Write up- Lab7
Simranpreet Kaur
June 3, 2022
Lab Section C

Objective:
The objective of this lab was to design and implement a game called Frog Frenzy where the frog can jump over or dive under the randomly generated plants to get points on the Basys3 Board.

Methods and Design:

In order to complete this lab, I imported many old modules from previous labs and also built new modules which include:

- Frog machine: This is my overall state machine that controls the whole game to indicate what needs to be done at which state throughout the game
- CountUD10L: 10-bit counter that counts up and down used multiple times in the game
- VGA: this module controls all my display in the active region and logic for water gradient, and plant randomization
- Ring counter, LFSR, hex 7 seg, selector are used from the previous labs

<u>Top Level:</u>

In my top level, I called all the modules that needed to be connected and made wires for all their outputs. In my top level, I also included the anode logic where I had to make sure that the board only displays the score counting up every time my frog dodges a plant and the two seconds logic. The two seconds logic is used for the frog to wait and blink for two seconds before starting the game.

<u>VGA:</u>

The VGA module is very important to my lab since it has the most important logic that creates active regions for my game and displays each part on the monitor. Each item created in the game is defined with boundaries and then the colors are put within those boundaries using the RGB hex values. The water needed to be a gradient lake where the water starts off as a lighter color but becomes a darker blue as it goes down the screen. In this module I also handled the plant randomization where each plant had to reset back on the screen at a new position to make the game harder to win. In order to do this, we used the previous LFSR module from lab6.

<u>Frog Machine:</u>

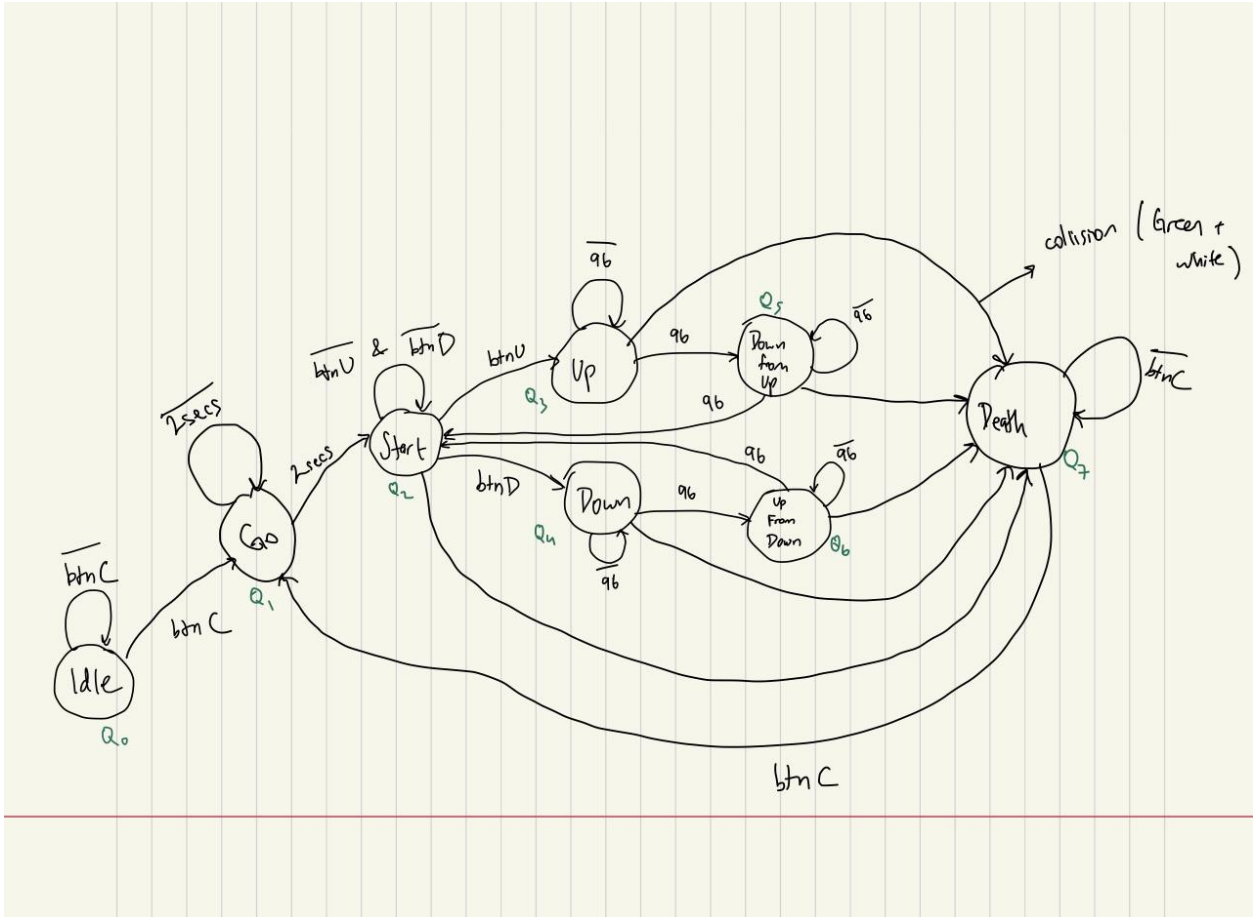
This module includes my state machine which has 8 states: Idle, Go, Start, Up, Down, Down from Up, Up from Down, Death. The game remains in the idle state until button C is pressed on

the board which causes the game to go to the Go state. The game remains in that state until 2 seconds have passed and then the game Starts from where you can press button U or button D to make the frog move up and down. When you go to either Up or Down state, the frog stays in that state until you pass 96 frames. Depending on whether you were in the Up or Down state, the frog has to come back to the center from both ways. All states except Go and Idle can lead to the death state since the frog can collide with the plant at any time during those states. In order to go back to the Go state, you have to press btnC to start the game again and keep track of a new score.

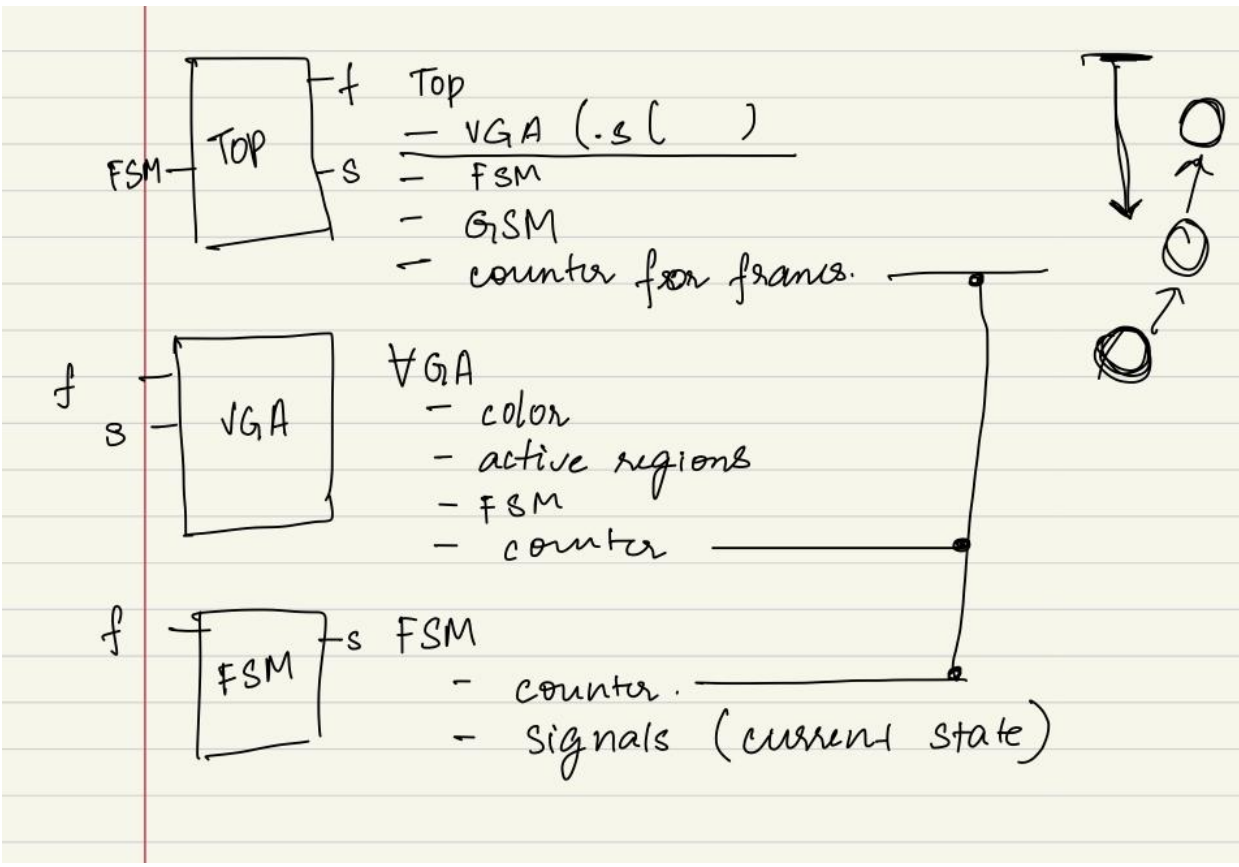
State Machine Equations:

```
D[0] = (Q[0] & ~btnC);

D[1] = (Q[0] & btnC) | (Q[7] & btnC) | (Q[1] & ~two_secs);

D[2] = (Q[1] & two_secs) | (Q[2] & (~btnD & ~btnU & ~collide)) | (Q[5] & count_output == 10'd32)
| (Q[6] & count_output == 10'd32);

D[3] = (Q[2] & btnU) | (Q[3] & count_output != 10'd32 & ~collide);

D[4] = (Q[2] & btnD) | (Q[4] & count_output != 10'd32 & ~collide);

D[5] = (Q[3] & count_output == 10'd32) | (Q[5] & count_output != 10'd32 & ~collide);

D[6] = (Q[4] & count_output == 10'd32) | (Q[6] & count_output != 10'd32 & ~collide);

D[7] = ((Q[2] & collide) | (Q[3] & collide) | (Q[4] & collide) | (Q[5] & collide) | (Q[6] & collide) |
(Q[7] & ~btnC));
```

State Machine Diagram:



The diagram below shows how all my main modules (VGA and Frog State Machine) are connected in the top module.

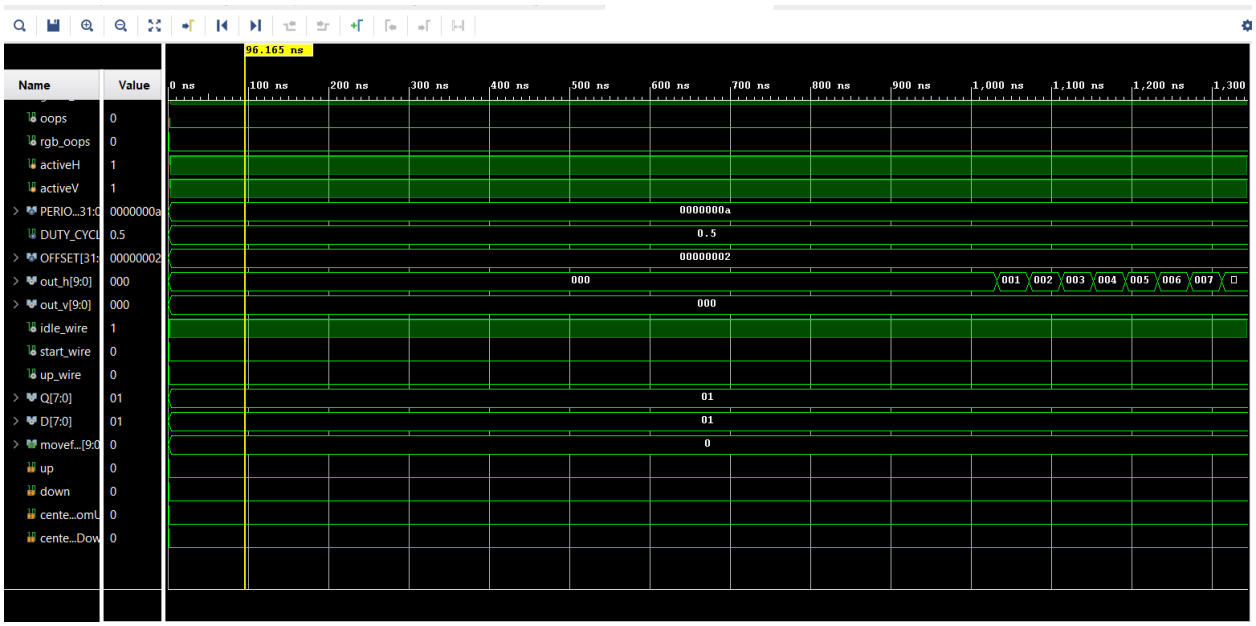


Results:

<u>Testing and Simulation:</u>

To debug and test my code, I built a simulation for my top module and for my state machine to avoid small errors that may take a while to debug. While creating this game, I kept having trouble with resetting my plants in a new frame for the frog to keep playing. After debugging my code I realized I was resetting each plant at its initial position every frame instead of randomizing after every frame.

Simulation for Top Module:



Design Timing and Clock Summaries

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	26.572 ns	Worst Hold Slack (WHS):	0.158 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	292	Total Number of Endpoints:	292	Total Number of Endpoints:	133

All user specified timing constraints are met.

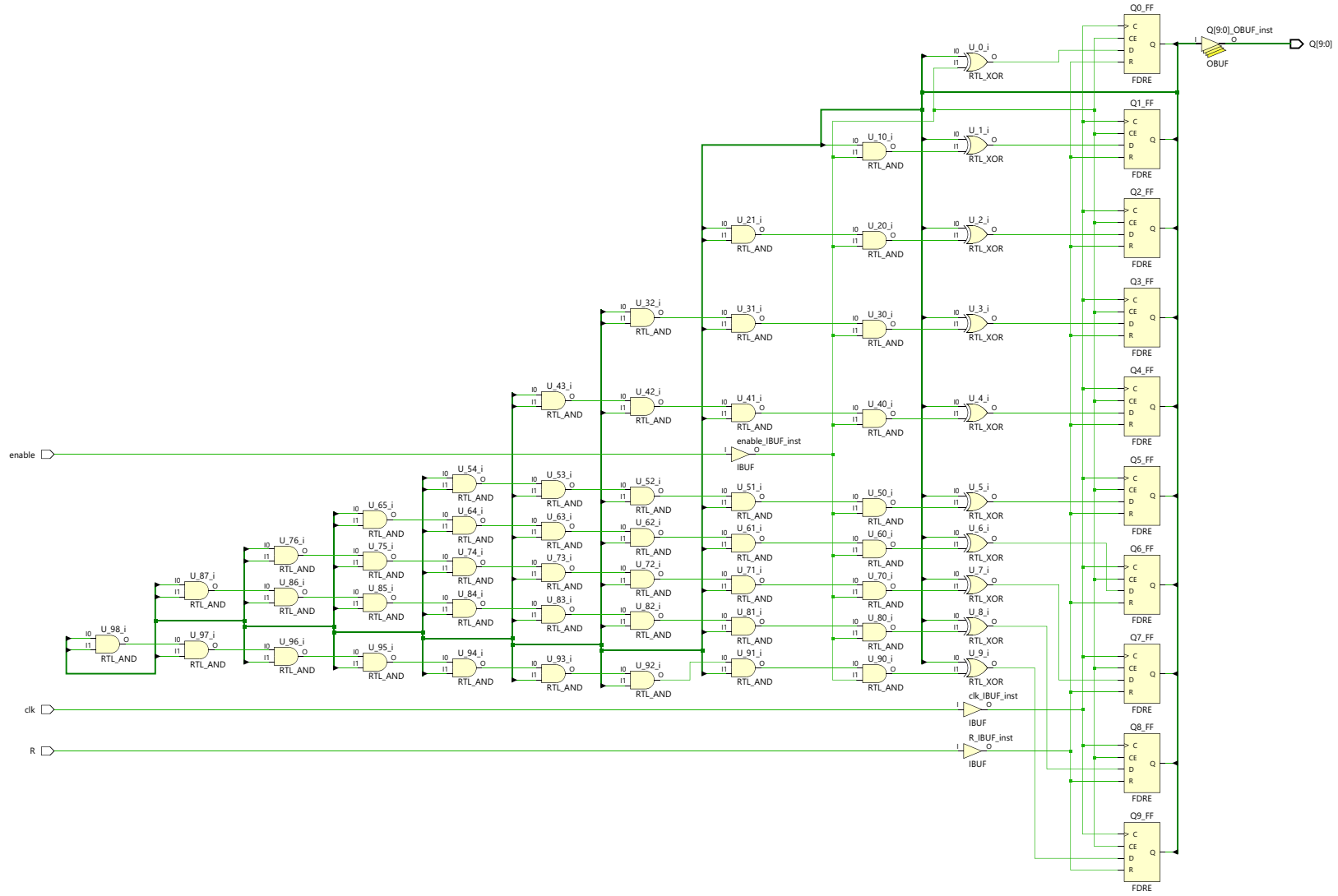
Clock Summary

Name	Waveform	Period (ns)	Frequency (MHz)
sys_clk_pin	{0.000 5.000}	10.000	100.000
clk_out1_clk_wiz_0	{0.000 20.000}	40.000	25.000
clkfbout_clk_wiz_0	{0.000 5.000}	10.000	100.000

Conclusion:

In conclusion, this game in my opinion was the hardest lab of this class since we had to create our own design without any top module diagram given and we had to learn how to work with the VGA monitors. I learned how game visuals are displayed on monitors since I didn't know that each item needs its own active region that has to individually move. After creating this simple game, I realized how difficult it is to give characters and backgrounds so much detail in the games most people play nowadays. It is really hard to make an item move smoothly without creating a static visual since you have to be consistent with how many pixels you can move in one frame.

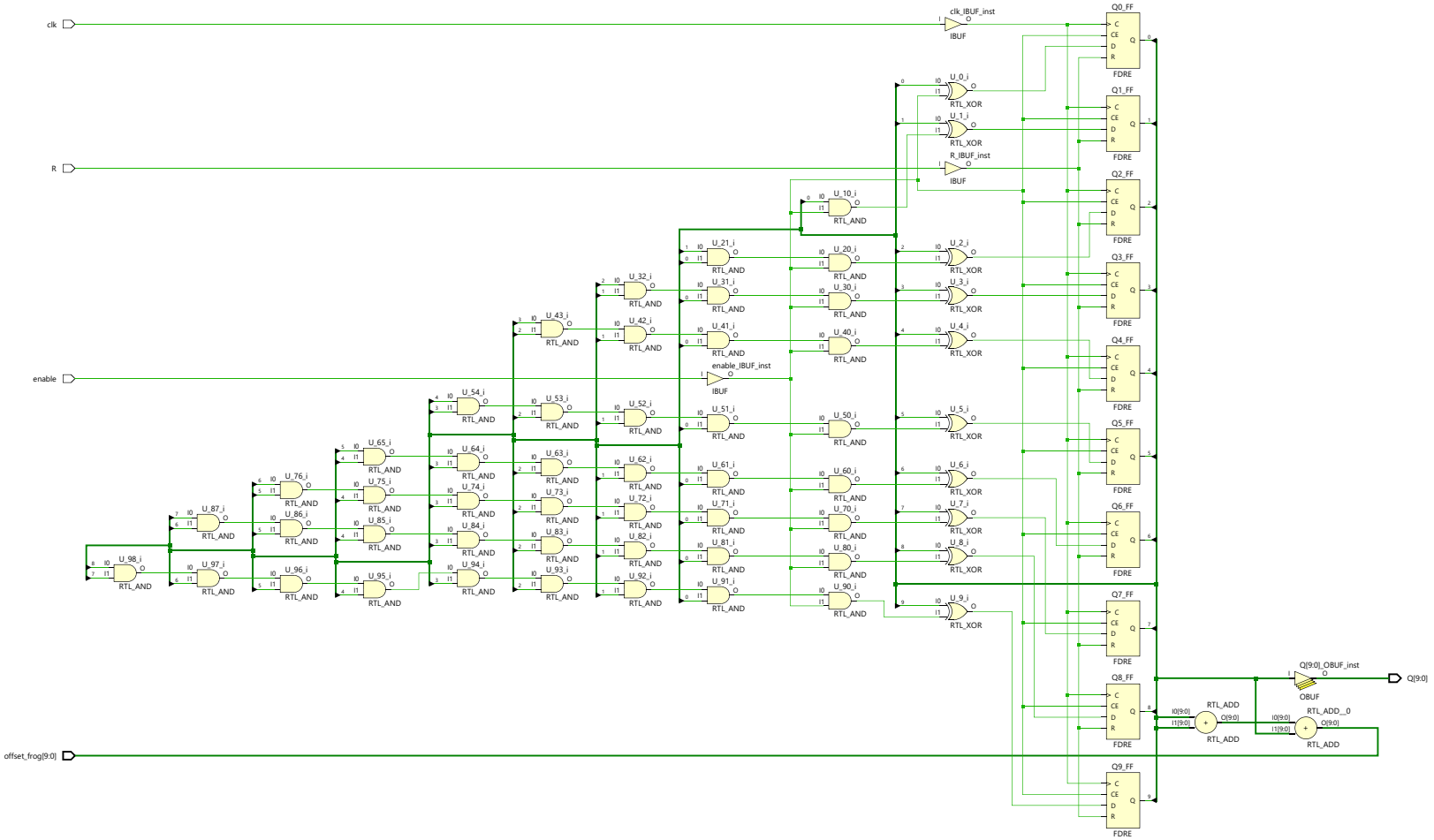
Appendix:


Q0_FF
Q1_FF
Q2_FF
Q3_FF
Q4_FF
Q5_FF
Q6_FF
Q7_FF
Q8_FF
Q9_FF
FDRE
Q[9:0]_OBUF_inst
OBUF
Q[9:0]
U_0_i
U_1_i
U_2_i
U_3_i
U_4_i
U_5_i
U_6_i
U_7_i
U_8_i
U_9_i
RTL_XOR
U_10_i
U_20_i
U_30_i
U_40_i
U_50_i
U_60_i
U_70_i
U_80_i
U_90_i
U_21_i
U_31_i
U_41_i
U_51_i
U_61_i
U_71_i
U_81_i
U_91_i
U_32_i
U_42_i
U_52_i
U_62_i
U_72_i
U_82_i
U_92_i
U_43_i
U_53_i
U_63_i
U_73_i
U_83_i
U_93_i
U_54_i
U_64_i
U_74_i
U_84_i
U_94_i
U_65_i
U_75_i
U_85_i
U_95_i
U_76_i
U_86_i
U_96_i
U_87_i
U_97_i
U_98_i
RTL_AND
enable
enable_IBUF_inst
IBUF
clk
clk_IBUF_inst
R
R_IBUF_inst

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/21/2022 01:43:56 PM
// Design Name:
// Module Name: countUD4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD10L(
    input clk,
    input enable,
    input R,
    output [9:0] Q
    //output UTC,
    //output DTC
    );

    wire [9:0] U;

    //assign UTC = Q[0] & Q[1] & Q[2] & Q[3] & Q[4] & Q[5] & Q[6] &
Q[7] & Q[8] & Q[8];
```

```
    //assign DTC = ~Q[0] & ~Q[1] & ~Q[2] & ~Q[3] & ~Q[4] & ~Q[5] &
~Q[6] & ~Q[7] & ~Q[8] & ~Q[9];

    FDRE #(.INIT(1'b0) ) Q0_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[0]), .D(U[0]));
    FDRE #(.INIT(1'b0) ) Q1_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[1]), .D(U[1]));
    FDRE #(.INIT(1'b0) ) Q2_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[2]), .D(U[2]));
    FDRE #(.INIT(1'b0) ) Q3_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[3]), .D(U[3]));
    FDRE #(.INIT(1'b0) ) Q4_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[4]), .D(U[4]));
    FDRE #(.INIT(1'b0) ) Q5_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[5]), .D(U[5]));
    FDRE #(.INIT(1'b0) ) Q6_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[6]), .D(U[6]));
    FDRE #(.INIT(1'b0) ) Q7_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[7]), .D(U[7]));
    FDRE #(.INIT(1'b0) ) Q8_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[8]), .D(U[8]));
    FDRE #(.INIT(1'b0) ) Q9_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[9]), .D(U[9]));

    // increment
    assign U[0] = Q[0] ^ enable;
    assign U[1] = Q[1] ^ (Q[0] & enable);                  // Q[1]
XOR Q[0]
    assign U[2] = Q[2] ^ (Q[1] & Q[0] & enable);           // Q[2]
XOR (Q[1] AND Q[0])
    assign U[3] = Q[3] ^ (Q[2] & Q[1] & Q[0] & enable);    // Q[3]
XOR (Q[1] AND Q[0] AND Q[3])
    assign U[4] = Q[4] ^ (Q[3] & Q[2] & Q[1] & Q[0] & enable);
    assign U[5] = Q[5] ^ (Q[4] & Q[3] & Q[2] & Q[1] & Q[0] &
enable);
    assign U[6] = Q[6] ^ (Q[5] & Q[4] & Q[3] & Q[2] & Q[1] & Q[0] &
enable);
```

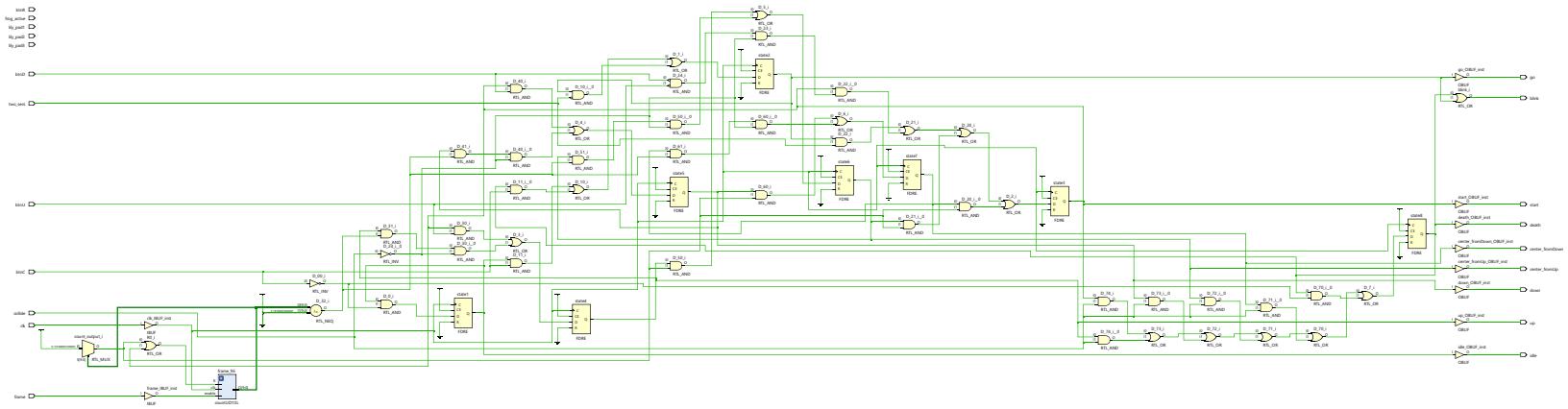
```
    assign U[7] = Q[7] ^ (Q[6] & Q[5] & Q[4] & Q[3] & Q[2] & Q[1] &
Q[0] & enable);
    assign U[8] = Q[8] ^ (Q[7] & Q[6] & Q[5] & Q[4] & Q[3] & Q[2] &
Q[1] & Q[0] & enable);
    assign U[9] = Q[9] ^ (Q[8] & Q[7] & Q[6] & Q[5] & Q[4] & Q[3] &
Q[2] & Q[1] & Q[0] & enable);

    //decrement
    //assign D[0] = Q[0] ^ enable;
    //assign D[1] = Q[1] ^ (~Q[0] & enable);
      // Q[1] XOR Q[0]
    //assign D[2] = Q[2] ^ (~Q[1] & ~Q[0] & enable);
      // Q[2] XOR (Q[1] AND Q[0])
    //assign D[3] = Q[3] ^ (~Q[2] & ~Q[1] & ~Q[0] & enable);
      // Q[3] XOR (Q[1] AND Q[0] AND Q[3])
    //assign D[4] = Q[4] ^ (~Q[3] & ~Q[2] & ~Q[1] & ~Q[0] &
enable);
    //assign D[5] = Q[5] ^ (~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0] &
enable);
    //assign D[6] = Q[6] ^ (~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] &
~Q[0] & enable);
    //assign D[7] = Q[7] ^ (~Q[6] & ~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] &
~Q[1] & ~Q[0] & enable);
    //assign D[8] = Q[8] ^ (~Q[7] & ~Q[6] & ~Q[5] & ~Q[4] & ~Q[3] &
~Q[2] & ~Q[1] & ~Q[0] & enable);
    //assign D[9] = Q[9] ^ (~Q[8] & ~Q[7] & ~Q[6] & ~Q[5] & ~Q[4] &
~Q[3] & ~Q[2] & ~Q[1] & ~Q[0] & enable);

endmodule
```


clk
R
enable
offset_frog[9:0]
clk_IBUF_inst
R_IBUF_inst
enable_IBUF_inst
IBUF
Q[9:0]_OBUF_inst
OBUF
Q[9:0]
RTL_ADD
RTL_ADD__0
I0[9:0]
I1[9:0]
O[9:0]
Q0_FF
Q1_FF
Q2_FF
Q3_FF
Q4_FF
Q5_FF
Q6_FF
Q7_FF
Q8_FF
Q9_FF
FDRE
RTL_XOR
RTL_AND

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/21/2022 01:43:56 PM
// Design Name:
// Module Name: countUD4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////


module frog_counter(
    input clk,
    input enable,
    input R,
    input [9:0] offset_frog,
    output [9:0] Q
    //output UTC,
    //output DTC
    );

    wire [9:0] U;

    //assign UTC = Q[0] & Q[1] & Q[2] & Q[3] & Q[4] & Q[5] & Q[6] &
```

```
Q[7] & Q[8] & Q[8];
    //assign DTC = ~Q[0] & ~Q[1] & ~Q[2] & ~Q[3] & ~Q[4] & ~Q[5] &
~Q[6] & ~Q[7] & ~Q[8] & ~Q[9];

    FDRE #(.INIT(1'b0) ) Q0_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[0]), .D(U[0]));
    FDRE #(.INIT(1'b0) ) Q1_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[1]), .D(U[1]));
    FDRE #(.INIT(1'b0) ) Q2_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[2]), .D(U[2]));
    FDRE #(.INIT(1'b0) ) Q3_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[3]), .D(U[3]));
    FDRE #(.INIT(1'b0) ) Q4_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[4]), .D(U[4]));
    FDRE #(.INIT(1'b0) ) Q5_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[5]), .D(U[5]));
    FDRE #(.INIT(1'b0) ) Q6_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[6]), .D(U[6]));
    FDRE #(.INIT(1'b0) ) Q7_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[7]), .D(U[7]));
    FDRE #(.INIT(1'b0) ) Q8_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[8]), .D(U[8]));
    FDRE #(.INIT(1'b0) ) Q9_FF (.C(clk), .R(R), .CE(enable),
.Q(Q[9]), .D(U[9]));

    // increment
    assign U[0] = Q[0] ^ enable;
    assign U[1] = Q[1] ^ (Q[0] & enable);                  // Q[1]
XOR Q[0]
    assign U[2] = Q[2] ^ (Q[1] & Q[0] & enable);           // Q[2]
XOR (Q[1] AND Q[0])
    assign U[3] = Q[3] ^ (Q[2] & Q[1] & Q[0] & enable);    // Q[3]
XOR (Q[1] AND Q[0] AND Q[3])
    assign U[4] = Q[4] ^ (Q[3] & Q[2] & Q[1] & Q[0] & enable);
    assign U[5] = Q[5] ^ (Q[4] & Q[3] & Q[2] & Q[1] & Q[0] &
enable);
    assign U[6] = Q[6] ^ (Q[5] & Q[4] & Q[3] & Q[2] & Q[1] & Q[0] &
```

```
enable);
    assign U[7] = Q[7] ^ (Q[6] & Q[5] & Q[4] & Q[3] & Q[2] & Q[1] &
Q[0] & enable);
    assign U[8] = Q[8] ^ (Q[7] & Q[6] & Q[5] & Q[4] & Q[3] & Q[2] &
Q[1] & Q[0] & enable);
    assign U[9] = Q[9] ^ (Q[8] & Q[7] & Q[6] & Q[5] & Q[4] & Q[3] &
Q[2] & Q[1] & Q[0] & enable);

    assign offset_frog = Q + Q + Q;

    //decrement
    //assign D[0] = Q[0] ^ enable;
    //assign D[1] = Q[1] ^ (~Q[0] & enable);
      // Q[1] XOR Q[0]
    //assign D[2] = Q[2] ^ (~Q[1] & ~Q[0] & enable);
      // Q[2] XOR (Q[1] AND Q[0])
    //assign D[3] = Q[3] ^ (~Q[2] & ~Q[1] & ~Q[0] & enable);
      // Q[3] XOR (Q[1] AND Q[0] AND Q[3])
    //assign D[4] = Q[4] ^ (~Q[3] & ~Q[2] & ~Q[1] & ~Q[0] &
enable);
    //assign D[5] = Q[5] ^ (~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0] &
enable);
    //assign D[6] = Q[6] ^ (~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] &
~Q[0] & enable);
    //assign D[7] = Q[7] ^ (~Q[6] & ~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] &
~Q[1] & ~Q[0] & enable);
    //assign D[8] = Q[8] ^ (~Q[7] & ~Q[6] & ~Q[5] & ~Q[4] & ~Q[3] &
~Q[2] & ~Q[1] & ~Q[0] & enable);
    //assign D[9] = Q[9] ^ (~Q[8] & ~Q[7] & ~Q[6] & ~Q[5] & ~Q[4] &
~Q[3] & ~Q[2] & ~Q[1] & ~Q[0] & enable);

endmodule
```


look
freq_active
lfg_pad1
lfg_pad2
lfg_pad3
toss2
two_sec
toss1
toss0
collide
clk
frame
RTL_AND
RTL_OR
RTL_INV
RTL_MUX
RTL_NEQ
FDRE
OBUF
IBUF
BUFG
state0
state1
state2
state3
state4
state5
state6
state7
go
blink
start
death
center_fromDown
center_fromUp
down
up
idle

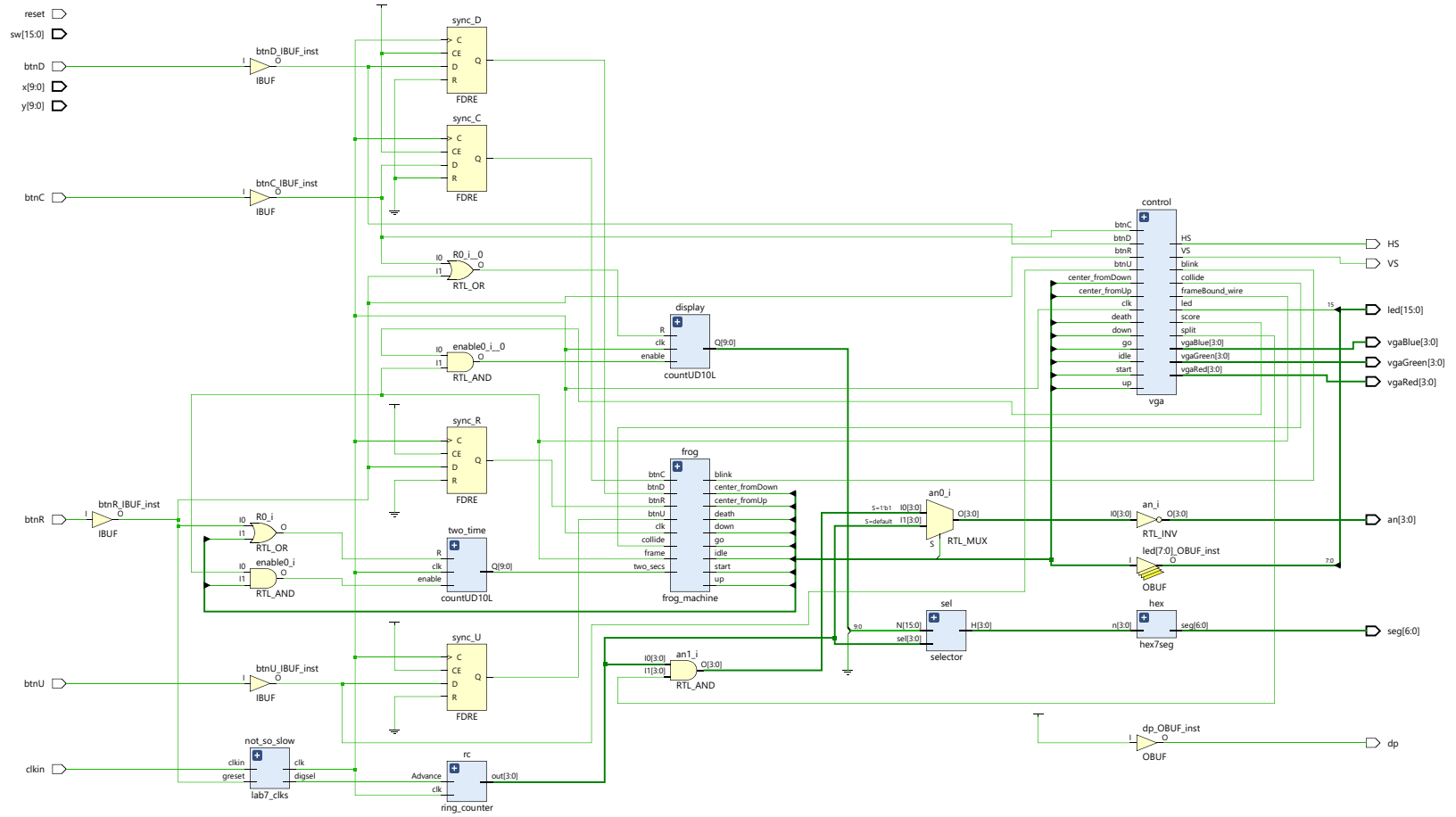
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/22/2022 01:40:01 PM
// Design Name:
// Module Name: frog_machine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module frog_machine(
    input clk,
    input btnC,
    input btnU,
    input btnD,
    input btnR,

    input frame,
    input two_secs,
    input collide,

    input frog_active,
    input lily_pad1,
```

```
    input lily_pad2,
    input lily_pad3,

    output idle,
    output go,
    output start,
    output up,
    output down,
    output center_fromUp,
    output center_fromDown,
    output death,
    output blink

    );

    wire [7:0] D, Q;
    wire [9:0] count_output;

    assign idle = Q[0];
    assign go = Q[1];
    assign start = Q[2];
    assign up = Q[3];
    assign down = Q[4];
    assign center_fromUp = Q[5];
    assign center_fromDown = Q[6];
    assign death = Q[7];
    assign blink = (Q[7]) | (Q[1]);


    assign D[0] = (Q[0] & ~btnC);
    assign D[1] = (Q[0] & btnC) | (Q[7] & btnC) | (Q[1] &
~two_secs);
    assign D[2] = (Q[1] & two_secs) | (Q[2] & (~btnD & ~btnU &
~collide)) | (Q[5] & count_output == 10'd32) | (Q[6] & count_output
== 10'd32);
    assign D[3] = (Q[2] & btnU) | (Q[3] & count_output != 10'd32 &
~collide);
```

```
    assign D[4] = (Q[2] & btnD) | (Q[4] & count_output != 10'd32 &
~collide);
    assign D[5] = (Q[3] & count_output == 10'd32) | (Q[5] &
count_output != 10'd32 & ~collide);
    assign D[6] = (Q[4] & count_output == 10'd32) | (Q[6] &
count_output != 10'd32 & ~collide);
    assign D[7] = ((Q[2] & collide) | (Q[3] & collide) | (Q[4] &
collide) | (Q[5] & collide) | (Q[6] & collide) | (Q[7] & ~btnC));

    FDRE #(.INIT(1'b1) ) state1 (.C(clk), .CE(1'b1), .Q(Q[0]),
.D(D[0]));
    FDRE #(.INIT(1'b0) ) state2 (.C(clk), .CE(1'b1), .Q(Q[1]),
.D(D[1]));
    FDRE #(.INIT(1'b0) ) state3 (.C(clk), .CE(1'b1), .Q(Q[2]),
.D(D[2]));
    FDRE #(.INIT(1'b0) ) state4 (.C(clk), .CE(1'b1), .Q(Q[3]),
.D(D[3]));
    FDRE #(.INIT(1'b0) ) state5 (.C(clk), .CE(1'b1), .Q(Q[4]),
.D(D[4]));
    FDRE #(.INIT(1'b0) ) state6 (.C(clk), .CE(1'b1), .Q(Q[5]),
.D(D[5]));
    FDRE #(.INIT(1'b0) ) state7 (.C(clk), .CE(1'b1), .Q(Q[6]),
.D(D[6]));
    FDRE #(.INIT(1'b0) ) state8 (.C(clk), .CE(1'b1), .Q(Q[7]),
.D(D[7]));

    countUD10L frame_96 (.clk(clk), .enable(frame), .R(count_output
== 10'd32 | (Q[2])), .Q(count_output));

endmodule
```

```
//    input clk,
//    input btnC,
//    input btnU,
//    input btnD,
//    input btnR,
//    input frame,

//    output idle,
//    output start,
//    output up,
//    output down,
//    output center_fromUp,
//    output center_fromDown

//    );

//    wire [5:0] D, Q;
//    wire [9:0] count_output;

//    assign idle = Q[0];
//    assign start = Q[1];
//    assign up = Q[2];
//    assign down = Q[3];
//    assign center_fromUp = Q[4];
//    assign center_fromDown = Q[5];


//    assign D[0] = (Q[0] & ~btnC);
//    assign D[1] = (Q[0] & btnC) | (Q[1] & (~btnD & ~btnU)) |
(Q[4] & (count_output == 10'd32)) | (Q[5] & (count_output ==
10'd32) );
//    assign D[2] = (Q[1] & btnU) | (Q[2] & (count_output !=
10'd32));
//    assign D[3] = (Q[1] & btnD) | (Q[3] & (count_output !=
10'd32));
//    assign D[4] = (Q[2] & (count_output == 10'd32)) | (Q[4] &
(count_output != 10'd32));
```

```
//    assign D[5] = (Q[3] & (count_output == 10'd32)) | (Q[5] &
(count_output != 10'd32));

//    FDRE #(.INIT(1'b1) ) state1 (.C(clk), .CE(1'b1), .Q(Q[0]),
.D(D[0]));
//    FDRE #(.INIT(1'b0) ) state2 (.C(clk), .CE(1'b1), .Q(Q[1]),
.D(D[1]));
//    FDRE #(.INIT(1'b0) ) state3 (.C(clk), .CE(1'b1), .Q(Q[2]),
.D(D[2]));
//    FDRE #(.INIT(1'b0) ) state4 (.C(clk), .CE(1'b1), .Q(Q[3]),
.D(D[3]));
//    FDRE #(.INIT(1'b0) ) state5 (.C(clk), .CE(1'b1), .Q(Q[4]),
.D(D[4]));
//    FDRE #(.INIT(1'b0) ) state6 (.C(clk), .CE(1'b1), .Q(Q[5]),
.D(D[5]));

//    countUD10L frame_96 (.clk(clk), .enable(frame),
.R(count_output == 10'd32 | (Q[1])), .Q(count_output));

//endmodule
```

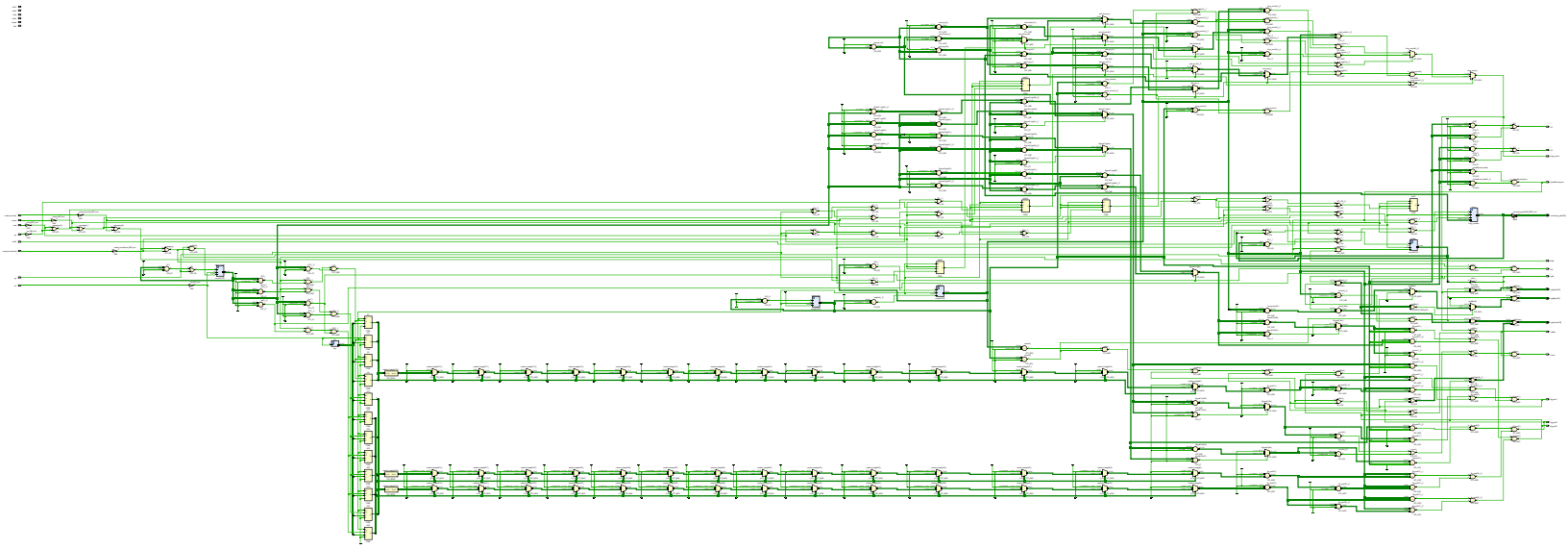

reset
sw[15:0]
btnD
x[9:0]
y[9:0]
btnC
btnR
btnU
clkin
btnD_IBUF_inst
IBUF
btnC_IBUF_inst
IBUF
btnR_IBUF_inst
IBUF
btnU_IBUF_inst
IBUF
sync_D
FDRE
sync_C
FDRE
sync_R
FDRE
sync_U
FDRE
R0_i__0
RTL_OR
enable0_i__0
RTL_AND
R0_i
RTL_OR
enable0_i
RTL_AND
display
countUD10L
two_time
countUD10L
frog
frog_machine
control
vga
an0_i
RTL_MUX
an_i
RTL_INV
led[7:0]_OBUF_inst
OBUF
sel
selector
hex
hex7seg
an1_i
RTL_AND
not_so_slow
lab7_clks
rc
ring_counter
dp_OBUF_inst
OBUF
HS
VS
led[15:0]
vgaBlue[3:0]
vgaGreen[3:0]
vgaRed[3:0]
an[3:0]
seg[6:0]
dp

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2022 01:03:38 PM
// Design Name:
// Module Name: top_mod
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_mod(
    input clkin,
    input btnC,
    input btnU,
    input btnD,
    input btnR,
    input reset,
    input [9:0] x,
    input [9:0] y,
    input [15:0] sw,

    output HS,
    output VS,
```

```
output [3:0] vgaRed,
output [3:0] vgaBlue,
output [3:0] vgaGreen,
output [6:0] seg,
output [3:0] an,
output [15:0] led,
output dp

);

wire clk;
wire digsel;
wire [3:0] sel_wire;
wire [6:0] seg_wire;
wire [3:0] ring_wire;
wire [15:0] sel_input;

wire idle_wire;
wire start_wire;
wire go_wire;
wire up_wire;
wire down_wire;
wire cfu_wire;
wire cfd_wire;
wire death_wire;

wire btnd_wire;
wire btnu_wire;
wire btnc_wire;
wire btnr_wire;

assign led[0] = idle_wire;
assign led[1] = go_wire;
assign led[2] = start_wire;
assign led[3] = up_wire;
assign led[4] = down_wire;
assign led[5] = cfu_wire;
```

```
    assign led[6] = cfd_wire;
    assign led[7] = death_wire;


    wire HS_wire;
    wire VS_wire;
    wire frame;
    wire collide;
    wire blink;
    wire two_secs;
    wire [11:0] frame_two;
    wire [9:0] movefrog_wire;
    wire [9:0] display_counter;
    wire score;
    wire split;

    countUD10L two_time (.clk(clk), .enable(frame & go_wire),
.R(btnR | start_wire), .Q(frame_two));
    assign two_secs = frame_two[7];

    FDRE #(.INIT(1'b0) ) sync_D (.C(clk), .CE(1'b1), .Q(btnd_wire),
.D(btnD));
    FDRE #(.INIT(1'b0) ) sync_U (.C(clk), .CE(1'b1), .Q(btnu_wire),
.D(btnU));
    FDRE #(.INIT(1'b0) ) sync_R (.C(clk), .CE(1'b1), .Q(btnr_wire),
.D(btnR));
    FDRE #(.INIT(1'b0) ) sync_C (.C(clk), .CE(1'b1), .Q(btnc_wire),
.D(btnC));

    assign an = ~(death_wire ? ring_wire & {4{split}} : ring_wire);
    assign seg = seg_wire;
    assign dp = 1'b1;

    vga control (.clk(clk), .VS(VS), .HS(HS), .vgaRed(vgaRed),
.vgaBlue(vgaBlue), .vgaGreen(vgaGreen),.btnC(btnC), .btnD(btnD),
.btnU(btnU), .btnR(btnR), .idle(idle_wire), .start(start_wire),
.up(up_wire), .down(down_wire), .center_fromUp(cfu_wire),
```

```
.center_fromDown(cfd_wire), .frameBound_wire(frame),
.collide(collide), .blink(blink), .death(death_wire), .go(go_wire),
.score(score), .split(split), .led(led[15]));

    frog_machine frog (.clk(clk), .btnC(btnc_wire),
.btnD(btnd_wire), .btnU(btnu_wire), .btnR(btnr_wire),
.idle(idle_wire), .start(start_wire), .go(go_wire), .up(up_wire),
.down(down_wire), .center_fromUp(cfu_wire),
.center_fromDown(cfd_wire), .collide(collide), .death(death_wire),
.frame(frame), .lily_pad1(lily_pad1), .lily_pad2(lily_pad2),
.lily_pad3(lily_pad3), .frog_active(frog_active),
.two_secs(two_secs), .blink(blink));

    lab7_clks not_so_slow (.clkin(clkin), .greset(btnR), .clk(clk),
.digsel(digsel));

    hex7seg hex (.n(sel_wire), .seg(seg_wire));

    ring_counter rc (.Advance(digsel), .clk(clk), .out(ring_wire));

    selector sel (.N(display_counter), .sel(ring_wire),
.H(sel_wire));

    countUD10L display (.clk(clk), .enable(score & frame), .R(btnC
| btnR), .Q(display_counter));

endmodule
```



```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2022 12:38:28 PM
// Design Name:
// Module Name: vga
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////


module vga(
    input clk,
    input btnL,
    input btnC,
    input btnU,
    input btnD,
    input btnR,

    input idle,
    input start,
    input up,
    input down,
    input center_fromUp,
```

```
input center_fromDown,
input death,
input go,

output led,

output HS,
output VS,
output [3:0] vgaRed, // 4 bit colors (12 in total)
output [3:0] vgaBlue,
output [3:0] vgaGreen,
output frameBound_wire,
output lily_pad1,
output lily_pad2,
output lily_pad3,
output frog_active,
output collide,
output blink,
output score,
output split,
output [9:0] movefrog_wire
);

wire [9:0] out_h;
wire [9:0] out_v;

wire active;
assign frameBound_wire = out_h == 10'd642 & out_v == 10'd482;

wire water;
wire [3:0] deep_blue;
wire [9:0] compute_diff;

wire [9:0] plant_out;
wire [9:0] blink_out;

wire [9:0] offset_frog;
```

```
    assign VS = out_v < 10'd489 | out_v > 10'd490;
    assign HS = out_h < 10'd655 | out_h > 10'd750;
    assign active = out_h <= 10'd639 & out_v <= 10'd479;
    assign compute_diff = (out_v - 10'd240);
    assign deep_blue = (4'hf - compute_diff[7:4]);
    assign water = out_v >= 10'd240 & out_h <= 10'd639;

    wire [9:0] upper_bound;
    wire [9:0] lower_bound;
    wire [9:0] frog_left;
    wire [9:0] frog_right;
    wire [9:0] wire_up_u;
    wire [9:0] wire_up_d;
    wire [9:0] wire_down_u;
    wire [9:0] wire_down_d;

    wire [9:0] death_down_u;
    wire [9:0] death_up_u;
    wire [9:0] death_down_d;
    wire [9:0] death_up_d;

    wire [4:0] hold_state;
    wire keep_frog;

    assign upper_bound = 10'd232;
    assign lower_bound = 10'd248;
    assign frog_left = 10'd120;
    assign frog_right = 10'd136;

    //up and center from up states
    assign wire_up_u = (hold_state[1]) ? upper_bound - offset_frog
: hold_state[3] ? ((upper_bound + offset_frog) - 10'd96):
upper_bound;
    assign wire_up_d = (hold_state[1]) ? lower_bound - offset_frog
: hold_state[3] ? ((lower_bound + offset_frog) - 10'd96):
lower_bound;
```

```
    //down and center from down state
    assign wire_down_u = (hold_state[2]) ? upper_bound +
offset_frog : hold_state[4] ? ((upper_bound - offset_frog) +
10'd96): upper_bound;
    assign wire_down_d = (hold_state[2]) ? lower_bound +
offset_frog : hold_state[4] ? ((lower_bound - offset_frog) +
10'd96): lower_bound;

      FDRE #(.INIT(1'b0) ) hold1 (.C(clk), .CE(start), .R(up | down
| center_fromUp | center_fromDown), .Q(hold_state[0]), .D(start));
      FDRE #(.INIT(1'b0) ) hold2 (.C(clk), .CE(up), .R(start | go |
down | center_fromUp | center_fromDown), .Q(hold_state[1]),
.D(up));
      FDRE #(.INIT(1'b0) ) hold3 (.C(clk), .CE(down), .R(start | up
| go | center_fromUp | center_fromDown), .Q(hold_state[2]),
.D(down));
      FDRE #(.INIT(1'b0) ) hold4 (.C(clk), .CE(center_fromUp),
.R(start | up | down | go | center_fromDown), .Q(hold_state[3]),
.D(center_fromUp));
      FDRE #(.INIT(1'b0) ) hold5 (.C(clk), .CE(center_fromDown),
.R(start | up | down | center_fromUp | go), .Q(hold_state[4]),
.D(center_fromDown));


    assign frog_active =
    (hold_state[1] | hold_state[3]) ? (out_h <= 10'd136 & out_h >
10'd120 & out_v >= wire_up_u & out_v < wire_up_d)
    : (hold_state[2] | hold_state[4]) ? (out_h < 10'd136 & out_h >
10'd120 & out_v >= wire_down_u & out_v < wire_down_d)
    : (out_h < 10'd136 & out_h > 10'd120 & out_v >= upper_bound &
out_v < lower_bound);

    //Random height logic
    wire [3:0]lfsr_out, pause_lfsr1, pause_lfsr2, pause_lfsr3;
    wire gameplay;
    assign gameplay = start | up | down | center_fromUp |
center_fromDown;
```

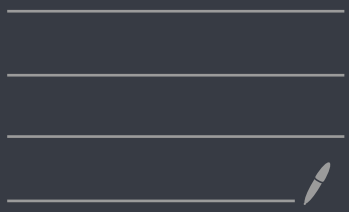
```
    rng lfsr (.clk(clk), .Q(lfsr_out));
    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .CE(gameplay & plant_out <
10'd88 & plant_out > 10'd84), .D(lfsr_out[0]), .Q(pause_lfsr1[0]));
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .CE(gameplay & plant_out <
10'd88 & plant_out > 10'd84), .D(lfsr_out[1]), .Q(pause_lfsr1[1]));
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .CE(gameplay & plant_out <
10'd88 & plant_out > 10'd84), .D(lfsr_out[2]), .Q(pause_lfsr1[2]));
    FDRE #(.INIT(1'b0) ) ff4 (.C(clk), .CE(gameplay & plant_out <
10'd88 & plant_out > 10'd84), .D(lfsr_out[3]), .Q(pause_lfsr1[3]));
    FDRE #(.INIT(1'b0) ) ff12 (.C(clk), .CE(gameplay & plant_out <
10'd168 & plant_out > 10'd164), .D(lfsr_out[0]),
.Q(pause_lfsr2[0]));
    FDRE #(.INIT(1'b0) ) ff22 (.C(clk), .CE(gameplay & plant_out <
10'd168 & plant_out > 10'd164), .D(lfsr_out[1]),
.Q(pause_lfsr2[1]));
    FDRE #(.INIT(1'b0) ) ff32 (.C(clk), .CE(gameplay & plant_out <
10'd168 & plant_out > 10'd164), .D(lfsr_out[2]),
.Q(pause_lfsr2[2]));
    FDRE #(.INIT(1'b0) ) ff42 (.C(clk), .CE(gameplay & plant_out <
10'd168 & plant_out > 10'd164), .D(lfsr_out[3]),
.Q(pause_lfsr2[3]));
    FDRE #(.INIT(1'b0) ) ff13 (.C(clk), .CE(gameplay & plant_out <
10'd244 & plant_out > 10'd237), .D(lfsr_out[0]),
.Q(pause_lfsr3[0]));
    FDRE #(.INIT(1'b0) ) ff23 (.C(clk), .CE(gameplay & plant_out <
10'd244 & plant_out > 10'd237), .D(lfsr_out[1]),
.Q(pause_lfsr3[1]));
    FDRE #(.INIT(1'b0) ) ff33 (.C(clk), .CE(gameplay & plant_out <
10'd244 & plant_out > 10'd237), .D(lfsr_out[2]),
.Q(pause_lfsr3[2]));
    FDRE #(.INIT(1'b0) ) ff43 (.C(clk), .CE(gameplay & plant_out <
10'd244 & plant_out > 10'd237), .D(lfsr_out[3]),
.Q(pause_lfsr3[3]));

    wire [9:0]upper_height;
    wire [9:0]random_height1, random_height2, random_height3;
    assign upper_height = 10'd164;
```

```
assign random_height1 =
pause_lfsr1 == 4'h0 ? 10'd0:
pause_lfsr1 == 4'h1 ? 10'd4:
pause_lfsr1 == 4'h2 ? 10'd8:
pause_lfsr1 == 4'h3 ? 10'd12:
pause_lfsr1 == 4'h4 ? 10'd16:
pause_lfsr1 == 4'h5 ? 10'd20:
pause_lfsr1 == 4'h6 ? 10'd24:
pause_lfsr1 == 4'h7 ? 10'd28:
pause_lfsr1 == 4'h8 ? 10'd32:
pause_lfsr1 == 4'h9 ? 10'd36:
pause_lfsr1 == 4'ha ? 10'd40:
pause_lfsr1 == 4'hb ? 10'd44:
pause_lfsr1 == 4'hc ? 10'd48:
pause_lfsr1 == 4'hd ? 10'd52:
pause_lfsr1 == 4'he ? 10'd56:
10'd0;

assign random_height2 =
pause_lfsr2 == 4'h0 ? 10'd0:
pause_lfsr2 == 4'h1 ? 10'd4:
pause_lfsr2 == 4'h2 ? 10'd8:
pause_lfsr2 == 4'h3 ? 10'd12:
pause_lfsr2 == 4'h4 ? 10'd16:
pause_lfsr2 == 4'h5 ? 10'd20:
pause_lfsr2 == 4'h6 ? 10'd24:
pause_lfsr2 == 4'h7 ? 10'd28:
pause_lfsr2 == 4'h8 ? 10'd32:
pause_lfsr2 == 4'h9 ? 10'd36:
pause_lfsr2 == 4'ha ? 10'd40:
pause_lfsr2 == 4'hb ? 10'd44:
pause_lfsr2 == 4'hc ? 10'd48:
pause_lfsr2 == 4'hd ? 10'd52:
pause_lfsr2 == 4'he ? 10'd56:
10'd0;

assign random_height3 =
```

```
    pause_lfsr3 == 4'h0 ? 10'd0:
    pause_lfsr3 == 4'h1 ? 10'd4:
    pause_lfsr3 == 4'h2 ? 10'd8:
    pause_lfsr3 == 4'h3 ? 10'd12:
    pause_lfsr3 == 4'h4 ? 10'd16:
    pause_lfsr3 == 4'h5 ? 10'd20:
    pause_lfsr3 == 4'h6 ? 10'd24:
    pause_lfsr3 == 4'h7 ? 10'd28:
    pause_lfsr3 == 4'h8 ? 10'd32:
    pause_lfsr3 == 4'h9 ? 10'd36:
    pause_lfsr3 == 4'ha ? 10'd40:
    pause_lfsr3 == 4'hb ? 10'd44:
    pause_lfsr3 == 4'hc ? 10'd48:
    pause_lfsr3 == 4'hd ? 10'd52:
    pause_lfsr3 == 4'he ? 10'd56:
    10'd0;

    wire [11:0] lilypad1_left, lilypad1_right;
    wire [11:0] lilypad2_left, lilypad2_right;
    wire [11:0] lilypad3_left, lilypad3_right;
    assign lilypad1_right = plant_out > 10'd79 ? 10'd239 + 10'd720
- plant_out - plant_out - plant_out : 10'd239 - plant_out -
plant_out - plant_out;
    assign lilypad1_left = lilypad1_right < 10'd40 ? 10'd0 :
lilypad1_right - 10'd40;
    assign lily_pad1 = (lilypad1_left <= out_h & lilypad1_right
>=out_h) & (out_v >= upper_height + random_height1 & out_v <=
upper_height + random_height1 + 10'd96);

    assign lilypad2_right = plant_out > 10'd159 ? 10'd479 + 10'd720
- plant_out - plant_out - plant_out : 10'd479 - plant_out -
plant_out - plant_out;
    assign lilypad2_left = lilypad2_right < 10'd40 ? 10'd0 :
lilypad2_right - 10'd40;
    assign lily_pad2 = (lilypad2_left <= out_h & lilypad2_right
>=out_h) & (out_v >= upper_height + random_height2 & out_v <=
upper_height + random_height2 + 10'd96);
```

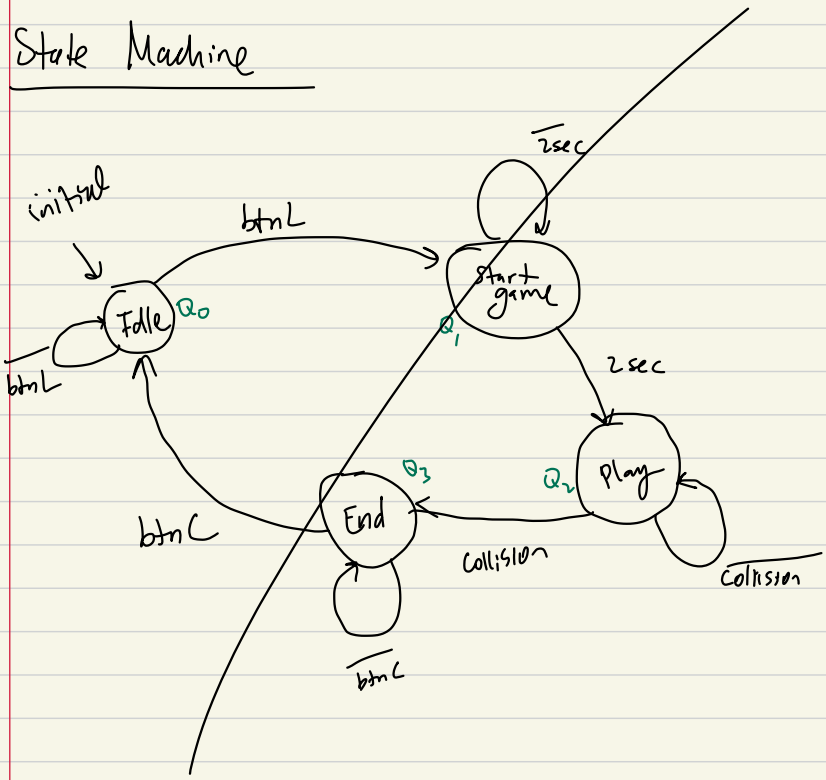
```
    assign lilypad3_right = plant_out > 10'd237 ? 10'd719 + 10'd720
- plant_out - plant_out - plant_out : 10'd719 - plant_out -
plant_out - plant_out;
    assign lilypad3_left = lilypad3_right < 10'd40 ? 10'd0 :
lilypad3_right - 10'd40;
    assign lily_pad3 = (lilypad3_left <= out_h & lilypad3_right >=
out_h) & (out_v >= upper_height + random_height3 & out_v <=
upper_height + random_height3 + 10'd96);



    assign led = gameplay & plant_out > 10'd237 & plant_out <
10'd244;
    //assign lily_pad1 = (plant_out < 10'd67 ? (out_h >= (10'd200 -
plant_out - plant_out - plant_out)) : out_h >= 10'd0) & (out_h <=
(10'd240 - plant_out - plant_out - plant_out)) & (out_v >
upper_height + random_height1) & (out_v < upper_height +
random_height1 + 10'd96);
    //assign lily_pad2 = (out_h >= (10'd440 - plant_out - plant_out
- plant_out)) & (out_h <= (10'd480 - plant_out - plant_out -
plant_out)) & ((out_v > upper_height + random_height1) & (out_v <
upper_height + random_height1 + 10'd96));
    //assign lily_pad3 = (out_h >= (10'd680 - plant_out - plant_out
- plant_out)) & (out_h <= (10'd720 - plant_out - plant_out -
plant_out)) & ((out_v > upper_height + random_height1) & (out_v <
upper_height + random_height1 + 10'd96));

    assign split = ~blink_out[5];
    assign score = (lilypad1_left == 10'd40 | lilypad2_left ==
10'd40 | lilypad3_left == 10'd40) & ~collide;

    assign vgaRed = {4{active}} & ({4{frog_active}}) & {4{split}};
    assign vgaBlue = {4{active}} & (({4{frog_active}}) &
{4{split}}? 10'hf : (water & ~lily_pad1 & ~lily_pad2 & ~lily_pad3)
? deep_blue : 10'h0);
    assign vgaGreen = {4{active}} & (({4{frog_active}}) &
{4{split}}) | ({4{lily_pad1}} | {4{lily_pad2}} |
```

```
{4{lily_pad3}});//? 10'hf: lily_pad1 ? 10'hf : lily_pad2 ? 4'hf :
lily_pad3 ? 10'hf : 10'h0);
    assign collide = frog_active & (lily_pad1 | lily_pad2 |
lily_pad3);

   countUD10L vertical (.clk(clk), .enable(out_h == 10'd799),
.R(out_v == 10'd524), .Q(out_v));
   countUD10L horizontal (.clk(clk), .enable(1'b1), .R(out_h >
10'd799), .Q(out_h));
   countUD10L plant_count (.clk(clk), .enable(frameBound_wire &
(start | up | down | center_fromUp | center_fromDown)),
.R(plant_out > 10'd239 | go), .Q(plant_out));
   frog_counter frog (.clk(clk), .enable(frameBound_wire & (up |
down | center_fromDown | center_fromUp)), .R(movefrog_wire ==
10'd32 | go | start), .Q(movefrog_wire),
.offset_frog(offset_frog));
   countUD10L blinker (.clk(clk), .enable(frameBound_wire & blink),
.R(btnR | blink_out[7] | start), .Q(blink_out));

endmodule
```

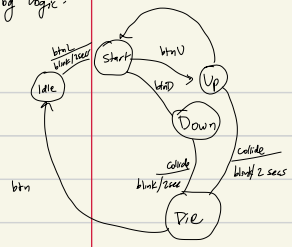


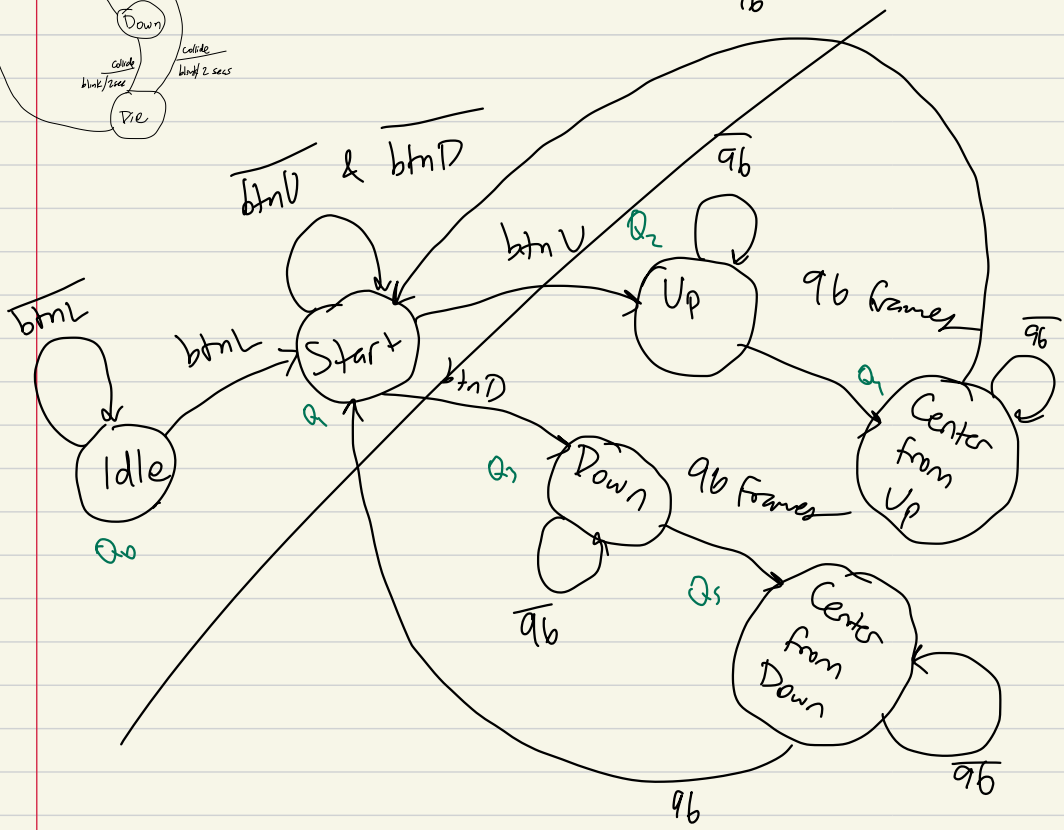
VGA Controller

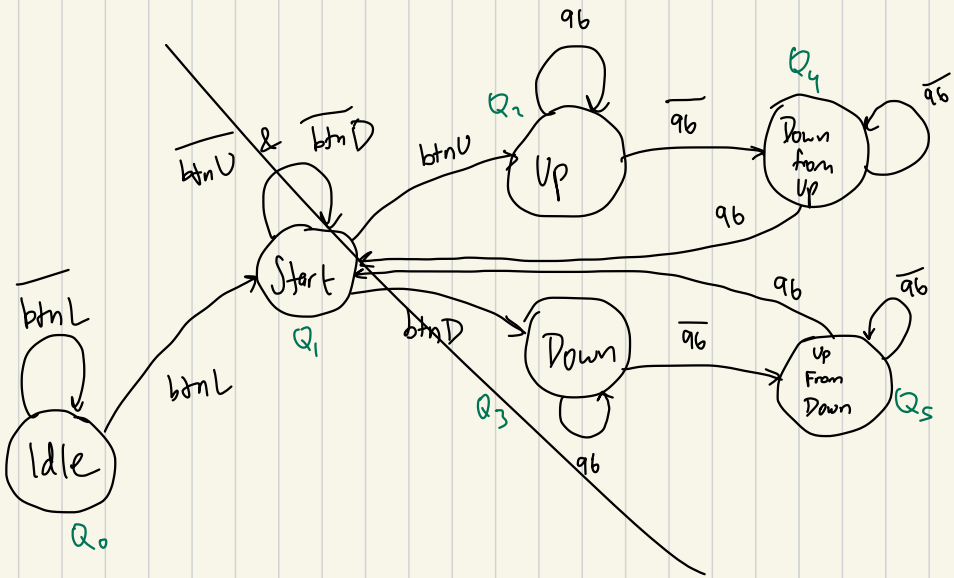
State Machine

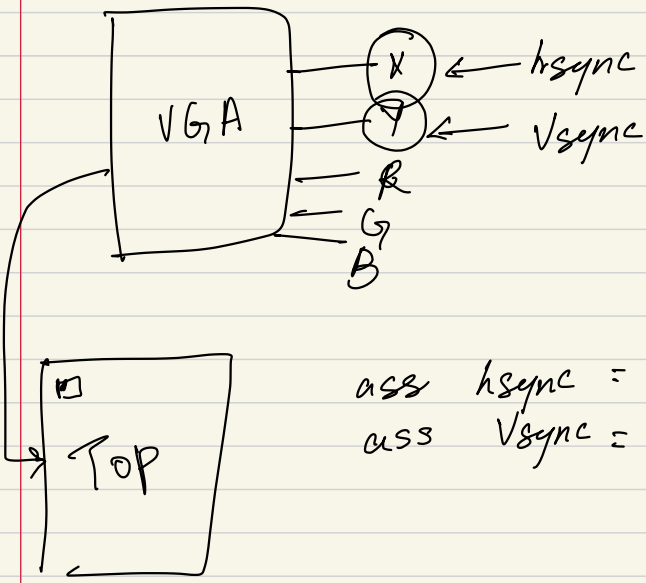


Frog logic:


Start
btnL
blink/2sec
Idle
btnU
Up
btnD
Down
collide
blink/2sec
collide
blink/2 secs
Die
btn


btnU & btnD
96
96
btnU
Q2
Up
96 frames
96
btnL
btnL
Start
Q1
Center from Up
btnD
Q0
Idle
Q3
Down
96 frames
Q4
96
Center from Down
96
96

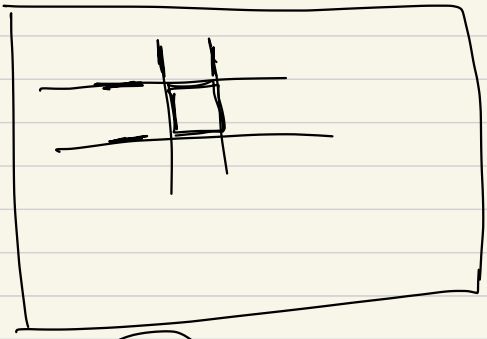

Idle
Q0
btnL
btnL
Start
Q1
btnU & btnD
btnU
Up
Q2
96
96
Down from Up
Q4
96
96
Down
Q3
96
btnD
96
Up From Down
Q5
96
96



ass hsync =

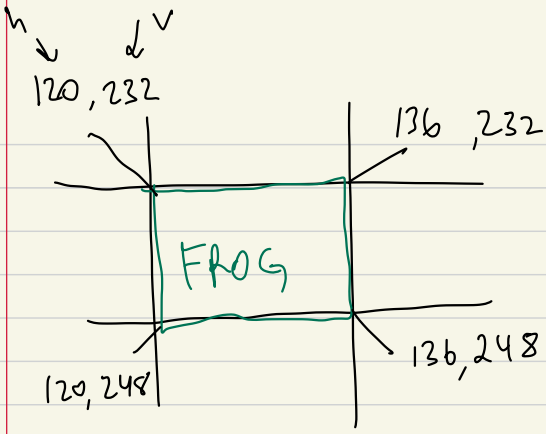
ass Vsync = Vertical counter

active region



sel ? way1 : way0

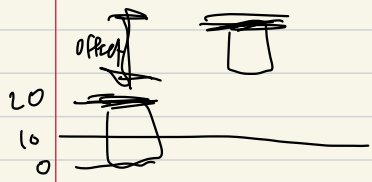
X



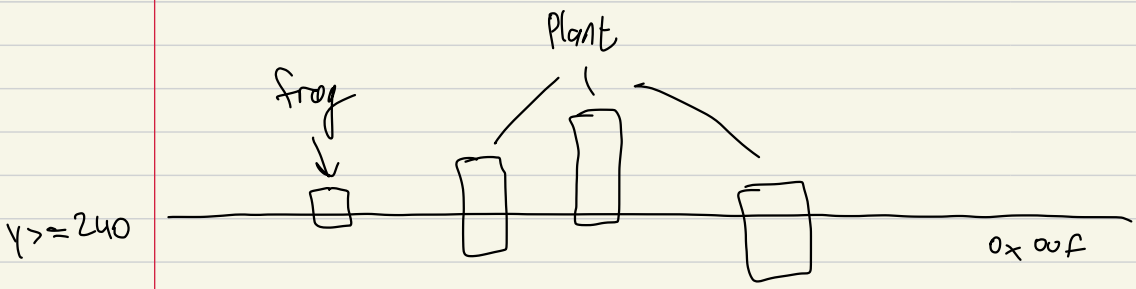
assign vgaRed = frog ? 4'hF : 4'h0

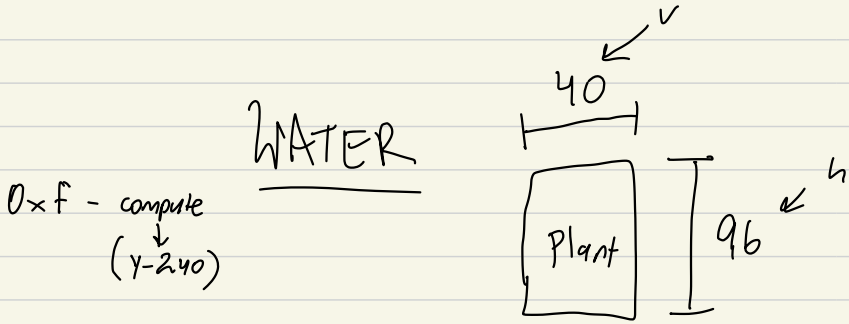
(condition) ? expression : ____

hierarchy

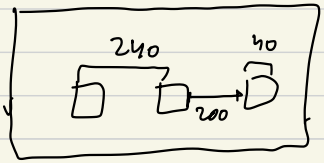


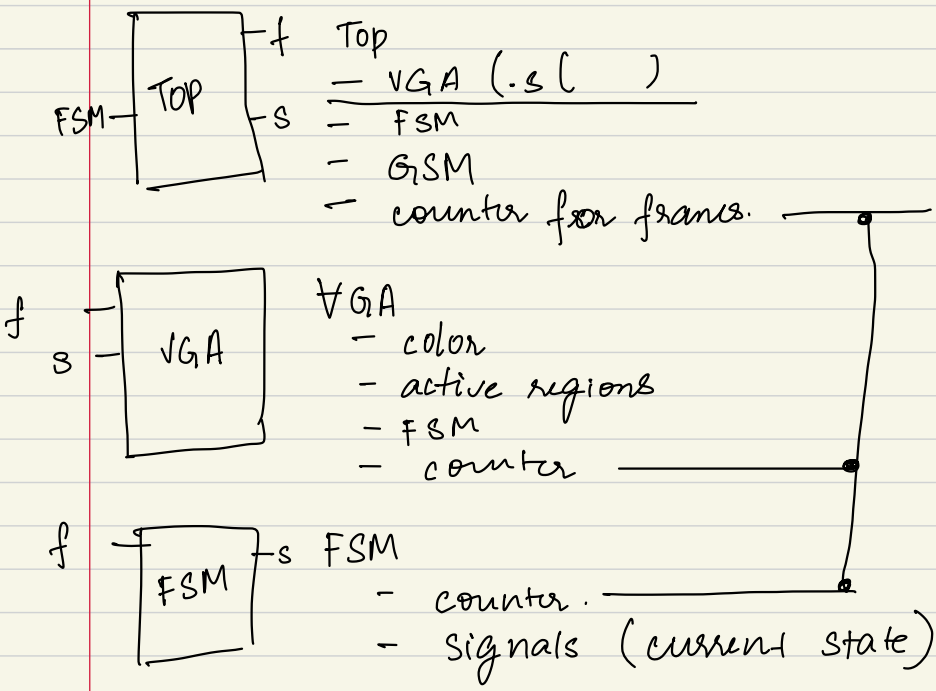
position_y = 20 + offset

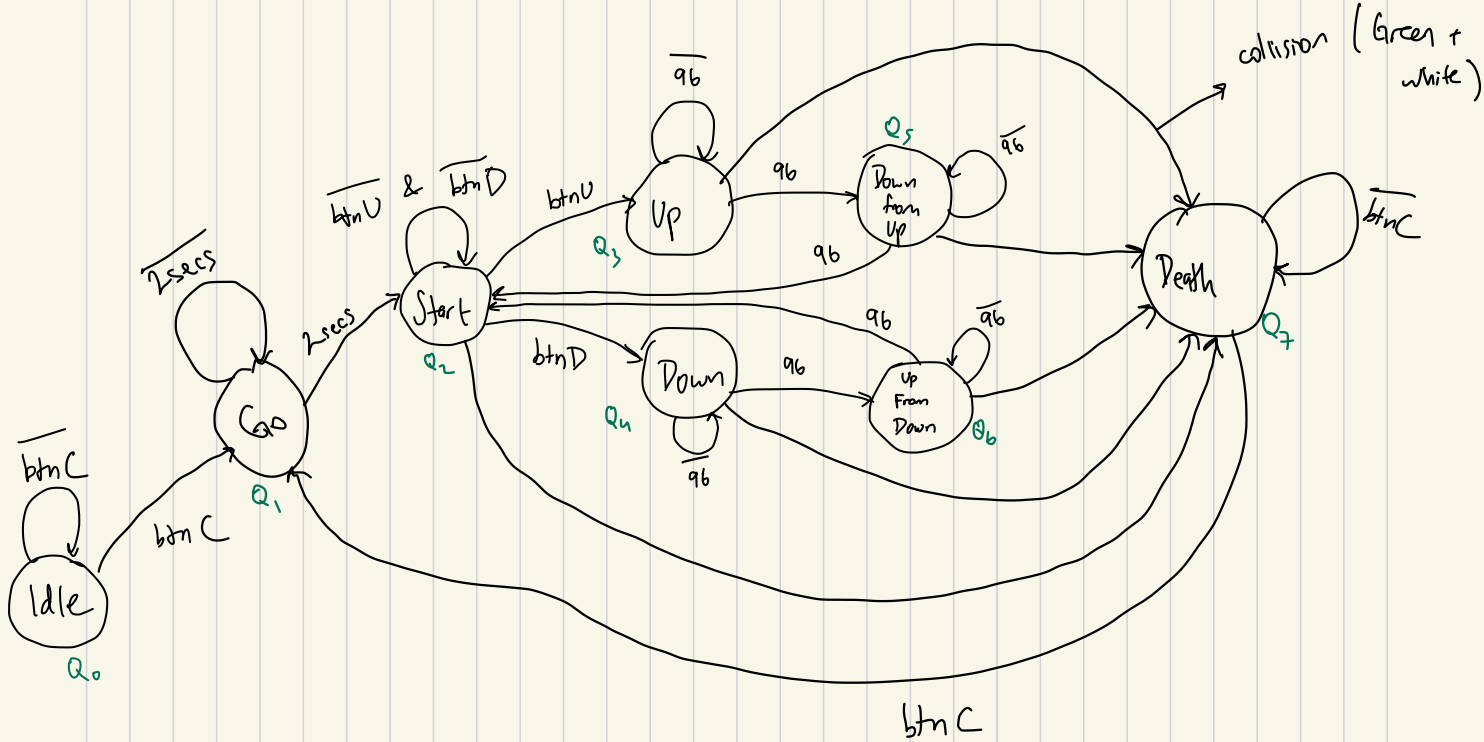




1 $0+240 = 240$
2 $200 + 240 = 440$
3 $400 + 240 = 640$




TOP
f
s
FSM
Top
- VGA (.s( )
- FSM
- GSM
- counter for frames.
VGA
f
s
VGA
- color
- active regions
- FSM
- counter
FSM
f
s
FSM
- counter.
- signals (current state)


Idle
Q0
btnC
btnC
Go
Q1
2secs
2secs
Start
Q2
btnU & btnD
btnU
Up
Q3
96
96
Down from Up
Q5
96
96
btnD
Down
Q4
96
96
Up From Down
Q6
96
96
Death
Q7
btnC
collision (Green + white)
btnC

blink : use counter then 32 frames frog
& 32 frames ~frog

assign blink = $\overbrace{((\text{Present States}) \,\&\, (\text{Next States}))}^{\text{Death}}$ | Present State (Q)
(Q) (D) Go

$D[0] = Q[0] \,\&\, \sim btnC$

$D[1] = (Q[0] \,\&\, btnC) \mid Q[1] \,\&\, \sim two\text{-}secs$

$D[2] = (Q[1] \,\&\, secs) \mid (Q[2] \,\&\, (\sim btnD \,\&\, \sim btnU)) \mid (Q[5] \,\&\, 96 \mid Q[4] \,\&\, 96)$

$D[3] = (Q[2] \,\&\, btnU) \mid (Q[3] \,\&\, \sim 96)$

$D[4] = (Q[2] \,\&\, btnD) \mid (Q[4] \,\&\, \sim 96)$

$D[5] = (Q[3] \,\&\, 96) \mid (Q[5] \,\&\, \sim 96)$

$D[6] = (Q[4] \,\&\, 96) \mid (Q[6] \,\&\, \sim 96)$

$D[7] = (\text{all 5 states collision}) \mid$

BLINK

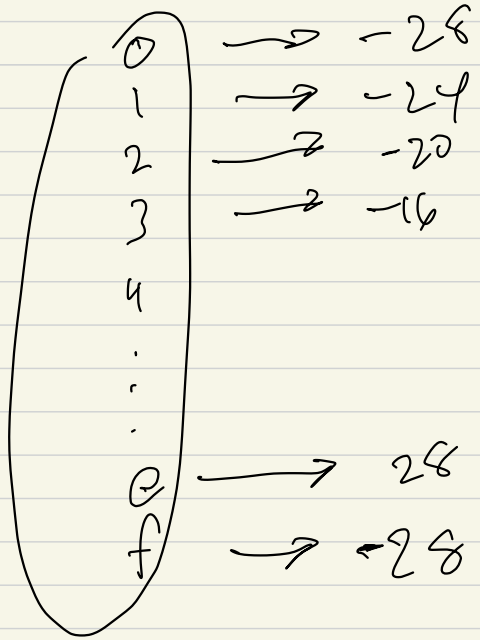
freq = & V_sync >= 74

0 10{state} & 10'232 |
1
2
3 10{state} & 10'232 - frame
4 +
5 40 +
6 424 -
7 10'232
7
7 end + each above
7
7

need either

- FF that stores the tasty pos

- FFs that store last state before death




```
random_height = lfsr == 4'h0 ? -28:
                lfsr == 4'h1 ? -24:
                      ⋮
                lfsr == 4'he ? 28:
                -28
```

center = 10'd 240 - 10'd 48

height1 = 164 + 1fsr

height2 = height1 + 96