Lab 7: Frog Frenzy

Gabriel Dimas
6 June 2022
Section C

Description

The purpose of this lab was to create a game using VGA wiring, timing, state machines, and point accuracy when displaying data to a screen. Although one of the hardest labs, it did in fact do its duty in teaching how data is displayed to a user and the arithmetic and inequalities that go into game design and hardware management.

Design

- **Top_Module_Main**
  - Inputs: btnC, btnU, btnD, btnR, btnL, [15:0] sw, clkin,
  - Outputs: Hsync, Vsync, [3:0] vgaRed, [3:0] vgaGreen, [3:0] vgaBlue, [15:0] led, [6:0] seg, dp, [3:0] an
  - Implementation: The top module holds the implementation of all of the other modules together. This holds instances of a frog, the VGA values, three trees, the counter (and segment displays), the frame counters, and all the delays for the state machine.

```
//3 pixels / frame clk
wire [3:0] count;
wire ThreePixFrameClk;
countUD4L faster_clk (.Up(count < 3), .Dw(1'b0), .LD(frame), .Q(4'b0), .clk(clk), .Reset(1'b0), .Qout(count));
assign ThreePixFrameClk = count < 3;
```

Figure 1

- **VSync_Tracker**
  - Inputs: clk, NextCol
  - Output: Frame, [11:0] Position
  - Implementation: This module uses four 4-bit counters to continuously count upon the clock signal until it is told to reset when the value of the entire counter is greater than 524. When this value is reached, the counter resets, and the output for the Frame would be high for one clock cycle, indicating the end of a frame. Here is a snippet of the code that ensures we have a frame value.

```
assign Reset = out > 524 ? 1'b1 : 1'b0;
Edge_Detector end_frame (.clk(clk), .btn(Reset), .out(Frame));  //end of a full frame when HIGH
assign Position = out;
```

Figure 2

- **HSync_Tracker**
  - Inputs: clk
  - Outputs: RowFinish, [11:0] Position
  - Implementation: This module uses four 4-bit counters to continuously count up on the clock signal until it is told to reset when the value of the entire counter is greater than 798. When this value is reached, the counter resets and the output for

the RowFinihs is high for one clock cycle to indicate that a row is finished and that the **VSynch_Tracker** should move onto the next column. Below is the snippet of code that makes the colors and different shapes appear on the screen to the user.

```
assign Red   = (Frog_Position <= currentVCount) & (currentVCount <= Frog_Position+Frog_WIDTH) & (Frog_Start <= currentHCount) & (currentHCount <= Frog_Start+Frog_WIDTH)    ? 4'hf : 4'h0;


assign Green = (Frog_Position <= currentVCount) & (currentVCount <= Frog_Position+Frog_WIDTH) & (Frog_Start <= currentHCount) & (currentHCount <= Frog_Start+Frog_WIDTH)                    ? 4'hf :
               (currentVCount-Tree_Height <= Tree1Row) & (Tree1Row <= currentVCount)   & (currentHCount <= Tree1Col)   & (Tree1Col <= currentHCount+Tree_Width) & (currentHCount < 639)      ? 4'hf :
               (currentVCount-Tree_Height <= Tree2Row) & (Tree2Row <= currentVCount)   & (currentHCount <= Tree2Col)   & (Tree2Col <= currentHCount+Tree_Width) & (currentHCount < 639)      ? 4'hf :
               (currentVCount-Tree_Height <= Tree3Row) & (Tree3Row <= currentVCount)   & (currentHCount <= Tree3Col)   & (Tree3Col <= currentHCount+Tree_Width) & (currentHCount < 639)      ? 4'hf : 4'h0;


assign Blue  = (Frog_Position <= currentVCount) & (currentVCount <= Frog_Position+Frog_WIDTH) & (Frog_Start <= currentHCount) & (currentHCount <= Frog_Start+Frog_WIDTH)   ? 4'hf :   //Frog
               (currentVCount-Tree_Height <= Tree1Row) & (Tree1Row <= currentVCount)   & (currentHCount <= Tree1Col)   & (Tree1Col <= currentHCount+Tree_Width)     ? 4'h0 :
               (currentVCount-Tree_Height <= Tree2Row) & (Tree2Row <= currentVCount)   & (currentHCount <= Tree2Col)   & (Tree2Col <= currentHCount+Tree_Width)     ? 4'h0 :
               (currentVCount-Tree_Height <= Tree3Row) & (Tree3Row <= currentVCount)   & (currentHCount <= Tree3Col)   & (Tree3Col <= currentHCount+Tree_Width)     ? 4'h0 :


          (240 <= currentVCount) & (currentVCount <= 256) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b1111 :
          (257 <= currentVCount) & (currentVCount <= 273) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b1100 :
          (274 <= currentVCount) & (currentVCount <= 290) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b1011 :
          (240 <= currentVCount) & (currentVCount <= 307) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b1010 :
          (308 <= currentVCount) & (currentVCount <= 324) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b1001 :
          (325 <= currentVCount) & (currentVCount <= 341) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b1000 :
          (240 <= currentVCount) & (currentVCount <= 358) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b0111 :
          (359 <= currentVCount) & (currentVCount <= 375) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b0110 :
          (376 <= currentVCount) & (currentVCount <= 392) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b0101 :
          (393 <= currentVCount) & (currentVCount <= 409) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b0100 :
          (410 <= currentVCount) & (currentVCount <= 426) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b0011 :
          (427 <= currentVCount) & (currentVCount <= 443) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b0010 :
          (444 <= currentVCount) & (currentVCount <= 460) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b0001 :
          (461 <= currentVCount) & (currentVCount <= 479) & (0 <= currentHCount) & (currentHCount <= 639)   ? 4'b0000 : 4'h0;
```

Figure 3

- **StateMachine**
  - Inputs: Up, Down, Center, SEC2, Moving, HIT, clk
  - Output: Rungame, INITSTATE, TimerStart2Sec, Frog_Up, Frog_Down, Frog_Blink, Reset
  - Implementation: This module consists of a six-state state machine. The states include an initial state, Delay for two-second states, a forward state, a frog move upstate, a frog moves downstate, and an end delays state. During the initial state, the frog is stationary and nothing is moving or blinking. During the delay for two seconds state, the fog and the seven segment displays are blinking two times for a total of two seconds. The forward state is a state where the trees move (not the frog) forward until the action happens. During the upstate, the frog is given a signal to move up for the entire motion. This state also includes the motion for the frog to move back down to the original state. During the downstate, the frog is given a signal to move down for the entire motion. This state also includes the motion for the frog to move back up to the original state. During the end delay state, the plants stop moving in their original position, the frog stops moving, and the frog and the seven segment display all blink in unison until the btnC button is pressed. When the btnC button is pressed, the frog resets to the middle position, the plants move back to their original positions, and the seven-segment displays reset to their zero values. Here is a snapshot of the code of all the states.

```
assign Next_INIT = INIT & !center;
FDRE #(.INIT(1'b1)) init     (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_INIT),      .Q(INIT));

assign Next_DELAY2SEC = (DELAY2SEC & !SEC2) | (INIT & center) | (END_DELAY & center);
FDRE #(.INIT(1'b0)) del2sec  (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_DELAY2SEC), .Q(DELAY2SEC));

assign Next_FORWARD = (FORWARD & (up ~^ down) & !HIT) | (DELAY2SEC & SEC2) | (UP & !Moving) | (DOWN & !Moving);
FDRE #(.INIT(1'b0)) forward  (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_FORWARD),   .Q(FORWARD));

assign Next_UP = (UP & Moving & !HIT) | (FORWARD & up & !down) | (UP & !Moving & up & !down);
FDRE #(.INIT(1'b0)) upff     (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_UP),        .Q(UP));

assign Next_DOWN = (DOWN & Moving & !HIT) | (FORWARD & !up & down) | (DOWN & !Moving & !up & down);
FDRE #(.INIT(1'b0)) downff   (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_DOWN),      .Q(DOWN));

assign Next_END_DELAY = (END_DELAY & !center) | (FORWARD & HIT) | (UP & HIT) | (DOWN & HIT);
FDRE #(.INIT(1'b0)) endDelay (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_END_DELAY), .Q(END_DELAY));
```

Figure 4

- **Blink**
  - Inputs: InputSignal, Framerate, clk
  - Output: OutputSignal
  - Implementation: This module uses two 4-bit counters that measure the number of times the counter counts upon the frame clock cycle. When the timer reaches 64 (about one second) the flip flop is triggered to go to the opposite value using the XOR logic. This '0' and '1' action repeats as long as the InputSignal is high. The output is the current flip flop value. Here is a snippet of the code that makes this action happen. **Figure 5** is the code that makes the seven-segment displays blink, while **Figure 6** is the code that does the actually blinking logic.

```
wire [3:0]Qring;
RingCounter ring_cntr (.digsel(digsel), .clk(clk), .Q(Qring));
assign an[0] = !(Qring[0]) | blink;
assign an[1] = !(Qring[1]) | blink;
assign an[2] = !(Qring[2]) | blink;
assign an[3] = !(Qring[3]) | blink;
assign dp = 1;
wire [3:0]sel;
Selector select(.sel(Qring), .N(bit16out), .H(sel));
hex7seg segment_disp (.n(sel), .seg(seg));
```

Figure 5 (Left) and Figure 6 (Below)

```
countUD4L blink1 (.Up(InputSignal),     .Dw(1'b0), .LD(1'b0), .Reset(TIME_UP),  .UTC(utc),  .Q(4'b0), .clk(Framerate), .Qout(timer[3:0]));
countUD4L blink2 (.Up(InputSignal&utc), .Dw(1'b0), .LD(1'b0), .Reset(TIME_UP),  .UTC(utc2), .Q(4'b0), .clk(Framerate), .Qout(timer[7:4]));

wire XOR_OUT = InputSignal & (TIME_UP ^ blink);
FDRE #(.INIT(1'b0) ) ffB2 (.C(Framerate), .R(1'b0), .CE(1'b1), .D(XOR_OUT), .Q(blink));

assign OutputSignal = blink;
```

- **Plants**
  - Inputs: Rungame, Framerate, INIT, [11:0] INITPosition, FirstPlant, ThreePixFrame,Clk, ResetPlant, clk
  - Output: [11:0] Row, [11:0] Col
  - Implementation: This module controls the position of the plants. When an instance of a plant is created, the inputs determine if this is the first plant in the sequence of plants. There should only be one of these values HIGH for any string

of **Plants** call. The two 4-bit counters count down from their INITPosition values (loaded at the delay state of the game) until the counters reach zero value. When this happens, the counters will then reset to the "12'b001011010101" value, which is used as the reset column value. The row value is randomly chosen using the LFSR values. This module chooses a 4-bit pseudorandom number and selects the index value of that number from a pool of parameters created in the **Plant** module. (see **Figure 7** below).

```
//All these numbers are WTR 232
parameter pos1 = 156, pos2 = 160, pos3 = 164, pos4 = 168, pos5 = 172, pos6 = 176, pos7 = 180, pos8 = 184;
parameter pos9 = 188, pos10 = 192, pos11 = 196, pos12 = 200, pos13 = 216, pos14 = 232, pos15 = 236;

//This randomly choses a row to start off on
wire [3:0]rand;
wire getRandRow = (Col == 680);
LFSR randNumGen (.clk(clk), .GetNum(getRandRow), .Q(rand));

wire initHold;
FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(Reset), .CE(INIT|Reset), .D(INIT), .Q(initHold)); //if we started with an initial position

assign Row = FirstPlant & initHold ? pos8:
                  (rand == 0) ? pos1  :
                  (rand == 1) ? pos2  :
                  (rand == 2) ? pos3  :
                  (rand == 3) ? pos4  :
                  (rand == 4) ? pos5  :
                  (rand == 5) ? pos6  :
                  (rand == 6) ? pos7  :
                  (rand == 7) ? pos8  :
                  (rand == 8) ? pos9  :
                  (rand == 9) ? pos10 :
                  (rand == 10)? pos11 :
                  (rand == 11)? pos12 :
                  (rand == 12)? pos13 :
                  (rand == 13)? pos14 :
                  (rand == 14)? pos15 : pos8;   //to further randomize
```

Figure 7

- **Frog**
  - Inputs: MoveUp, MoveDown, In_End_Delay, INIT, Reset_btnC, Framerate, clk, ThreePixPerFrame, FrogBlink, Rungame
  - Output: Moving, [11:0] Position
  - Implementation: This module controls the position of the frog. The MoveUp and MoveDown inputs are implemented in a way that the entire movement of the frog happens in one continuous motion. For example, when the MoveUp signal is high (and stays high), the Moving output is high (signal for the state machine that the frog is not in the resting state and is out of idle position) and the counters count down towards zero. When the counters reach a value of less than 120, the godown flip flop goes high and the counter begins to count back up towards the frog's idle value position. When the frog reaches its idle position, the counters stop counting, the initial position is loaded into the frog to ensure no movement, and the Moving output goes LOW. This signals to the state machine that the frog has returned to its original position and can continue with the forward state. Without loss of generality, this process is the same when the frog moves in the MoveDown state. Here is a snippet of the code used to make the frog move up and down

```
FDRE #(.INIT(1'b0)) downff (.C(Framerate), .R(godown & Position >= CENTER),     .CE(5 < Position & Position <= 120), .D(1'b1), .Q(godown));
FDRE #(.INIT(1'b0)) upff   (.C(Framerate), .R(goup   & Position <= CENTER),     .CE(Position >= 328),               .D(1'b1), .Q(goup));

wire Up_Direction   = Rungame & ((MoveUp & !godown) | (MoveDown & goup));
wire Down_Direction = Rungame & ((MoveUp & godown)  | (MoveDown & !goup));

countUD4L trackerfrg1 (.Up(ThreePixPerFrame&Down_Direction),              .Dw(ThreePixPerFrame&Up_Direction),              .LD(INIT | (Rungame & noMotion) | (Reset_btnC & In_End_Delay)),
countUD4L trackerfrg2 (.Up(ThreePixPerFrame&Down_Direction & utc1),       .Dw(ThreePixPerFrame&Up_Direction & dtc1),       .LD(INIT | (Rungame & noMotion) | (Reset_btnC & In_End_Delay)),
countUD4L trackerfrg3 (.Up(ThreePixPerFrame&Down_Direction & utc1 & utc2), .Dw(ThreePixPerFrame&Up_Direction & dtc1 & dtc2), .LD(INIT | (Rungame & noMotion) | (Reset_btnC & In_End_Delay)),
```

Figure 8

- **Delay2sec**
  - Inputs: Start, FrameClk, clk
  - Output: Signal
  - Implementation: This module is a daily simple implementation of a 2-second delay. This uses three 4-bit counters to count up to a safe value of greater than 250 and then resets to zero when the value is greater than 250. This number is used because 64 frames are about ½ a second. So 64 frames times 4 half seconds = 250 on the counter value. See the snippet below of how this was implemented.

```
countUD4L tracker1 (.Up(Start),           .Dw(1'b0), .LD(1'b0), .Reset(Signal), .Q(4'b0), .clk(FrameClk), .UTC(utc1), .Qout(out[3:0]));
countUD4L tracker2 (.Up(Start&utc1),      .Dw(1'b0), .LD(1'b0), .Reset(Signal), .Q(4'b0), .clk(FrameClk), .UTC(utc2), .Qout(out[7:4]));
countUD4L tracker3 (.Up(Start&utc1&utc2), .Dw(1'b0), .LD(1'b0), .Reset(Signal), .Q(4'b0), .clk(FrameClk),             .Qout(out[11:8]));
```

Figure 9

- **Ring_Counter**
  - Inputs: digsel, clk
  - Outputs: [3:0]out
  - Implementation: See **Design:** Ring_Counter *Lab 4: Multiplexers, Full Adders, and Seven Segment Displays*
- **Selector**
  - Inputs: [3:0]in
  - Outputs: [3:0]out
  - Implementation: See **Design:** Selector *Lab 4: Multiplexers, Full Adders, and Seven Segment Displays*
  -
- **hex7seg**
  - Inputs: [3:0]in
  - Outputs: [6:0]out
  - Implementation: See **Design:** hex7Seg *Lab 4: Multiplexers, Full Adders, and Seven Segment Displays*

Testing & Simulation

During the beginning of the lab, all portions of the lab were able to be tested. As soon as the VGA and the display-only states came into play, it was very difficult to try and test the display as a simulation. For the parts that *can* be tested, there needed to be exact states for every part of the state. Certain inputs were chosen for reasons that every possible state needed to be tested and considered. For example, the state machine in this lab was very similar to that of Lab

5 because the current input of the state machine depended on its own outputs that needed a feedback signal. It was also important to note that some modules used both a frame clock and a system clock. This was later relinquished when using which clocks needed to go where and how they were going to be used (i.e which clock should a counter or flip flop be in sync with: the frame or the system clock?). This is very important when developing because one clock is significantly slower than the other.

Results

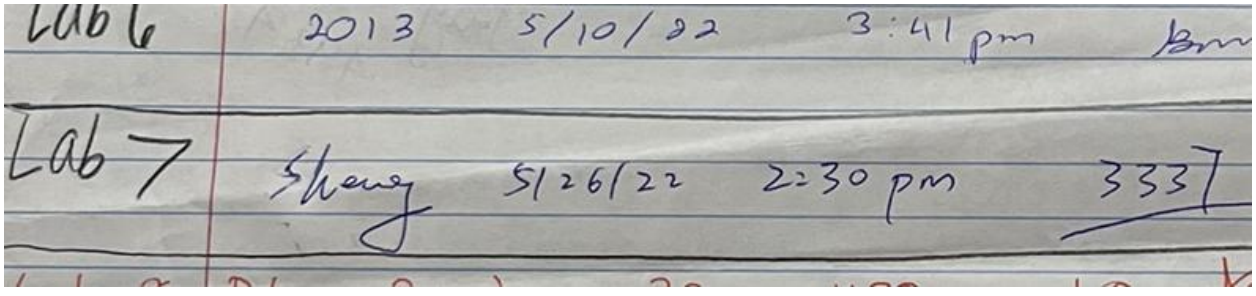
The final result of this lab was a success! The final and initial check off was one of the most important parts that was able to be done successfully. It is important to note that the state machine hardly gave any types of issues. It was mainly the coloring (VGA outputs) that was the issue. For example, the frog and the plants had moments when the green color would disappear. This issue was trying to be resolved for nearly six hours, and would finally be solved. This issue was the result of enabling color where color shouldn't be enabled off the screen. Because there are certain Hsync and Vsync areas where the RGB values should be LOW, but there are also places on-screen where those same RGB values should be HIGH. This resulted in the RGB values trying to be HIGH and LOW simultaneously, which resulted in a brief discoloration of the frog and the plants.

As this was solved, another issue came up with the randomness of the plant row. This was quickly resolved by enabling the flip flops in the LFSR module, as they weren't done beforehand. As a result, the flip flops were not active and the LFSR value was the same throughout the program run.

Conclusion

This lab, although high in its difficulty, created many new paths to learning. This also brought to light the amazing aspects of VGA and real world applications to a VGA-enabled screen.

Appendix



LAB 7 Signed Off: 5/26/22 2:30PM. CODE: 3337

Below are the Lab 5 Schematics and Verilog Code.

```
module Top_Module_Main(
    input btnC,
    input btnU,
    input btnD,
    input btnR,
    input btnL,
    input [15:0] sw,
    input clkin,
    output Hsync,
    output Vsync,
    output [3:0] vgaRed,
    output [3:0] vgaGreen,
    output [3:0] vgaBlue,
    output [15:0] led,
    output [6:0] seg,
    output dp,
    output [3:0]an
    );
lab7_clks not_so_slow (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel));

wire [11:0] H_Count_Value;
wire [11:0] V_Count_Value, frog_position, treecol1, treecol2, treecol3, treerow1,
treerow2, treerow3;
wire [15:0] bit16out;
wire rowFinish, EDrowFinish, frame;
wire moveup, movedown, moving, start_timer, frog_blink, delay_signal;
//VGA CONNECTIONS
//need a module to count where we are horizontally (pass in H_Count_Value)
Edge_Detector finishRow (.clk(clk), .btn(rowFinish), .out(EDrowFinish));
HSync_Tracker hsynch_track (.clk(clk), .RowFinish(rowFinish),
.Position(H_Count_Value));
//need a module to count where we are vertically   (pass in H_Count_Value)
VSync_Tracker vsynch_track (.clk(clk), .NextCol(EDrowFinish), .Frame(frame),
.Position(V_Count_Value));

//3 pixels / frame clk
wire [3:0] count;
wire ThreePixFrameClk;
countUD4L faster_clk (.Up(count < 3), .Dw(1'b0), .LD(frame), .Q(4'b0), .clk(clk),
.Reset(1'b0), .Qout(count));
assign ThreePixFrameClk = count < 3;

// VGA sync outputs
assign Hsync = (H_Count_Value >= 656 & H_Count_Value <= 751) ? 1'b0 : 1'b1;
assign Vsync = (V_Count_Value >= 489 & V_Count_Value <= 490) ? 1'b0 : 1'b1;
```

```
//LOGIC
//add state machine
wire edge_L, rungame, blink, reset_game, initial_state, plant_hit;
StateMachine statemachine (.Up(btnU), .Down(btnD), .Center(btnC),
.SEC2(delay_signal), .Moving(moving),
                           .HIT(plant_hit&!sw[0] | sw[2]), .clk(clk),
.TimerStart2Sec(start_timer), .Frog_Up(moveup), .INITSTATE(initial_state),
//switches used to simulate HIT
                           .Frog_Down(movedown), .Rungame(rungame),
.Frog_Blink(frog_blink), .Reset(reset_game));

//Logic for making the segments and the frog blink
Blink blinker (.InputSignal(frog_blink), .Framerate(frame), .clk(clk),
.OutputSignal(blink));

//Logic for the switches and LEDs
assign led = sw;
//add instances of 3 trees
wire resetPlantPosition;        //while in the END_DELAY state, btnC is pressed and
we can start over
Edge_Detector rstPlants (.clk(clk), .btn(btnC&!rungame), .out(resetPlantPosition));
wire goAheadPlant2, goAheadPlant3;
Plants plant1 (.Rungame(!sw[3]&rungame), .INIT(initial_state | start_timer),
.clk(clk), .FirstPlant(1'b1), .Row(treerow1), .Col(treecol1), .INITPosition(12'd300),
               .ResetPlant(resetPlantPosition), .Framerate(frame),
.ThreePixFrameClk(ThreePixFrameClk));

//FDRE #(.INIT(1'b0)) readyplant2 (.C(clk), .R(resetPlantPosition), .CE(treecol1 ==
470), .D(1'b1), .Q(goAheadPlant2));  //when plant1 is in the middle of the screen
Plants plant2 (.Rungame(!sw[3]&rungame), .INIT(initial_state | start_timer),
.clk(clk), .FirstPlant(1'b0), .Row(treerow2), .Col(treecol2), .INITPosition(12'd530),
               .ResetPlant(resetPlantPosition), .Framerate(frame),
.ThreePixFrameClk(ThreePixFrameClk));

//FDRE #(.INIT(1'b0)) readyplant3 (.C(clk), .R(resetPlantPosition), .CE(treecol1 ==
267), .D(1'b1), .Q(goAheadPlant3)); //when plant1 is in the first third of the screen
Plants plant3 (.Rungame(!sw[3]&rungame), .INIT( | start_timer), .clk(clk),
.FirstPlant(1'b0), .Row(treerow3), .Col(treecol3), .INITPosition(12'd745),
               .ResetPlant(resetPlantPosition), .Framerate(frame),
.ThreePixFrameClk(ThreePixFrameClk));

//add 2 sec timer
Delay2sec timer (.Start(start_timer), .FrameClk(frame), .clk(clk),
.Signal(delay_signal));

//add frog
```

```
Frog frog (.MoveUp(moveup), .MoveDown(movedown), .Reset_btnC(btnC),
.Framerate(frame), .In_End_Delay(frog_blink), .clk(clk),
          .ThreePixPerFrame(ThreePixFrameClk), .INIT(initial_state),
.Moving(moving), .Rungame(rungame), .FrogBlink(blink), .Position(frog_position));
//count number of times frog passed a plant
wire utc1, utc2, utc3, increment;
wire [15:0]switch;
countUD4L trackpoint1 (.Up(rungame&increment),                .Dw(1'b0),
.LD(btnC&!rungame), .Reset(1'b0), .Q(4'b0), .clk(frame), .UTC(utc1),
.Qout(switch[3:0]));
countUD4L trackpoint2 (.Up(rungame&increment&utc1),           .Dw(1'b0),
.LD(btnC&!rungame), .Reset(1'b0), .Q(4'b0), .clk(frame), .UTC(utc2),
.Qout(switch[7:4]));
countUD4L trackpoint3 (.Up(rungame&increment&utc1&utc2),      .Dw(1'b0),
.LD(btnC&!rungame), .Reset(1'b0), .Q(4'b0), .clk(frame), .UTC(utc3),
.Qout(switch[11:8]));
countUD4L trackpoint4 (.Up(rungame&increment&utc1&utc2&utc3), .Dw(1'b0),
.LD(btnC&!rungame), .Reset(1'b0), .Q(4'b0), .clk(frame),
.Qout(switch[15:12]));
assign bit16out[3] = switch[0], bit16out[2] = switch[1], bit16out[1] = switch[2],
bit16out[0] = switch[3];
assign bit16out[7] = switch[4], bit16out[6] = switch[5], bit16out[5] = switch[6],
bit16out[4] = switch[7];
assign bit16out[11] = switch[8], bit16out[10] = switch[9], bit16out[9] = switch[10],
bit16out[8] = switch[11];
assign bit16out[15] = switch[12], bit16out[14] = switch[13], bit16out[13] =
switch[14], bit16out[12] = switch[15];

//Logic for the VGA colors
VGA_Control vga (.Frog_Position(frog_position), .Tree1Row(treerow1),
.Tree2Row(treerow2), .Tree3Row(treerow3), .Tree1Col(treecol1), .Tree2Col(treecol2),
.AddOne(increment),
                .Tree3Col(treecol3), .currentHCount(H_Count_Value),
.currentVCount(V_Count_Value), .Red(vgaRed), .Green(vgaGreen), .Blue(vgaBlue),
.HIT(plant_hit));
wire [3:0]Qring;
RingCounter ring_cntr (.digsel(digsel), .clk(clk), .Q(Qring));
assign an[0] = !(Qring[0]) | blink;
assign an[1] = !(Qring[1]) | blink;
assign an[2] = !(Qring[2]) | blink;
assign an[3] = !(Qring[3]) | blink;
assign dp = 1;
wire [3:0]sel;
Selector select(.sel(Qring), .N(bit16out), .H(sel));
hex7seg segment_disp (.n(sel), .seg(seg));
endmodule
```

```
module HSync_Tracker(
    input clk,
    output RowFinish,
    output [11:0] Position
    );
    wire Reset;
    wire utc1, utc2, utc3;
    wire [11:0]out;
   //the !clk simulated the negative clock edge
    countUD4L tracker1 (.Up(!clk),           .Dw(1'b0), .LD(1'b0), .Reset(Reset),
.Q(4'b0), .clk(clk), .UTC(utc1), .Qout(out[3:0] ));
    countUD4L tracker2 (.Up(!clk&utc1),      .Dw(1'b0), .LD(1'b0), .Reset(Reset),
.Q(4'b0), .clk(clk), .UTC(utc2), .Qout(out[7:4] ));
    countUD4L tracker3 (.Up(!clk&utc1&utc2), .Dw(1'b0), .LD(1'b0), .Reset(Reset),
.Q(4'b0), .clk(clk), .UTC(utc3), .Qout(out[11:8]));

    assign Reset = out > 798 ? 1'b1 : 1'b0;
    assign RowFinish = out >= 798;
    assign Position = out;

endmodule
```

```
module VSync_Tracker(
    input clk,
    input NextCol,
    output Frame,
    output [11:0] Position
    );
    wire Reset;
    wire utc1, utc2, utc3;
    wire [11:0]out;
    countUD4L tracker1 (.Up(!clk&NextCol),           .Dw(1'b0), .LD(1'b0),
.Reset(Reset), .Q(4'b0), .clk(clk), .UTC(utc1), .Qout(out[3:0] ));
    countUD4L tracker2 (.Up(!clk&NextCol&utc1),      .Dw(1'b0), .LD(1'b0),
.Reset(Reset), .Q(4'b0), .clk(clk), .UTC(utc2), .Qout(out[7:4] ));
    countUD4L tracker3 (.Up(!clk&NextCol&utc1&utc2), .Dw(1'b0), .LD(1'b0),
.Reset(Reset), .Q(4'b0), .clk(clk), .UTC(utc3), .Qout(out[11:8]));

    assign Reset = out > 524 ? 1'b1 : 1'b0;
    Edge_Detector end_frame (.clk(clk), .btn(Reset), .out(Frame));  //end of a full
frame when HIGH
    assign Position = out;


endmodule
```

```
module StateMachine(
    input Up,
    input Down,
    input Center,
    input SEC2,
    input Moving,
    input HIT,
    input clk,
    output Rungame,
    output INITSTATE,
    output TimerStart2Sec,
    output Frog_Up,
    output Frog_Down,
    output Frog_Blink,
    output Reset
    );
    wire up, down, center;
    wire INIT, DELAY2SEC, FORWARD, UP, DOWN, END_DELAY;
    wire Next_INIT, Next_DELAY2SEC, Next_FORWARD, Next_UP, Next_DOWN, Next_END_DELAY;
    FDRE #(.INIT(1'b0)) sync1 (.C(clk), .R(1'b0), .CE(1'b1), .D(Up), .Q(up));
    FDRE #(.INIT(1'b0)) sync2 (.C(clk), .R(1'b0), .CE(1'b1), .D(Down), .Q(down));
    FDRE #(.INIT(1'b0)) sync3 (.C(clk), .R(1'b0), .CE(1'b1), .D(Center), .Q(center));


    assign Next_INIT = INIT & !center;
    FDRE #(.INIT(1'b1)) init     (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_INIT),
.Q(INIT));

    assign Next_DELAY2SEC = (DELAY2SEC & !SEC2) | (INIT & center) | (END_DELAY &
center);
    FDRE #(.INIT(1'b0)) del2sec  (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_DELAY2SEC),
.Q(DELAY2SEC));

    assign Next_FORWARD = (FORWARD & (up ~^ down) & !HIT) | (DELAY2SEC & SEC2) | (UP
& !Moving) | (DOWN & !Moving);
    FDRE #(.INIT(1'b0)) forward  (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_FORWARD),
.Q(FORWARD));

    assign Next_UP = (UP & Moving & !HIT) | (FORWARD & up & !down) | (UP & !Moving &
up & !down);
    FDRE #(.INIT(1'b0)) upff     (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_UP),
.Q(UP));

    assign Next_DOWN = (DOWN & Moving & !HIT) | (FORWARD & !up & down) | (DOWN &
!Moving & !up & down);
    FDRE #(.INIT(1'b0)) downff   (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_DOWN),
```

```
.Q(DOWN));

    assign Next_END_DELAY = (END_DELAY & !center) | (FORWARD & HIT) | (UP & HIT) |
(DOWN & HIT);
    FDRE #(.INIT(1'b0)) endDelay (.C(clk), .R(1'b0), .CE(1'b1), .D(Next_END_DELAY),
.Q(END_DELAY));

    assign INITSTATE = INIT;
    assign TimerStart2Sec = DELAY2SEC;
    assign Frog_Up = UP;
    assign Frog_Down = DOWN;
    assign Frog_Blink = DELAY2SEC | END_DELAY;
    assign Rungame = !(INIT | DELAY2SEC | END_DELAY);
    assign Reset = (END_DELAY & center);

endmodule
```

```
module Blink(
    input InputSignal,
    input Framerate,
    input clk,
    output OutputSignal
    );
wire [7:0]timer;
wire utc,blink, utc2;
wire TIME_UP = timer == 64;
countUD4L blink1 (.Up(InputSignal),     .Dw(1'b0), .LD(1'b0), .Reset(TIME_UP),
.UTC(utc),  .Q(4'b0), .clk(Framerate), .Qout(timer[3:0]));
countUD4L blink2 (.Up(InputSignal&utc), .Dw(1'b0), .LD(1'b0), .Reset(TIME_UP),
.UTC(utc2), .Q(4'b0), .clk(Framerate), .Qout(timer[7:4]));

wire XOR_OUT = InputSignal & (TIME_UP ^ blink);
FDRE #(.INIT(1'b0) ) ffB2 (.C(Framerate), .R(1'b0), .CE(1'b1), .D(XOR_OUT), .Q(blink)

assign OutputSignal = blink;
endmodule
```

```
module Plants(
   input Rungame,
   input Framerate,
   input INIT,
   input [11:0] INITPosition,
   input FirstPlant,
   input ThreePixFrameClk,
   input ResetPlant,
   input clk,
   output [11:0] Row,
   output [11:0] Col
   );
   wire dtc1, dtc2;

   wire Reset = (Col == 4079); //when to reset the tree to the other side
   wire [11:0] PlantPosition = INIT ? INITPosition : 12'b001011010101;
    countUD4L plantcount1 (.Up(1'b0), .Dw(ThreePixFrameClk & Rungame),
.LD(INIT | Reset | ResetPlant), .Reset(1'b0), .Q(PlantPosition[3:0]), .clk(clk),
.DTC(dtc1), .Qout(Col[3:0]));  //counters for the columns of the plants
    countUD4L plantcount2 (.Up(1'b0), .Dw(ThreePixFrameClk & Rungame & dtc1),
.LD(INIT | Reset | ResetPlant), .Reset(1'b0), .Q(PlantPosition[7:4]), .clk(clk),
.DTC(dtc2), .Qout(Col[7:4]));
    countUD4L plantcount3 (.Up(1'b0), .Dw(ThreePixFrameClk & Rungame & dtc1 & dtc2),
.LD(INIT | Reset | ResetPlant), .Reset(1'b0), .Q(PlantPosition[11:8]), .clk(clk),
         .Qout(Col[11:8]));   //the Q bits reset the plants to 680

    //All these numbers are WTR 232
    parameter pos1 = 156, pos2 = 160, pos3 = 164, pos4 = 168, pos5 = 172, pos6 =
176, pos7 = 180, pos8 = 184;
    parameter pos9 = 188, pos10 = 192, pos11 = 196, pos12 = 200, pos13 = 216, pos14
= 232, pos15 = 236;

    //This randomly choses a row to start off on
    wire [3:0]rand;
    wire getRandRow = (Col == 680);
    LFSR randNumGen (.clk(clk), .GetNum(getRandRow), .Q(rand));

    wire initHold;
    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(Reset), .CE(INIT|Reset), .D(INIT),
.Q(initHold)); //if we started with an initial position (only the first plant) it
should hold until the next value

    assign Row = FirstPlant & initHold ? pos8:
                        (rand == 0) ? pos1  :
                        (rand == 1) ? pos2  :
                        (rand == 2) ? pos3  :
```

```
                        (rand == 3) ? pos4  :
                        (rand == 4) ? pos5  :
                        (rand == 5) ? pos6  :
                        (rand == 6) ? pos7  :
                        (rand == 7) ? pos8  :
                        (rand == 8) ? pos9  :
                        (rand == 9) ? pos10 :
                        (rand == 10)? pos11 :
                        (rand == 11)? pos12 :
                        (rand == 12)? pos13 :
                        (rand == 13)? pos14 :
                        (rand == 14)? pos15 : pos8;   //to further randomize

endmodule
```

```
module Delay2sec(
    input Start,
    input FrameClk,
    input clk,
    output Signal
    );
      wire [11:0]out;
      wire utc1, utc2;
     countUD4L tracker1 (.Up(Start),           .Dw(1'b0), .LD(1'b0), .Reset(Signal),
.Q(4'b0), .clk(FrameClk), .UTC(utc1), .Qout(out[3:0]));
     countUD4L tracker2 (.Up(Start&utc1),      .Dw(1'b0), .LD(1'b0), .Reset(Signal),
.Q(4'b0), .clk(FrameClk), .UTC(utc2), .Qout(out[7:4]));
     countUD4L tracker3 (.Up(Start&utc1&utc2), .Dw(1'b0), .LD(1'b0), .Reset(Signal),
.Q(4'b0), .clk(FrameClk),             .Qout(out[11:8]));

     assign Signal = out > 250; //gives a small margin for blinking two complete time

endmodule
```

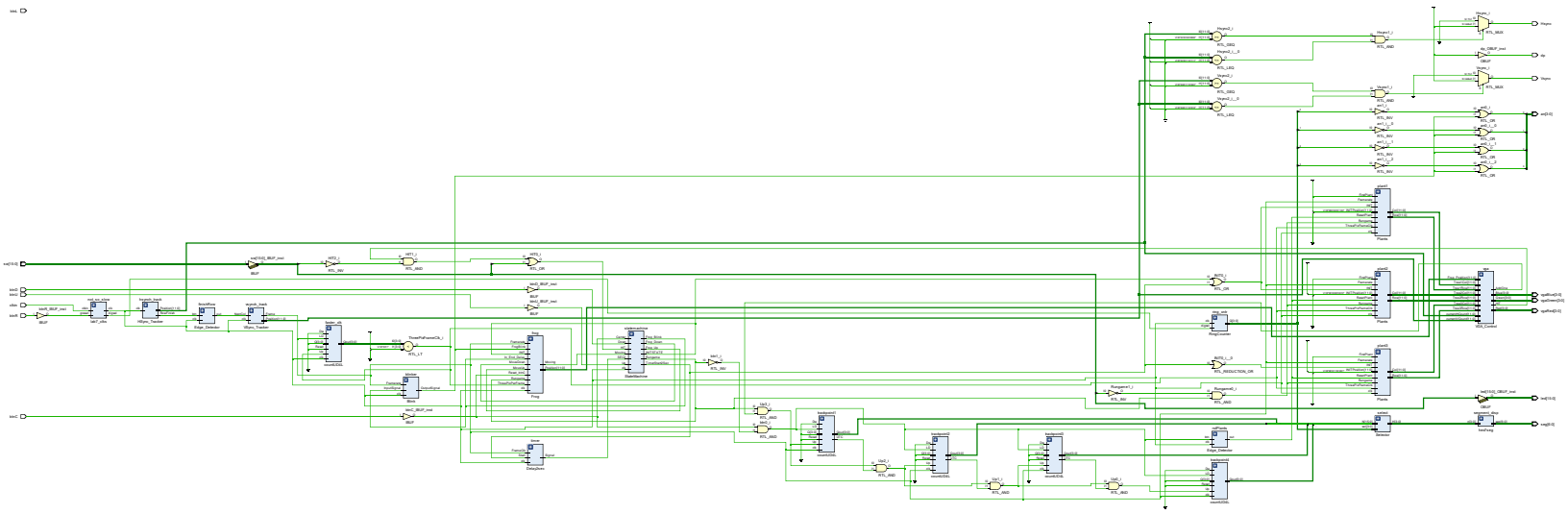
```
module Frog(
    input MoveUp,
    input MoveDown,
    input In_End_Delay,
    input INIT,
    input Reset_btnC,
    input Framerate,
    input clk,
    input ThreePixPerFrame,
    input FrogBlink,
    input Rungame,
    output Moving,
    output [11:0]Position
    );
    wire utc1, utc2, dtc1, dtc2, goup, godown, spikeMove;
    wire [11:0] frog_position;
    wire noMotion = !(MoveUp | MoveDown);
    parameter CENTER = 232;
    parameter OFFSCREEN_BLINK = 12'd480;

    FDRE #(.INIT(1'b0)) downff (.C(Framerate), .R(godown & Position >= CENTER),
.CE(5 < Position & Position <= 120), .D(1'b1), .Q(godown));
    FDRE #(.INIT(1'b0)) upff   (.C(Framerate), .R(goup   & Position <= CENTER),
.CE(Position >= 328),               .D(1'b1), .Q(goup));

    wire Up_Direction   = Rungame & ((MoveUp & !godown) | (MoveDown & goup));
    wire Down_Direction = Rungame & ((MoveUp & godown)  | (MoveDown & !goup));

    countUD4L trackerfrg1 (.Up(ThreePixPerFrame&Down_Direction),
.Dw(ThreePixPerFrame&Up_Direction),              .LD(INIT | (Rungame & noMotion) |
(Reset_btnC & In_End_Delay)), .Reset(1'b0), .Q(4'b1000), .clk(clk), .UTC(utc1),
.DTC(dtc1), .Qout(frog_position[3:0]));
    countUD4L trackerfrg2 (.Up(ThreePixPerFrame&Down_Direction & utc1),
.Dw(ThreePixPerFrame&Up_Direction & dtc1),       .LD(INIT | (Rungame & noMotion) |
(Reset_btnC & In_End_Delay)), .Reset(1'b0), .Q(4'b1110), .clk(clk), .UTC(utc2),
.DTC(dtc2), .Qout(frog_position[7:4]));
    countUD4L trackerfrg3 (.Up(ThreePixPerFrame&Down_Direction & utc1 & utc2),
.Dw(ThreePixPerFrame&Up_Direction & dtc1 & dtc2), .LD(INIT | (Rungame & noMotion) |
(Reset_btnC & In_End_Delay)), .Reset(1'b0), .Q(4'b0000), .clk(clk),
      .Qout(frog_position[11:8]));

    assign Position = FrogBlink ? OFFSCREEN_BLINK : frog_position;
    assign Moving = Position != 232 ;
endmodule
```

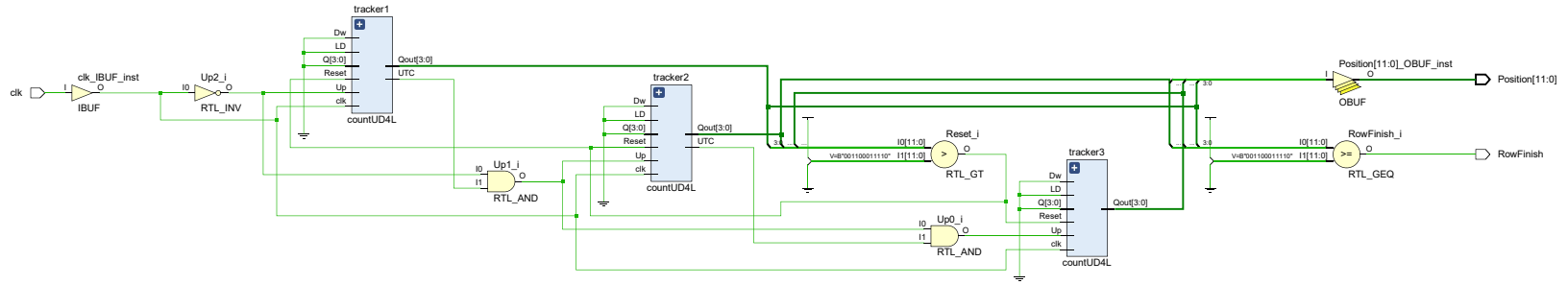
```
module VGA_Control (
    input [11:0] Frog_Position,
    input [11:0] Tree1Row,
    input [11:0] Tree2Row,
    input [11:0] Tree3Row,
    input [11:0] Tree1Col,
    input [11:0] Tree2Col,
    input [11:0] Tree3Col,
    input [11:0]currentHCount,
    input [11:0]currentVCount,
    output [3:0] Red,
    output [3:0] Green,
    output [3:0] Blue,
    output AddOne,
    output HIT
    );

parameter Frog_WIDTH  = 16;  // 120 -> 136
parameter Tree_Width  = 40;  // X -> X + 40
parameter Tree_Height = 96;  // Y -> Y + 96
parameter Frog_Start = 120;
assign Red   = (Frog_Position <= currentVCount) & (currentVCount <=
Frog_Position+Frog_WIDTH) & (Frog_Start <= currentHCount) & (currentHCount <=
Frog_Start+Frog_WIDTH)    ? 4'hf : 4'h0;


assign Green = (Frog_Position <= currentVCount) & (currentVCount <=
Frog_Position+Frog_WIDTH) & (Frog_Start <= currentHCount) & (currentHCount <=
Frog_Start+Frog_WIDTH)                      ? 4'hf :
               (currentVCount-Tree_Height <= Tree1Row) & (Tree1Row <=
currentVCount)   & (currentHCount <= Tree1Col)   & (Tree1Col <=
currentHCount+Tree_Width) & (currentHCount < 639)      ? 4'hf :
               (currentVCount-Tree_Height <= Tree2Row) & (Tree2Row <=
currentVCount)   & (currentHCount <= Tree2Col)   & (Tree2Col <=
currentHCount+Tree_Width) & (currentHCount < 639)      ? 4'hf :
               (currentVCount-Tree_Height <= Tree3Row) & (Tree3Row <=
currentVCount)   & (currentHCount <= Tree3Col)   & (Tree3Col <=
currentHCount+Tree_Width) & (currentHCount < 639)      ? 4'hf : 4'h0;


assign Blue  = (Frog_Position <= currentVCount) & (currentVCount <=
Frog_Position+Frog_WIDTH) & (Frog_Start <= currentHCount) & (currentHCount <=
Frog_Start+Frog_WIDTH)   ? 4'hf :   //Frog
                     (currentVCount-Tree_Height <= Tree1Row) & (Tree1Row <=
currentVCount)   & (currentHCount <= Tree1Col)   & (Tree1Col <=
currentHCount+Tree_Width)     ? 4'h0 :
```

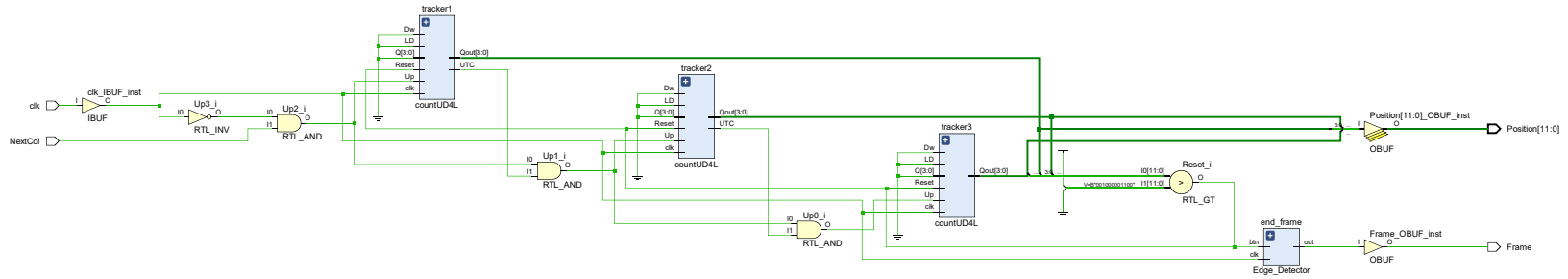
```
                    (currentVCount-Tree_Height <= Tree2Row) & (Tree2Row <=
currentVCount)   & (currentHCount <= Tree2Col)   & (Tree2Col <=
currentHCount+Tree_Width)     ? 4'h0 :
                    (currentVCount-Tree_Height <= Tree3Row) & (Tree3Row <=
currentVCount)   & (currentHCount <= Tree3Col)   & (Tree3Col <=
currentHCount+Tree_Width)     ? 4'h0 :


                  (240 <= currentVCount) & (currentVCount <= 256) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b1111 :
                  (257 <= currentVCount) & (currentVCount <= 273) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b1100 :
                  (274 <= currentVCount) & (currentVCount <= 290) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b1011 :
                  (240 <= currentVCount) & (currentVCount <= 307) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b1010 :
                  (308 <= currentVCount) & (currentVCount <= 324) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b1001 :
                  (325 <= currentVCount) & (currentVCount <= 341) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b1000 :
                  (240 <= currentVCount) & (currentVCount <= 358) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b0111 :
                  (359 <= currentVCount) & (currentVCount <= 375) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b0110 :
                  (376 <= currentVCount) & (currentVCount <= 392) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b0101 :
                  (393 <= currentVCount) & (currentVCount <= 409) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b0100 :
                  (410 <= currentVCount) & (currentVCount <= 426) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b0011 :
                  (427 <= currentVCount) & (currentVCount <= 443) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b0010 :
                  (444 <= currentVCount) & (currentVCount <= 460) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b0001 :
                  (461 <= currentVCount) & (currentVCount <= 479) & (0 <=
currentHCount) & (currentHCount <= 639)   ? 4'b0000 : 4'h0;


 wire [11:0] Frog_top_leftRow = Frog_Position;
 wire [11:0] Frog_top_leftCol = Frog_Start;

 wire [11:0] Frog_top_rightRow = Frog_Position;
 wire [11:0] Frog_top_rightCol = Frog_Start + Frog_WIDTH; //fixed point

 wire [11:0] Frog_bottom_rightRow = Frog_Position + Frog_WIDTH;
 wire [11:0] Frog_bottom_rightCol = Frog_Start + Frog_WIDTH;      //fixed point
```

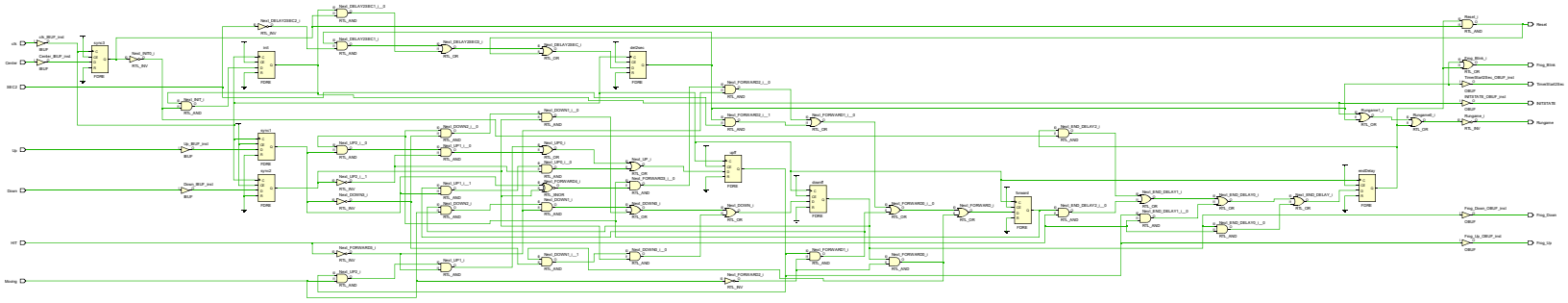
```
 wire [11:0] Frog_bottom_leftRow = Frog_Position + Frog_WIDTH;
 wire [11:0] Frog_bottom_leftCol = Frog_Start;          //fixed point


 assign HIT = (Tree1Row <= Frog_top_rightRow)     & (Frog_top_rightRow <= Tree1Row +
Tree_Height)      & (Tree1Col-Tree_Width <= Frog_top_rightCol)    &
(Frog_top_rightCol <= Tree1Col) |  //head on with top right corner hit
              (Tree1Row <= Frog_bottom_rightRow)  & (Frog_bottom_rightRow <=
Tree1Row + Tree_Height)   & (Tree1Col-Tree_Width <= Frog_bottom_rightCol) &
(Frog_bottom_rightCol <= Tree1Col)  |   //hit head on with the bottom left corner

              //Plant 2
              (Tree2Row <= Frog_top_rightRow)     & (Frog_top_rightRow <= Tree2Row +
Tree_Height)      & (Tree2Col-Tree_Width <= Frog_top_rightCol)    &
(Frog_top_rightCol <= Tree2Col) |  //head on with top right corner hit
              (Tree2Row <= Frog_bottom_rightRow)  & (Frog_bottom_rightRow <=
Tree2Row + Tree_Height)   & (Tree2Col-Tree_Width <= Frog_bottom_rightCol) &
(Frog_bottom_rightCol <= Tree2Col)  |   //hit head on with the bottom left corner

              //Plant 3
              (Tree3Row <= Frog_top_rightRow)     & (Frog_top_rightRow <= Tree3Row +
Tree_Height)      & (Tree3Col-Tree_Width <= Frog_top_rightCol)    &
(Frog_top_rightCol <= Tree3Col) |  //head on with top right corner hit
              (Tree3Row <= Frog_bottom_rightRow)  & (Frog_bottom_rightRow <=
Tree3Row + Tree_Height)   & (Tree3Col-Tree_Width <= Frog_bottom_rightCol) &
(Frog_bottom_rightCol <= Tree3Col)  |   //hit head on with the bottom left corner




              (Tree1Row <= Frog_top_leftRow)     & (Frog_top_leftRow <= Tree1Row +
Tree_Height)      & (Tree1Col-Tree_Width <= Frog_top_leftCol)    & (Frog_top_leftCol
<= Tree1Col) |  //head on with top right corner hit
              (Tree1Row <= Frog_bottom_leftRow)  & (Frog_bottom_leftRow <= Tree1Row
+ Tree_Height)   & (Tree1Col-Tree_Width <= Frog_bottom_leftCol) &
(Frog_bottom_leftCol <= Tree1Col)  |   //hit head on with the bottom left corner

              //Plant 2
              (Tree2Row <= Frog_top_leftRow)     & (Frog_top_leftRow <= Tree2Row +
Tree_Height)      & (Tree2Col-Tree_Width <= Frog_top_leftCol)    & (Frog_top_leftCol
<= Tree2Col) |  //head on with top right corner hit
              (Tree2Row <= Frog_bottom_leftRow)  & (Frog_bottom_leftRow <= Tree2Row
+ Tree_Height)   & (Tree2Col-Tree_Width <= Frog_bottom_leftCol) &
(Frog_bottom_leftCol <= Tree2Col)  |   //hit head on with the bottom left corner
```

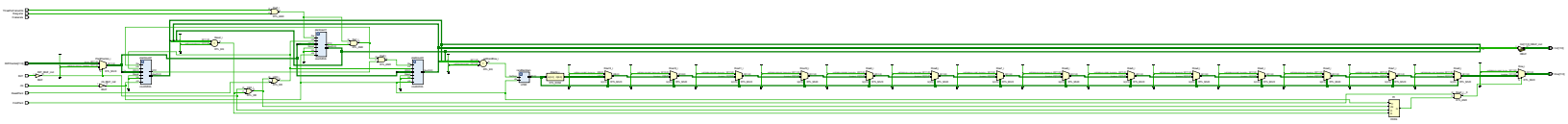
```
              //Plant 3
              (Tree3Row <= Frog_top_leftRow)     & (Frog_top_leftRow <= Tree3Row +
Tree_Height)       & (Tree3Col-Tree_Width <= Frog_top_leftCol)    & (Frog_top_leftCol
<= Tree3Col) |  //head on with top right corner hit
              (Tree3Row <= Frog_bottom_leftRow)  & (Frog_bottom_leftRow <= Tree3Row
+ Tree_Height)   & (Tree3Col-Tree_Width <= Frog_bottom_leftCol) &
(Frog_bottom_leftCol <= Tree3Col)  ;   //hit head on with the bottom left corner



 assign AddOne = (Frog_Start-1 <= Tree1Col) & (Tree1Col <= Frog_Start+1) |  //range
for one frame
                (Frog_Start-1 <= Tree2Col) & (Tree2Col <= Frog_Start+1) |
                (Frog_Start-1 <= Tree3Col) & (Tree3Col <= Frog_Start+1) ;









endmodule
```

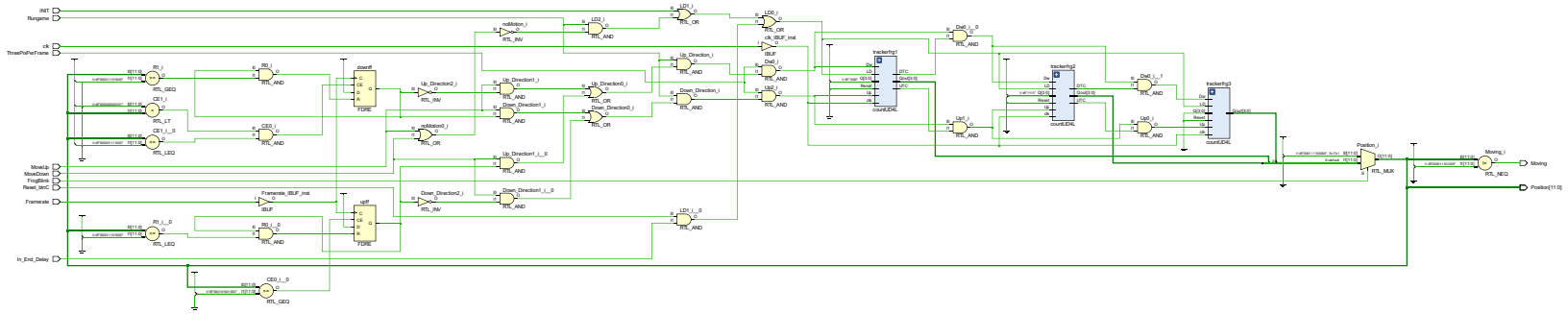


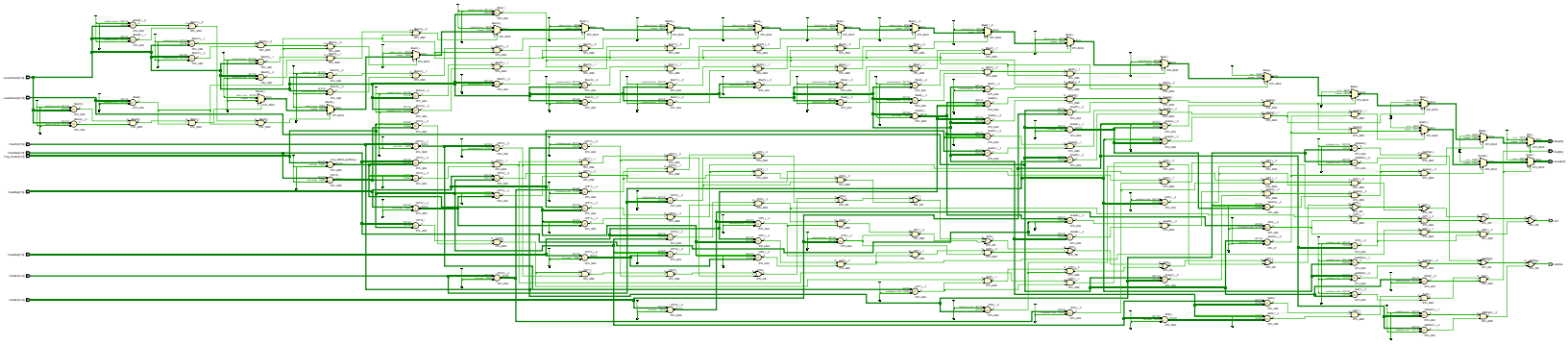

clk
clk_IBUF_inst
I
O
IBUF
Up2_i
I0
O
RTL_INV
tracker1
Dw
LD
Q[3:0]
Reset
Up
clk
Qout[3:0]
UTC
countUD4L
Up1_i
I0
I1
O
RTL_AND
tracker2
Dw
LD
Q[3:0]
Reset
Up
clk
Qout[3:0]
UTC
countUD4L
Reset_i
I0[11:0]
V=B"001100011110"
I1[11:0]
O
RTL_GT
tracker3
Dw
LD
Q[3:0]
Reset
Up
clk
Qout[3:0]
countUD4L
Up0_i
I0
I1
O
RTL_AND
3:0
Position[11:0]_OBUF_inst
I
O
OBUF
Position[11:0]
RowFinish_i
I0[11:0]
V=B"001100011110"
I1[11:0]
>=
O
RTL_GEQ
RowFinish

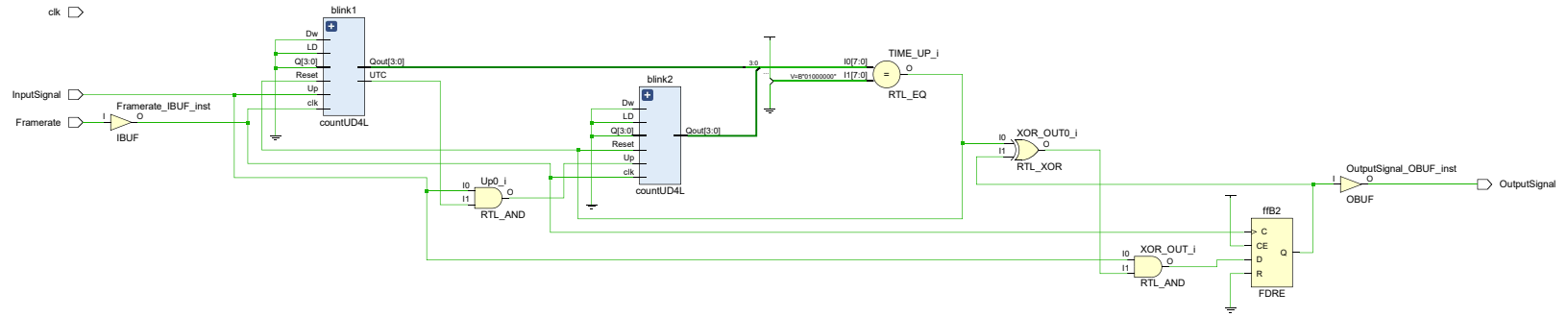

clk
NextCol
clk_IBUF_inst
IBUF
Up3_i
RTL_INV
Up2_i
RTL_AND
tracker1
Dw
LD
Q[3:0]
Reset
Up
clk
Qout[3:0]
UTC
countUD4L
Up1_i
RTL_AND
tracker2
countUD4L
Up0_i
RTL_AND
tracker3
countUD4L
I0[11:0]
I1[11:0]
Reset_i
RTL_GT
Position[11:0]_OBUF_inst
OBUF
Position[11:0]
end_frame
btn
clk
out
Edge_Detector
Frame_OBUF_inst
OBUF
Frame

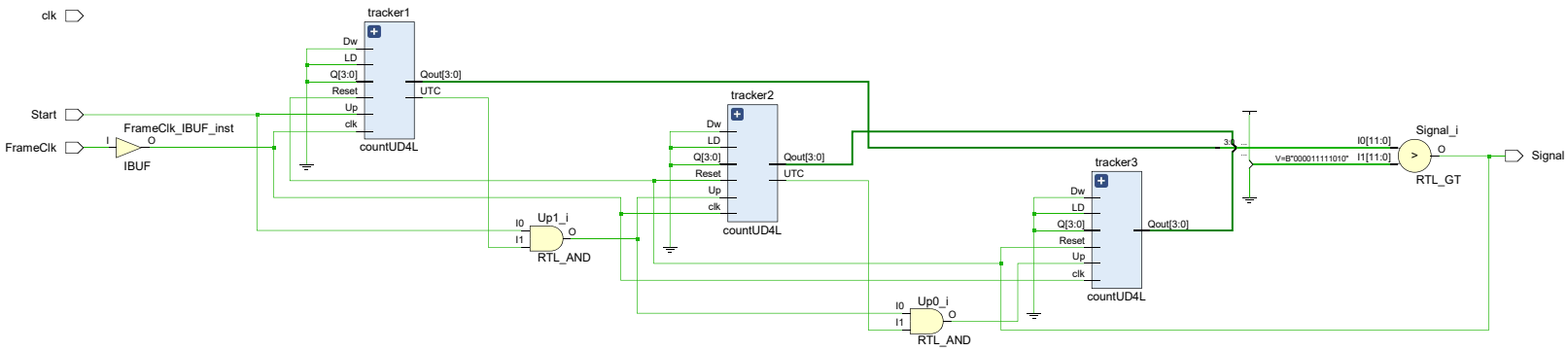





INIT
Rungame
clk
ThreePixPerFrame
MoveUp
MoveDown
FrogBlink
Reset_btnC
Framerate
In_End_Delay
RTL_GEQ
RTL_LT
RTL_LEQ
RTL_AND
RTL_INV
RTL_OR
FDRE
IBUF
countUDML
RTL_MUX
RTL_NEQ
Moving
Position[11:0]




clk
blink1
Dw
LD
Q[3:0]
Reset
Up
clk
Qout[3:0]
UTC
countUD4L
InputSignal
Framerate
Framerate_IBUF_inst
I
O
IBUF
3:0
I0[7:0]
I1[7:0]
V="B"01000000"
TIME_UP_i
=
RTL_EQ
blink2
Qout[3:0]
countUD4L
XOR_OUT0_i
I0
I1
O
RTL_XOR
Up0_i
RTL_AND
OutputSignal_OBUF_inst
OBUF
OutputSignal
ffB2
C
CE
D
R
Q
FDRE
XOR_OUT_i
RTL_AND


clk
tracker1
Dw
LD
Q[3:0]
Reset
Up
clk
Qout[3:0]
UTC
countUD4L
Start
FrameClk_IBUF_inst
I
O
FrameClk
IBUF
tracker2
Dw
LD
Q[3:0]
Reset
Up
clk
Qout[3:0]
UTC
countUD4L
Up1_i
I0
I1
O
RTL_AND
tracker3
Dw
LD
Q[3:0]
Reset
Up
clk
Qout[3:0]
countUD4L
Up0_i
I0
I1
O
RTL_AND
3:0
Signal_i
I0[11:0]
V=B"000011111010"
I1[11:0]
>
O
RTL_GT
Signal