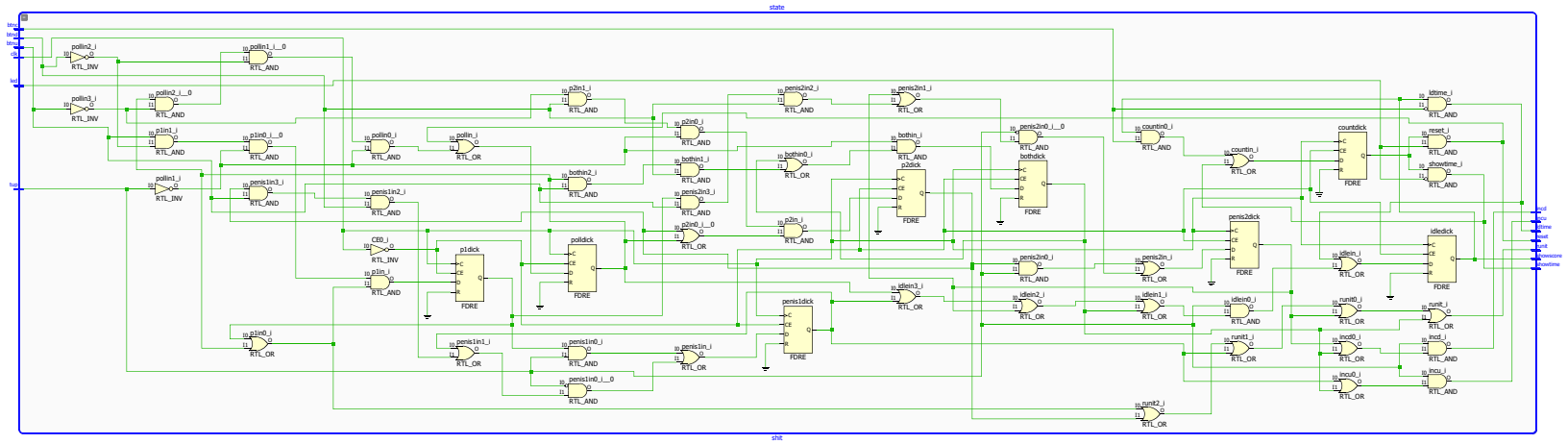
Appendix

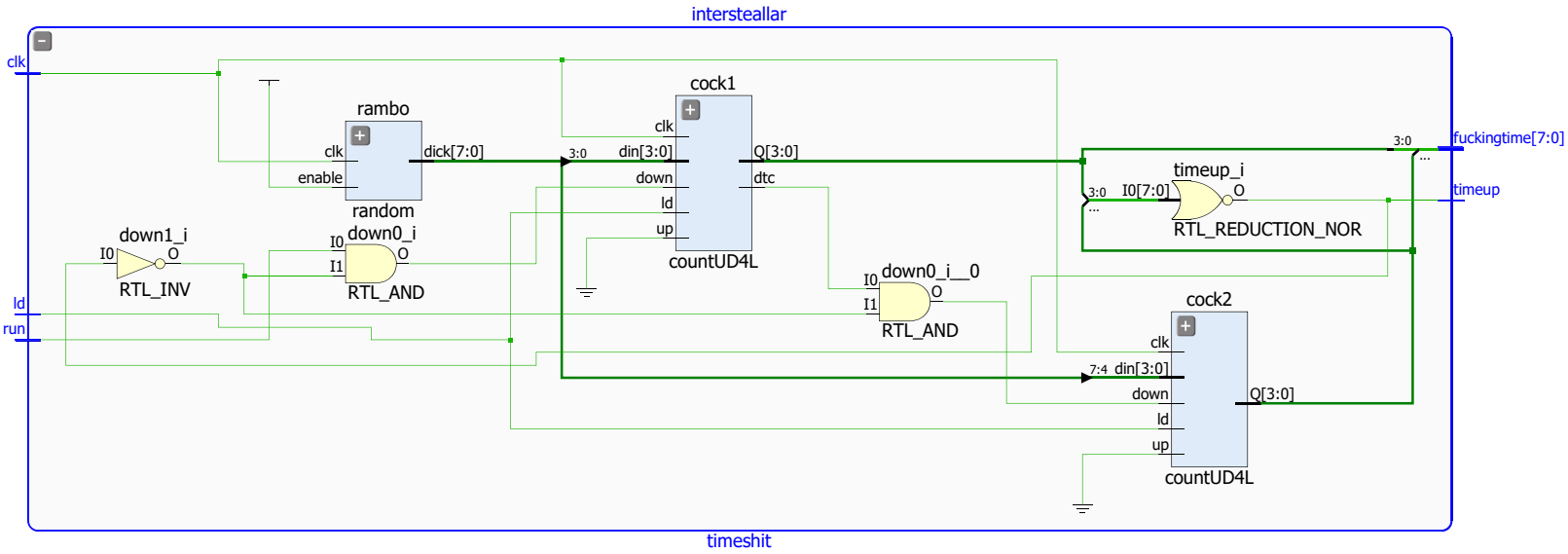

player1
clk
din[3:0]
down
up
Du1_i
Du1_i__0
Du1_i__1
Du1_i__2
Du1_i__3
Du1_i__4
Du0_i
Du0_i__0
Du0_i__1
Dd1_i
Dd1_i__0
Dd1_i__1
Dd1_i__2
Dd0_i
Dd0_i__0
Dd0_i__1
Du2_i
Du2_i__0
Dd2_i__1
Du3_i
Du3_i__0
Dd2_i
Dd2_i__0
RTL_AND
RTL_OR
RTL_INV
RTL_REDUCTION_AND
RTL_REDUCTION_OR
q0
q1
q2
q3
FDRE
D1_i
D1_i__0
D0_i
D0_i__0
D_i
dtc1_i
utc0_i
dtc_i
utc_i
enable0_i
enable_i
Q[3:0]
dtc n/c
utc n/c
countUD4L

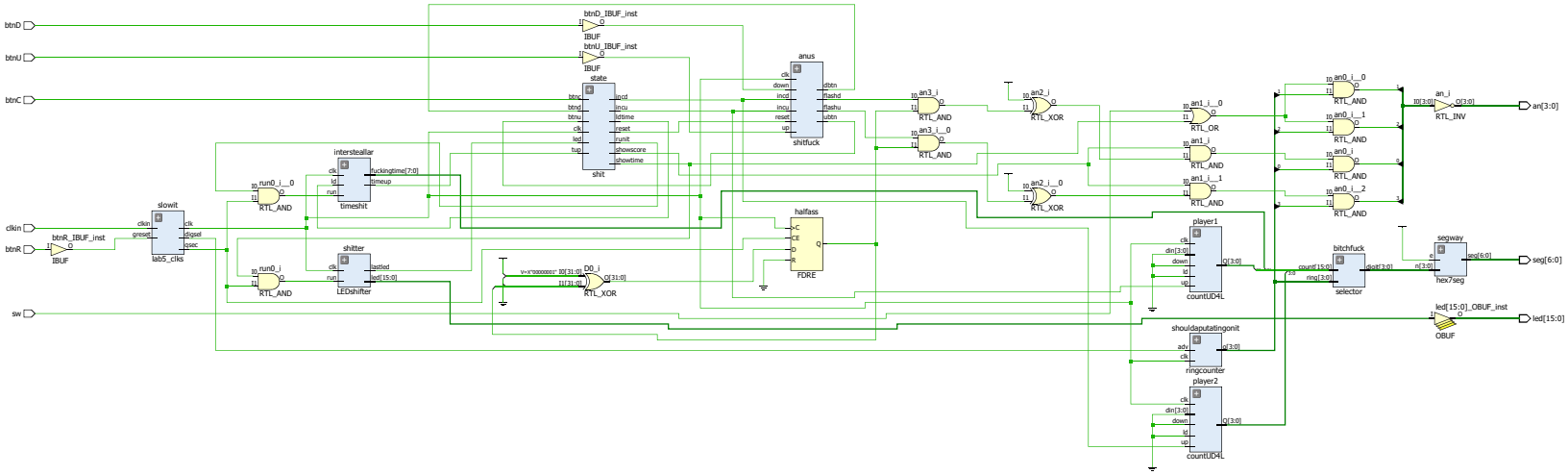

shitter
clk
run
q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q11
q12
q13
q14
q15
q16
FDRE
lastled
led[15:0]
LEDshifter

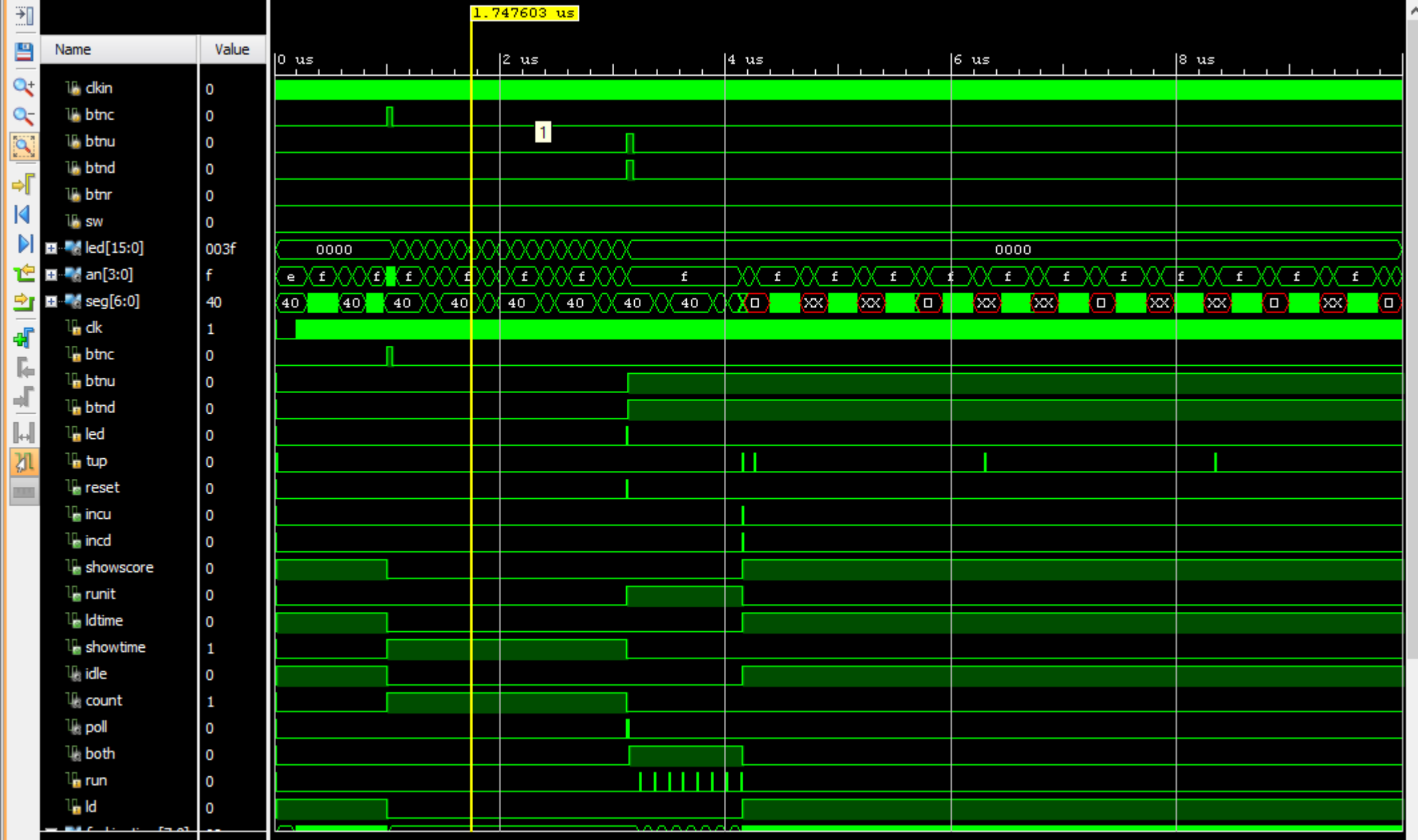

anus
flashitup
C
CE
D
R
Q
FDRE
downbutton
dbtn
D0_i__0
I0
I1
O
RTL_OR
D0_i__1
clk
CE0_i
RTL_INV
downsy
flashd
flashu
upbutton
down
D0_i
ubtn
incd
incu
reset
uppity
D0_i__2
flashitdown
up
shitfuck


intersteallar
rambo
ass1_i
RTL_XOR
ass0_i
RTL_XOR
ass_i
RTL_XOR
clk
enable
b7
FDRE
b0
b1
b2
b3
b4
b5
b6
FDRE
dick[7:0]
6:0
random
timeshit


state
shit
btnu
btnd
btnr
clk
led
tup
pollin2_i
RTL_INV
pollin1_i__0
RTL_AND
pollin3_i
RTL_INV
pollin2_i__0
RTL_AND
p1in1_i
RTL_AND
p1in0_i__0
RTL_AND
pollin1_i
RTL_INV
penis1in3_i
RTL_AND
pollin0_i
RTL_AND
pollin_i
RTL_OR
penis1in2_i
RTL_AND
CE0_i
RTL_INV
p1in_i
RTL_AND
p1dick
FDRE
p1in0_i
RTL_OR
p2in1_i
RTL_AND
bothin2_i
RTL_AND
polldick
FDRE
penis1in1_i
RTL_OR
penis1in0_i
RTL_AND
penis1in0_i__0
RTL_AND
penis1in_i
RTL_OR
p2in0_i
RTL_AND
penis1in3_i
RTL_AND
p2in0_i__0
RTL_OR
penis2in2_i
RTL_AND
bothin0_i
RTL_OR
p2in_i
RTL_AND
penis1dick
FDRE
penis2in1_i
RTL_OR
bothin_i
RTL_AND
p2dick
FDRE
idlein3_i
RTL_OR
penis2in0_i__0
RTL_AND
bothin1_i
RTL_AND
bothdick
FDRE
penis2in0_i
RTL_AND
idlein2_i
RTL_OR
countin0_i
RTL_AND
countin_i
RTL_OR
penis2dick
FDRE
penis2in_i
RTL_OR
idlein1_i
RTL_OR
runit2_i
RTL_OR
countdick
FDRE
idlein_i
RTL_OR
idlein0_i
RTL_AND
runit1_i
RTL_OR
incd0_i
RTL_OR
incu0_i
RTL_OR
ldtime_i
RTL_AND
reset_i
RTL_AND
showtime_i
RTL_AND
idledick
FDRE
runit0_i
RTL_OR
incd_i
RTL_AND
incu_i
RTL_AND
led
ldtime
reset
tup
showscore
showtime


intersteallar
clk
ld
run
rambo
clk
enable
dick[7:0]
random
down1_i
I0
O
RTL_INV
down0_i
I0
I1
O
RTL_AND
cock1
clk
din[3:0]
down
ld
up
Q[3:0]
dtc
countUD4L
3:0
down0_i__0
I0
I1
O
RTL_AND
timeup_i
I0[7:0]
O
RTL_REDUCTION_NOR
3:0
fuckingtime[7:0]
timeup
cock2
clk
7:4
din[3:0]
down
ld
up
Q[3:0]
countUD4L
timeshit


btnD
btnU
btnC
clkin
btnR
sw
btnD_IBUF_inst
IBUF
btnU_IBUF_inst
IBUF
state
btnc
btnd
btnu
clk
led
tup
idcd
incu
idtime
reset
trunit
showscore
showtime
shit
anus
clk
down
incd
incu
reset
up
d0btn
flashd
flashu
ubtn
shitfuck
an3_i
RTL_AND
an3_i_0
RTL_AND
an2_i
RTL_XOR
an2_i_0
RTL_XOR
an1_i_0
RTL_OR
an1_i
RTL_AND
an1_i_1
RTL_AND
an0_i_0
RTL_AND
an0_i_1
RTL_AND
an0_i
RTL_AND
an0_i_2
RTL_AND
an_i
RTL_INV
an[3:0]
interstellar
clk
ld
run
fuckingtime[7:0]
timeup
timeshit
run0_i_0
RTL_AND
slowit
clkin
greset
clk
clqout
clqsec
lab5_clks
shitter
clk
run
lastled
led[15:0]
LEDshifter
run0_i
RTL_AND
D0_i
RTL_XOR
halfass
C
CE
D
R
Q
FDRE
player1
clk
din[3:0]
down
ld
up
Q[3:0]
countUD4L
shouldaputatingonit
adv
clk
Q[3:0]
ringcounter
player2
clk
din[3:0]
down
ld
up
Q[3:0]
countUD4L
bitchfuck
count[15:0]
ring[3:0]
dout[3:0]
selector
segway
clk
an[3:0]
seg[6:0]
hex7seg
seg[6:0]
led[15:0]_OBUF_inst
OBUF
led[15:0]


1.747603 us
0 us
2 us
4 us
6 us
8 us
Name
Value
clkin 0
btnc 0
btnu 0
btnd 0
btnr 0
sw 0
led[15:0] 003f
an[3:0] f
seg[6:0] 40
clk 1
btnc 0
btnu 0
btnd 0
led 0
tup 0
reset 0
incu 0
incd 0
showscore 0
runit 0
ldtime 0
showtime 1
idle 0
count 1
poll 0
both 0
run 0
ld 0
0000
0000
1

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/17/2018 11:55:15 AM
// Design Name:
// Module Name: fux4to1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//4 to 1 mux
//input 1 bit enable, 2 bit selector and 4 bit input
//output 1 bit according to input and selector bit

//input - single selector bit, 2 bit selector bus, and 4 bit input bus
//output - single bit

module fux4to1(
    input [1:0] s,
    input [3:0] i,
    input e,
    output out
    );

    assign out = (~s[0]&~s[1]&i[0] | s[0]&~s[1]&i[1]
         | ~s[0]&s[1]&i[2] | s[0]&s[1]&i[3]) & e;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/17/2018 12:02:06 PM
// Design Name:
// Module Name: fux8to1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//8 bit mux
//input 1 bit enable,
//1 bus of 8 bit width input,
//and 1 bus of 3 bit width selector
//output is one bit

module fux8to1(
    input [7:0] in,
    input [2:0] s,
    input e,
    output out
    );

    wire m1, m2;

    fux4to1 mux1 (.s(s[1:0]), .i(in[3:0]), .e(1), .out(m1));
    fux4to1 mux2 (.s(s[1:0]), .i(in[7:4]), .e(1), .out(m2));
    assign out = (m1&~s[2] | m2&s[2])^e;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/19/2018 01:35:26 AM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//interface with 7 segment display

module hex7seg(
    input [3:0] n,//4 bit input number
    input e, //enable pin
    output [6:0] seg // segment output

    );

    wire y, ny, x, z, w;
    assign y = n[0];
    assign ny = ~n[0];
    assign x = n[1];
    assign w = n[2];
    assign z = n[3];

    //a led,
    //min terms: 1, 4, 11, 13
    fux8tol a (.in(~{1'b0,y,y,1'b0,1'b0,ny,1'b0,y}),.s({z,w,x}), .e(e), .out(seg[0]));

    //b led
    //min terms: 5, 6, 11, 12, 14, 15
    fux8tol b (.in(~{1'b1,ny,y,1'b0,ny,y,1'b0,1'b0}),.s({z,w,x}), .e(e), .out(seg[1]));

    //c led
    //min terms: 2, 12, 14, 15
    fux8tol c (.in(~{1'b1,ny,1'b0,1'b0,1'b0,1'b0,ny,1'b0}),.s({z,w,x}), .e(e), .out(seg[2]));

    //d led
    //min terms: 1, 4, 7, 10, 15
    fux8tol d (.in(~{y,1'b0,ny,1'b0,y,ny,1'b0,y}),.s({z,w,x}), .e(e), .out(seg[3]));

    //e led
    //min terms: 1, 3, 4, 5, 7, 9
    fux8tol ce (.in(~{1'b0,1'b0,1'b0,y,y,1'b1,y,y}),.s({z,w,x}), .e(e), .out(seg[4]));

    //f led
    //min terms: 1, 2, 3, 7, 13
    fux8tol f (.in(~{1'b0,y,1'b0,1'b0,y,1'b0,1'b1,y}),.s({z,w,x}), .e(e), .out(seg[5]));

    //g led
```

```
    //min terms: 0, 1, 7, 12
    fux8tol g (.in({1'b0,ny,1'b0,1'b0,y,1'b0,1'b0,1'b1}),.s({z,w,x}), .e(e), .out(seg[6]));


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/10/2017 11:19:41 AM
// Design Name:
// Module Name: lab5_clks
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module lab5_clks(
    input clkin,
    input greset,  //btnR
    output clk,
    output digsel,
    output qsec,
    output fastclk);

       wire clk_int;
       assign fastclk = clk_int;
       clk_wiz_0 my_clk_inst (.clk_out1(clk_int), .reset(greset), .locked(), .clk_in1(clkin));

       clkcntrl4 slowclk (.clk_int(clk_int), .seldig(digsel), .clk_out(clk), .qsec(qsec));

       STARTUPE2 #(.PROG_USR("FALSE"), // Activate program event security feature. Requires en
crypted bitstreams.
                   .SIM_CCLK_FREQ(0.0)    // Set the Configuration Clock Frequency(ns) for si
mulation.
                                          )
              STARTUPE2_inst (.CFGCLK(),  // 1-bit output: Configuration main clock output
                              .CFGMCLK(), // 1-bit output: Configuration internal oscillator c
lock output
                              .EOS(),     // 1-bit output: Active high output signal indicatin
g the End Of Startup.
                              .PREQ(),// 1-bit output: PROGRAM request to fabric output
                              .CLK(),  // 1-bit input: User start-up clock input
                              .GSR(greset),  // 1-bit input: Global Set/Reset input (GSR canno
t be used for the port name)
                              .GTS(),  // 1-bit input: Global 3-state input (GTS cannot be use
d for the port name)
                              .KEYCLEARB(), // 1-bit input: Clear AES Decrypter Key input from
 Battery-Backed RAM (BBRAM)
                              .PACK(), // 1-bit input: PROGRAM acknowledge input
                              .USRCCLKO(), // 1-bit input: User CCLK input
                              .USRCCLKTS(), // 1-bit input: User CCLK 3-state enable input
                               .USRDONEO(), // 1-bit input: User DONE pin output control
                               .USRDONETS() // 1-bit input: User DONE 3-state enable output

                             );  // End of STARTUPE2_inst instantiation
```

```
endmodule

module clk_wiz_0

 (// Clock in ports
  // Clock out ports
  output        clk_out1,
  // Status and control signals
  input         reset,
  output        locked,
  input         clk_in1
 );
  // Input buffering
  //------------------------------------
wire clk_in1_clk_wiz_0;
wire clk_in2_clk_wiz_0;
  IBUF clkin1_ibufg
   (.O (clk_in1_clk_wiz_0),
    .I (clk_in1));


  // Clocking PRIMITIVE
  //------------------------------------

  // Instantiation of the MMCM PRIMITIVE
  //    * Unused inputs are tied off
  //    * Unused outputs are labeled unused

  wire        clk_out1_clk_wiz_0;
  wire        clk_out2_clk_wiz_0;
  wire        clk_out3_clk_wiz_0;
  wire        clk_out4_clk_wiz_0;
  wire        clk_out5_clk_wiz_0;
  wire        clk_out6_clk_wiz_0;
  wire        clk_out7_clk_wiz_0;

  wire [15:0] do_unused;
  wire        drdy_unused;
  wire        psdone_unused;
  wire        locked_int;
  wire        clkfbout_clk_wiz_0;
  wire        clkfbout_buf_clk_wiz_0;
  wire        clkfboutb_unused;
    wire clkout0b_unused;
   wire clkout1_unused;
   wire clkout1b_unused;
   wire clkout2_unused;
   wire clkout2b_unused;
   wire clkout3_unused;
   wire clkout3b_unused;
   wire clkout4_unused;
  wire        clkout5_unused;
  wire        clkout6_unused;
  wire        clkfbstopped_unused;
  wire        clkinstopped_unused;
  wire        reset_high;

  MMCME2_ADV
  #(.BANDWIDTH            ("OPTIMIZED"),
    .CLKOUT4_CASCADE      ("FALSE"),
    .COMPENSATION         ("ZHOLD"),
    .STARTUP_WAIT         ("FALSE"),
    .DIVCLK_DIVIDE        (1),
```

```
    .CLKFBOUT_MULT_F      (9.125),
    .CLKFBOUT_PHASE       (0.000),
    .CLKFBOUT_USE_FINE_PS ("FALSE"),
    .CLKOUT0_DIVIDE_F     (36.500),
    .CLKOUT0_PHASE        (0.000),
    .CLKOUT0_DUTY_CYCLE   (0.500),
    .CLKOUT0_USE_FINE_PS  ("FALSE"),
    .CLKIN1_PERIOD        (10.0))
  mmcm_adv_inst
    // Output clocks
   (
    .CLKFBOUT             (clkfbout_clk_wiz_0),
    .CLKFBOUTB            (clkfboutb_unused),
    .CLKOUT0              (clk_out1_clk_wiz_0),
    .CLKOUT0B             (clkout0b_unused),
    .CLKOUT1              (clkout1_unused),
    .CLKOUT1B             (clkout1b_unused),
    .CLKOUT2              (clkout2_unused),
    .CLKOUT2B             (clkout2b_unused),
    .CLKOUT3              (clkout3_unused),
    .CLKOUT3B             (clkout3b_unused),
    .CLKOUT4              (clkout4_unused),
    .CLKOUT5              (clkout5_unused),
    .CLKOUT6              (clkout6_unused),
     // Input clock control
    .CLKFBIN              (clkfbout_buf_clk_wiz_0),
    .CLKIN1               (clk_in1_clk_wiz_0),
    .CLKIN2               (1'b0),
     // Tied to always select the primary input clock
    .CLKINSEL             (1'b1),
    // Ports for dynamic reconfiguration
    .DADDR                (7'h0),
    .DCLK                 (1'b0),
    .DEN                  (1'b0),
    .DI                   (16'h0),
    .DO                   (do_unused),
    .DRDY                 (drdy_unused),
    .DWE                  (1'b0),
    // Ports for dynamic phase shift
    .PSCLK                (1'b0),
    .PSEN                 (1'b0),
    .PSINCDEC             (1'b0),
    .PSDONE               (psdone_unused),
    // Other control and status signals
    .LOCKED               (locked_int),
    .CLKINSTOPPED         (clkinstopped_unused),
    .CLKFBSTOPPED         (clkfbstopped_unused),
    .PWRDWN               (1'b0),
    .RST                  (reset_high));
  assign reset_high = reset;

  assign locked = locked_int;
// Clock Monitor clock assigning
//--------------------------------------
 // Output buffering
  //-----------------------------------

  BUFG clkf_buf
   (.O (clkfbout_buf_clk_wiz_0),
    .I (clkfbout_clk_wiz_0));
```

```
  BUFG clkout1_buf
   (.O   (clk_out1),
    .I   (clk_out1_clk_wiz_0));




endmodule

module clkcntrl4(
     input clk_int,
     output seldig,
     output clk_out,
     output qsec);

   //wire XLXN_38;
   //wire XLXN_39;
   wire XLXN_44;
   wire XLXN_47;
   wire XLXN_70;
   wire XLXN_71;
   wire XLXN_72;
   wire XLXN_73;
   wire XLXN_74;
   wire XLXN_75;
   wire XLXN_76;
   wire XLXN_77;
   wire XLXN_79;
   wire clkb2_DUMMY;

   GND  XLXI_24 (.G(XLXN_44));

   (* HU_SET = "XLXI_37_73" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_37 (.C(clkb2_DUMMY),
                                     .CE(XLXN_73),
                                     .CLR(XLXN_76),
                                     .CEO(XLXN_72),
                                     .Q0(),
                                     .Q1(),
                                     .Q2(XLXN_74),
                                     .Q3(),
                                     .TC());
   (* HU_SET = "XLXI_38_74" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_38 (.C(clkb2_DUMMY),
                                     .CE(XLXN_72),
                                     .CLR(XLXN_76),
                                     .CEO(XLXN_70),
                                     .Q0(),
                                     .Q1(),
                                     .Q2(),
                                     .Q3(),
                                     .TC());
   (* HU_SET = "XLXI_39_75" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_39 (.C(clkb2_DUMMY),
                                     .CE(XLXN_70),
                                     .CLR(XLXN_76),
                                     .CEO(XLXN_71),
                                    .Q0(),
                                     .Q1(),
                                     .Q2(),
                                     .Q3(XLXN_77),
                                     .TC());
   //(* HU_SET = "XLXI_40_76" *)
```

```
   CB4CE_MXILINX_clkcntrl4  XLXI_40 (.C(clk_out),
                                     .CE(XLXN_73),
                                     .CLR(XLXN_76),
                                     .CEO(XLXN_78),
                                     .Q0(),
                                     .Q1(),
                                     .Q2(),
                                     .Q3(),
                                     .TC(XLXN_75));

   CB4CE_MXILINX_clkcntrl4  XLXI_45 (.C(clk_out),
                                     .CE(XLXN_78),
                                     .CLR(XLXN_76),
                                     .CEO(XLXN_79),
                                     .Q0(),
                                     .Q1(),
                                     .Q2(),
                                     .Q3(),
                                     .TC());

   CB4CE_MXILINX_clkcntrl4  XLXI_44 (.C(clk_out),
                                     .CE(XLXN_79),
                                     .CLR(XLXN_76),
                                     .CEO(),
                                     .Q0(qsec0),
                                     .Q1(qsec2),
                                     .Q2(),
                                     .Q3(),
                                     .TC());

   AND3  I_12222 (.I0(qsec0),
                  .I1(qsec2),
                  .I2(XLXN_79),
                  //.I3(),
                  .O(qsec3));


   VCC  XLXI_41 (.P(XLXN_73));
   GND  XLXI_43 (.G(XLXN_76));
   BUF  XLXI_328 (.I(clk_int),
                  .O(clkb2_DUMMY));
`ifdef XILINX_SIMULATOR
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   BUF  XLXI_398 (.I(XLXN_75),.O(qsec));
   BUFG  XLXI_399 (.I(clk_int),.O(clk_out));
   //BUFG  XLXI_401 (.I(XLXN_77),.O(clk_out));
`else
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   BUF  XLXI_398 (.I(qsec3),.O(qsec));
   BUFG  XLXI_401 (.I(XLXN_77),.O(clk_out));
`endif

endmodule


module FTCE_MXILINX_clkcntrl4(C,
                              CE,
                              CLR,
                              T,
                              Q);

   parameter INIT = 1'b0;
```

```
    input C;
    input CE;
    input CLR;
    input T;
   output Q;

   wire TQ;
   wire Q_DUMMY;

   assign Q = Q_DUMMY;
   XOR2  I_36_32 (.I0(T),
                  .I1(Q_DUMMY),
                  .O(TQ));
   ///(* RLOC = "X0Y0" *)
   FDCE  I_36_35 (.C(C),
                  .CE(CE),
                  .CLR(CLR),
                  .D(TQ),
                  .Q(Q_DUMMY));
endmodule
`timescale 1ns / 1ps

module CB4CE_MXILINX_clkcntrl4(C,
                               CE,
                               CLR,
                               CEO,
                               Q0,
                               Q1,
                               Q2,
                               Q3,
                               TC);

    input C;
    input CE;
    input CLR;
   output CEO;
   output Q0;
   output Q1;
   output Q2;
   output Q3;
   output TC;

   wire T2;
   wire T3;
   wire XLXN_1;
   wire Q0_DUMMY;
   wire Q1_DUMMY;
   wire Q2_DUMMY;
   wire Q3_DUMMY;
   wire TC_DUMMY;

   assign Q0 = Q0_DUMMY;
   assign Q1 = Q1_DUMMY;
   assign Q2 = Q2_DUMMY;
   assign Q3 = Q3_DUMMY;
   assign TC = TC_DUMMY;
   (* HU_SET = "I_Q0_69" *)
   FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q0 (.C(C),
                               .CE(CE),
                               .CLR(CLR),
                               .T(XLXN_1),
                               .Q(Q0_DUMMY));
   (* HU_SET = "I_Q1_70" *)
```

```
   FTCE_MXILINX_clkcntr14 #( .INIT(1'b0) ) I_Q1 (.C(C),
                                .CE(CE),
                                .CLR(CLR),
                                .T(Q0_DUMMY),
                                .Q(Q1_DUMMY));
   (* HU_SET = "I_Q2_71" *)
   FTCE_MXILINX_clkcntr14 #( .INIT(1'b0) ) I_Q2 (.C(C),
                                .CE(CE),
                                .CLR(CLR),
                                .T(T2),
                                .Q(Q2_DUMMY));
   (* HU_SET = "I_Q3_72" *)
   FTCE_MXILINX_clkcntr14 #( .INIT(1'b0) ) I_Q3 (.C(C),
                                .CE(CE),
                                .CLR(CLR),
                                .T(T3),
                                .Q(Q3_DUMMY));
   AND4  I_36_31 (.I0(Q3_DUMMY),
                  .I1(Q2_DUMMY),
                  .I2(Q1_DUMMY),
                  .I3(Q0_DUMMY),
                  .O(TC_DUMMY));
   AND3  I_36_32 (.I0(Q2_DUMMY),
                  .I1(Q1_DUMMY),
                  .I2(Q0_DUMMY),
                  .O(T3));
   AND2  I_36_33 (.I0(Q1_DUMMY),
                  .I1(Q0_DUMMY),
                  .O(T2));
   VCC  I_36_58 (.P(XLXN_1));
   AND2  I_36_67 (.I0(CE),
                  .I1(TC_DUMMY),
                  .O(CEO));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/10/2018 12:38:16 AM
// Design Name:
// Module Name: LEDshifter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//16 bit shifter
//grows 1's in array
//grows while run is high

module LEDshifter(
    input clk,
    input run,
    output lastled,
    output [15:0] led
    );

     wire dick;
     assign reset = dick;
     assign lastled = dick;
     FDRE #(.INIT(1'b0) ) q0 (.C(clk), .R(reset), .CE(run), .D(1'b1), .Q(led[0]));
     FDRE #(.INIT(1'b0) ) q1 (.C(clk), .R(reset), .CE(run), .D(led[0]), .Q(led[1]));
     FDRE #(.INIT(1'b0) ) q2 (.C(clk), .R(reset), .CE(run), .D(led[1]), .Q(led[2]));
     FDRE #(.INIT(1'b0) ) q3 (.C(clk), .R(reset), .CE(run), .D(led[2]), .Q(led[3]));
     FDRE #(.INIT(1'b0) ) q4 (.C(clk), .R(reset), .CE(run), .D(led[3]), .Q(led[4]));
     FDRE #(.INIT(1'b0) ) q5 (.C(clk), .R(reset), .CE(run), .D(led[4]), .Q(led[5]));
     FDRE #(.INIT(1'b0) ) q6 (.C(clk), .R(reset), .CE(run), .D(led[5]), .Q(led[6]));
     FDRE #(.INIT(1'b0) ) q7 (.C(clk), .R(reset), .CE(run), .D(led[6]), .Q(led[7]));
     FDRE #(.INIT(1'b0) ) q8 (.C(clk), .R(reset), .CE(run), .D(led[7]), .Q(led[8]));
     FDRE #(.INIT(1'b0) ) q9 (.C(clk), .R(reset), .CE(run), .D(led[8]), .Q(led[9]));
     FDRE #(.INIT(1'b0) ) q10 (.C(clk), .R(reset), .CE(run), .D(led[9]), .Q(led[10]));
     FDRE #(.INIT(1'b0) ) q11 (.C(clk), .R(reset), .CE(run), .D(led[10]), .Q(led[11]));
     FDRE #(.INIT(1'b0) ) q12 (.C(clk), .R(reset), .CE(run), .D(led[11]), .Q(led[12]));
     FDRE #(.INIT(1'b0) ) q13 (.C(clk), .R(reset), .CE(run), .D(led[12]), .Q(led[13]));
     FDRE #(.INIT(1'b0) ) q14 (.C(clk), .R(reset), .CE(run), .D(led[13]), .Q(led[14]));
     FDRE #(.INIT(1'b0) ) q15 (.C(clk), .R(reset), .CE(run), .D(led[14]), .Q(led[15]));
     FDRE #(.INIT(1'b0) ) q16 (.C(clk), .R(reset), .CE(run), .D(led[15]), .Q(dick));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/09/2018 11:59:30 PM
// Design Name:
// Module Name: random
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//random number generator
//outputs some random 8 bit number, highest bit is 0
module random(
    input clk,
        input enable,
    output [7:0] dick
    );

    wire [7:0] shit;
    wire ass;
    assign ass = shit[0]^shit[5]^shit[6]^shit[7];

    FDRE #(.INIT(1'b1) ) b0 (.C(clk), .R(reset), .CE(enable), .D(ass), .Q(shit[0]));
    FDRE #(.INIT(1'b0) ) b1 (.C(clk), .R(reset), .CE(enable), .D(shit[0]), .Q(shit[1]));
    FDRE #(.INIT(1'b0) ) b2 (.C(clk), .R(reset), .CE(enable), .D(shit[1]), .Q(shit[2]));
    FDRE #(.INIT(1'b0) ) b3 (.C(clk), .R(reset), .CE(enable), .D(shit[2]), .Q(shit[3]));
    FDRE #(.INIT(1'b0) ) b4 (.C(clk), .R(reset), .CE(enable), .D(shit[3]), .Q(shit[4]));
    FDRE #(.INIT(1'b0) ) b5 (.C(clk), .R(reset), .CE(enable), .D(shit[4]), .Q(shit[5]));
    FDRE #(.INIT(1'b0) ) b6 (.C(clk), .R(reset), .CE(enable), .D(shit[5]), .Q(shit[6]));
    FDRE #(.INIT(1'b0) ) b7 (.C(clk), .R(reset), .CE(enable), .D(shit[6]), .Q(shit[7]));

    assign dick = {1'b0,shit[6:0]};
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/03/2018 12:42:46 PM
// Design Name:
// Module Name: ringcounter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ringcounter(
    input adv,
    input clk,
    output [3:0] o
    );

    //first bit
    FDRE #(.INIT(1'b1) ) q0 (.C(clk), .R(1'b0), .CE(adv), .D(o[3]), .Q(o[0]));
    //second bit
    FDRE #(.INIT(1'b0) ) q1 (.C(clk), .R(1'b0), .CE(adv), .D(o[0]), .Q(o[1]));
    //third bit
    FDRE #(.INIT(1'b0) ) q2 (.C(clk), .R(1'b0), .CE(adv), .D(o[1]), .Q(o[2]));
    //third bit
    FDRE #(.INIT(1'b0) ) q3 (.C(clk), .R(1'b0), .CE(adv), .D(o[2]), .Q(o[3]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2018 02:33:57 PM
// Design Name:
// Module Name: counterUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//mux to work with bit shifter
//output 4 bit section of 16 bit  count input
//4 bit shifter input
//count - 16 bit digit
//ring - bit rotator/bit shifter
//digit - 4 bit digit output

module selector(
    input [15:0] count,
    input [3:0] ring,
    output [3:0] digit
    );

    assign digit = {4{ring[3]}} & count[15:12] |
                   {4{ring[2]}} & count[11:8]  |
                   {4{ring[1]}} & count[7:4]   |
                   {4{ring[0]}} & count[3:0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/14/2018 11:42:39 PM
// Design Name:
// Module Name: shitfuck
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//interface to make shit last
//flip flops that remain high on single burst input

module shitfuck(
    input reset,
    input up,
    input down,
    input incu,
    input incd,
    input clk,
    output ubtn,
    output dbtn,
    output flashu,
    output flashd
    );


    wire btnu, btnd;

    FDRE #(.INIT(1'b0) ) uppity (.C(clk), .R(reset), .CE(~clk), .D(up), .Q(btnu));

    FDRE #(.INIT(1'b0) ) downsy (.C(clk), .R(reset), .CE(~clk), .D(down), .Q(btnd));

    FDRE #(.INIT(1'b0) ) upbutton (.C(clk), .R(reset), .CE(~clk), .D(btnu|ubtn), .Q(ubtn));

    FDRE #(.INIT(1'b0) ) downbutton (.C(clk), .R(reset), .CE(~clk), .D(btnd|dbtn), .Q(dbtn));

    FDRE #(.INIT(1'b0) ) flashitup (.C(clk), .R(reset), .CE(~clk), .D(incu|flashu), .Q(flashu)
);

    FDRE #(.INIT(1'b0) ) flashitdown (.C(clk), .R(reset), .CE(~clk), .D(incd|flashd), .Q(flash
d));



endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/08/2018 01:10:56 PM
// Design Name:
// Module Name: StateoftheUnion
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shit(
        input clk,
        input btnc,
        input btnu,
        input btnd,
        input led,
        input tup,
        output reset,
        output incu,
        output incd,
        output showscore,
        output runit,
        output ldtime,
        output showtime
        );


    //inputs
    wire idlein, countin, pollin, plin, p2in, penislin, penis2in, bothin;

    //outputs
    wire idle, count, poll, p1, p2, penis1, penis2, both;

    assign idlein  = idle&~btnc | tup & (poll | penis1 | penis2 | both);
    assign countin = idle&btnc | count&~led;
    assign pollin = count&led | poll&~btnu&~btnd&~tup;
    assign plin = (btnu&~btnd&~tup)&(p1 | poll);
    assign p2in = (~btnu&btnd&~tup)&(p2 | poll);
    assign penislin = p1&tup | ~tup&(penis1 | p2&btnu&btnd);
    assign penis2in = p2&tup | ~tup&(penis2 | p1&btnu&btnd);
    assign bothin = ~tup&(both | poll&btnu&btnd);


    //actual fucking outputs fucking shit fuck guck
    assign reset = count&led;
    assign showscore = idle;
    assign incu = tup&(penis1 | both);
    assign incd = tup&(penis2 | both);
    assign ldtime = idle&~btnc;
    assign showtime = count&~led;
    assign runit = poll | p1 | p2 | penis1 | penis2 | both;
```

```
        FDRE #(.INIT(1'b1) ) idledick (.C(clk), .R(0), .CE(~clk), .D(idlein), .Q(idle));

        FDRE #(.INIT(1'b0) ) countdick (.C(clk), .R(0), .CE(~clk), .D(countin), .Q(count));

        FDRE #(.INIT(1'b0) ) polldick (.C(clk), .R(0), .CE(~clk), .D(pollin), .Q(poll));

    FDRE #(.INIT(1'b0) ) pldick (.C(clk), .R(0), .CE(~clk), .D(plin), .Q(pl));

    FDRE #(.INIT(1'b0) ) p2dick (.C(clk), .R(0), .CE(~clk), .D(p2in), .Q(p2));

    FDRE #(.INIT(1'b0) ) penisldick (.C(clk), .R(0), .CE(~clk), .D(penislin), .Q(penisl));

    FDRE #(.INIT(1'b0) ) penis2dick (.C(clk), .R(0), .CE(~clk), .D(penis2in), .Q(penis2));

    FDRE #(.INIT(1'b0) ) bothdick (.C(clk), .R(0), .CE(~clk), .D(bothin), .Q(both));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/14/2018 11:23:44 PM
// Design Name:
// Module Name: timeshit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//time machine
//takes in as input run and load signals
//load while ld is high
//output on time, no roll down effect

module timeshit(
    input run,
    input ld,
    input clk,
    output [7:0] fuckingtime,
    output timeup
    );

    wire [7:0] fuck;
    wire fucking, shit;

    //random generator
    random rambo (.clk(clk), .enable(1'b1), .dick(fuck));

    //lower bits
    countUD4L cock1 (.clk(clk), .down(run&~timeup), .ld(ld), .up(1'b0),
        .din(fuck[3:0]), .Q(fuckingtime[3:0]), .dtc(fucking));


    //upper bits
    countUD4L cock2 (.clk(clk), .down(fucking&~timeup), .ld(ld), .up(1'b0),
        .din(fuck[7:4]), .Q(fuckingtime[7:4]), .dtc(shit));
    assign timeup = ~|fuckingtime;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/15/2018 12:08:41 AM
// Design Name:
// Module Name: topgun
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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

    wire clk, digsel, qsec, p1, p2, lastled, tup, run, ldtime, showtime, reset,
        showscore, flashp1, flashp2, incp1, incp2, halfsec, fuckingtime;

    wire [3:0] ring, n, p1score, p2score;
    wire [7:0] timebro;
    wire [15:0] display;

    //the fucking clock u fucking bitch
    lab5_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel), .qsec(qsec));

    FDRE #(.INIT(1'b1) ) halfass (.C(clk), .R(0), .CE(qsec), .D(1^halfsec), .Q(halfsec));

    //the fucking statemachine u whore
    shit state (.clk(clk), .btnc(btnC), .btnu(p1), .btnd(p2), .led(lastled), .tup(tup),
        .reset(reset), .incu(incp1), .incd(incp2), .showscore(showscore), .runit(run), .ldtime
(ldtime), .showtime(showtime));

    //fucking module shit
    shitfuck anus (.reset(reset), .up(btnU), .down(btnD), .incu(incp1), .incd(incp2), .clk(clk
), .ubtn(p1),
        .dbtn(p2), .flashu(flashp1), .flashd(flashp2));

    //led shifter shit
    //incriment on quarter second
    LEDshifter shitter (.clk(clk), .run(showtime&qsec), .lastled(lastled), .led(led));

    //FUCKING TIME BRO
    timeshit intersteallar (.clk(clk), .run(run&qsec), .ld(ldtime), .fuckingtime(timebro), .ti
meup(tup));
```

```
    //bitchass ring counter
    ringcounter shouldaputatingonit (.adv(digsel), .clk(clk), .o(ring));

    //bitch fuck selector
    selector bitchfuck (.count(display), .ring(ring), .digit(n));

    //fucking segway display bitch
    hex7seg segway (.n(n), .e(1'b1), .seg(seg));

    //p1 score shit
    countUD4L player1 (.clk(clk), .up(incp1), .Q(p1score));

    //p2 score shit
    countUD4L player2 (.clk(clk), .up(incp2), .Q(p2score));

    //set the fucking display data
    assign display = {p1score, timebro, p2score};

    //set the fucking display on shit
    assign an = ~{showscore&(1'b1^(flashp1&halfsec))&ring[3], (sw|showtime)&ring[2], (sw|showt
ime)&ring[1], showscore&(1'b1^(flashp2&halfsec))&ring[0]};


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/10/2018 12:38:16 AM
// Design Name:
// Module Name: LEDshifter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//16 bit shifter
//grows 1's in array
//grows while run is high

module LEDshifter(
   input clk,
   input run,
   output lastled,
   output [15:0] led
   );

    wire dick;
    assign reset = dick;
    assign lastled = dick;
    FDRE #(.INIT(1'b0) ) q0 (.C(clk), .R(reset), .CE(run), .D(1'b1), .Q(led[0]));
    FDRE #(.INIT(1'b0) ) q1 (.C(clk), .R(reset), .CE(run), .D(led[0]), .Q(led[1]));
    FDRE #(.INIT(1'b0) ) q2 (.C(clk), .R(reset), .CE(run), .D(led[1]), .Q(led[2]));
    FDRE #(.INIT(1'b0) ) q3 (.C(clk), .R(reset), .CE(run), .D(led[2]), .Q(led[3]));
    FDRE #(.INIT(1'b0) ) q4 (.C(clk), .R(reset), .CE(run), .D(led[3]), .Q(led[4]));
    FDRE #(.INIT(1'b0) ) q5 (.C(clk), .R(reset), .CE(run), .D(led[4]), .Q(led[5]));
    FDRE #(.INIT(1'b0) ) q6 (.C(clk), .R(reset), .CE(run), .D(led[5]), .Q(led[6]));
    FDRE #(.INIT(1'b0) ) q7 (.C(clk), .R(reset), .CE(run), .D(led[6]), .Q(led[7]));
    FDRE #(.INIT(1'b0) ) q8 (.C(clk), .R(reset), .CE(run), .D(led[7]), .Q(led[8]));
    FDRE #(.INIT(1'b0) ) q9 (.C(clk), .R(reset), .CE(run), .D(led[8]), .Q(led[9]));
    FDRE #(.INIT(1'b0) ) q10 (.C(clk), .R(reset), .CE(run), .D(led[9]), .Q(led[10]));
    FDRE #(.INIT(1'b0) ) q11 (.C(clk), .R(reset), .CE(run), .D(led[10]), .Q(led[11]));
    FDRE #(.INIT(1'b0) ) q12 (.C(clk), .R(reset), .CE(run), .D(led[11]), .Q(led[12]));
    FDRE #(.INIT(1'b0) ) q13 (.C(clk), .R(reset), .CE(run), .D(led[12]), .Q(led[13]));
```

```
    FDRE #(.INIT(1'b0) ) q14 (.C(clk), .R(reset), .CE(run), .D(led[13]), .Q(led[14]));
    FDRE #(.INIT(1'b0) ) q15 (.C(clk), .R(reset), .CE(run), .D(led[14]), .Q(led[15]));
    FDRE #(.INIT(1'b0) ) q16 (.C(clk), .R(reset), .CE(run), .D(led[15]), .Q(dick));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/09/2018 11:59:30 PM
// Design Name:
// Module Name: random
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//random number generator
//outputs some random 8 bit number, highest bit is 0
module random(
   input clk,
?input enable,
   output [7:0] dick
   );

   wire [7:0] shit;
   wire ass;
   assign ass = shit[0]^shit[5]^shit[6]^shit[7];

   FDRE #(.INIT(1'b1) ) b0 (.C(clk), .R(reset), .CE(enable), .D(ass), .Q(shit[0]));
   FDRE #(.INIT(1'b0) ) b1 (.C(clk), .R(reset), .CE(enable), .D(shit[0]), .Q(shit[1]));
   FDRE #(.INIT(1'b0) ) b2 (.C(clk), .R(reset), .CE(enable), .D(shit[1]), .Q(shit[2]));
   FDRE #(.INIT(1'b0) ) b3 (.C(clk), .R(reset), .CE(enable), .D(shit[2]), .Q(shit[3]));
   FDRE #(.INIT(1'b0) ) b4 (.C(clk), .R(reset), .CE(enable), .D(shit[3]), .Q(shit[4]));
   FDRE #(.INIT(1'b0) ) b5 (.C(clk), .R(reset), .CE(enable), .D(shit[4]), .Q(shit[5]));
   FDRE #(.INIT(1'b0) ) b6 (.C(clk), .R(reset), .CE(enable), .D(shit[5]), .Q(shit[6]));
   FDRE #(.INIT(1'b0) ) b7 (.C(clk), .R(reset), .CE(enable), .D(shit[6]), .Q(shit[7]));

   assign dick = {1'b0,shit[6:0]};
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/14/2018 11:42:39 PM
// Design Name:
// Module Name: shitfuck
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//interface to make shit last
//flip flops that remain high on single burst input

module shitfuck(
    input reset,
    input up,
    input down,
    input incu,
    input incd,
    input clk,
    output ubtn,
    output dbtn,
    output flashu,
    output flashd
    );


    wire btnu, btnd;

    FDRE #(.INIT(1'b0) ) uppity (.C(clk), .R(reset), .CE(~clk), .D(up), .Q(btnu));

    FDRE #(.INIT(1'b0) ) downsy (.C(clk), .R(reset), .CE(~clk), .D(down), .Q(btnd));

    FDRE #(.INIT(1'b0) ) upbutton (.C(clk), .R(reset), .CE(~clk), .D(btnu|ubtn), .Q(ubtn));

    FDRE #(.INIT(1'b0) ) downbutton (.C(clk), .R(reset), .CE(~clk), .D(btnd|dbtn), .Q(dbtn));

    FDRE #(.INIT(1'b0) ) flashitup (.C(clk), .R(reset), .CE(~clk), .D(incu|flashu), .Q(flashu));
```

```
    FDRE #(.INIT(1'b0) ) flashitdown (.C(clk), .R(reset), .CE(~clk), .D(incd|flashd), .Q(flashd));


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/08/2018 01:10:56 PM
// Design Name:
// Module Name: StateoftheUnion
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shit(
?input clk,
?input btnc,
?input btnu,
?input btnd,
?input led,
?input tup,
?output reset,
?output incu,
?output incd,
?output showscore,
?output runit,
?output ldtime,
?output showtime
?);


   //inputs
   wire idlein, countin, pollin, p1in, p2in, penis1in, penis2in, bothin;

   //outputs
   wire idle, count, poll, p1, p2, penis1, penis2, both;

   assign idlein  = idle&~btnc | tup & (poll | penis1 | penis2 | both);
   assign countin = idle&btnc | count&~led;
   assign pollin = count&led | poll&~btnu&~btnd&~tup;
   assign p1in = (btnu&~btnd&~tup)&(p1 | poll);
```

```
   assign p2in = (~btnu&btnd&~tup)&(p2 | poll);
   assign penis1in = p1&tup | ~tup&(penis1 | p2&btnu&btnd);
   assign penis2in = p2&tup | ~tup&(penis2 | p1&btnu&btnd);
   assign bothin = ~tup&(both | poll&btnu&btnd);


   //actual fucking outputs fucking shit fuck guck
   assign reset = count&led;
   assign showscore = idle;
   assign incu = tup&(penis1 | both);
   assign incd = tup&(penis2 | both);
   assign ldtime = idle&~btnc;
   assign showtime = count&~led;
   assign runit = poll | p1 | p2 | penis1 | penis2 | both;


?FDRE #(.INIT(1'b1) ) idledick (.C(clk), .R(0), .CE(~clk), .D(idlein), .Q(idle));


?FDRE #(.INIT(1'b0) ) countdick (.C(clk), .R(0), .CE(~clk), .D(countin), .Q(count));


?FDRE #(.INIT(1'b0) ) polldick (.C(clk), .R(0), .CE(~clk), .D(pollin), .Q(poll));
?

   FDRE #(.INIT(1'b0) ) p1dick (.C(clk), .R(0), .CE(~clk), .D(p1in), .Q(p1));


   FDRE #(.INIT(1'b0) ) p2dick (.C(clk), .R(0), .CE(~clk), .D(p2in), .Q(p2));


   FDRE #(.INIT(1'b0) ) penis1dick (.C(clk), .R(0), .CE(~clk), .D(penis1in), .Q(penis1));


   FDRE #(.INIT(1'b0) ) penis2dick (.C(clk), .R(0), .CE(~clk), .D(penis2in), .Q(penis2));


   FDRE #(.INIT(1'b0) ) bothdick (.C(clk), .R(0), .CE(~clk), .D(bothin), .Q(both));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/14/2018 11:23:44 PM
// Design Name:
// Module Name: timeshit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//time machine
//takes in as input run and load signals
//load while ld is high
//output on time, no roll down effect

module timeshit(
   input run,
   input ld,
   input clk,
   output [7:0] fuckingtime,
   output timeup
   );

   wire [7:0] fuck;
   wire fucking, shit;

   //random generator
   random rambo (.clk(clk), .enable(1'b1), .dick(fuck));

   //lower bits
   countUD4L cock1 (.clk(clk), .down(run&~timeup), .ld(ld), .up(1'b0),
      .din(fuck[3:0]), .Q(fuckingtime[3:0]), .dtc(fucking));


   //upper bits
   countUD4L cock2 (.clk(clk), .down(fucking&~timeup), .ld(ld), .up(1'b0),
      .din(fuck[7:4]), .Q(fuckingtime[7:4]), .dtc(shit));
```

```
    assign timeup = ~|fuckingtime;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/15/2018 12:08:41 AM
// Design Name:
// Module Name: topgun
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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

    wire clk, digsel, qsec, p1, p2, lastled, tup, run, ldtime, showtime, reset,
        showscore, flashp1, flashp2, incp1, incp2, halfsec, fuckingtime;

    wire [3:0] ring, n, p1score, p2score;
    wire [7:0] timebro;
    wire [15:0] display;

    //the fucking clock u fucking bitch
    lab5_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel), .qsec(qsec));

    FDRE #(.INIT(1'b1) ) halfass (.C(clk), .R(0), .CE(qsec), .D(1^halfsec), .Q(halfsec));

    //the fucking statemachine u whore
    shit state (.clk(clk), .btnc(btnC), .btnu(p1), .btnd(p2), .led(lastled), .tup(tup),
        .reset(reset), .incu(incp1), .incd(incp2), .showscore(showscore), .runit(run), .ldtime(ldtime), .show
```

```
    //fucking module shit
    shitfuck anus (.reset(reset), .up(btnU), .down(btnD), .incu(incp1), .incd(incp2), .clk(clk), .ubtn(p1),
       .dbtn(p2), .flashu(flashp1), .flashd(flashp2));

    //led shifter shit
    //incriment on quarter second
    LEDshifter shitter (.clk(clk), .run(showtime&qsec), .lastled(lastled), .led(led));

    //FUCKING TIME BRO
    timeshit interstellar (.clk(clk), .run(run&qsec), .ld(ldtime), .fuckingtime(timebro), .timeup(tup));

    //bitchass ring counter
    ringcounter shouldaputatingonit (.adv(digsel), .clk(clk), .o(ring));

    //bitch fuck selector
    selector bitchfuck (.count(display), .ring(ring), .digit(n));

    //fucking segway display bitch
    hex7seg segway (.n(n), .e(1'b1), .seg(seg));

    //p1 score shit
    countUD4L player1 (.clk(clk), .up(incp1), .Q(p1score));

    //p2 score shit
    countUD4L player2 (.clk(clk), .up(incp2), .Q(p2score));

    //set the fucking display data
    assign display = {p1score, timebro, p2score};

    //set the fucking display on shit
    assign an = ~{showscore&(1'b1^(flashp1&halfsec))&ring[3], (sw|showtime)&ring[2], (sw|showtime


endmodule
```