Raymond Huang

Professor Martine Schlag

CSE 100/L Section C

June 4, 2023

Lab 6 Write Up

Overview: In this lab we designed and implemented a version of the game “Bug Fest” through Verilog utilizing the BASYS3 board and a VGA monitor to display the game. The basics of the game is as follows, the slug can fly up and down on the screen and the platforms move from the right to the left, the slug can attach to the platform but if collided with the left end of the platform or with the water below the slug is dead and will no longer move. The objective of the game is to collect as many bugs before the slug dies, where the points are displayed on the BASYS3 board.

Top Module Design: Below is an overview of the top module

Figure 1:

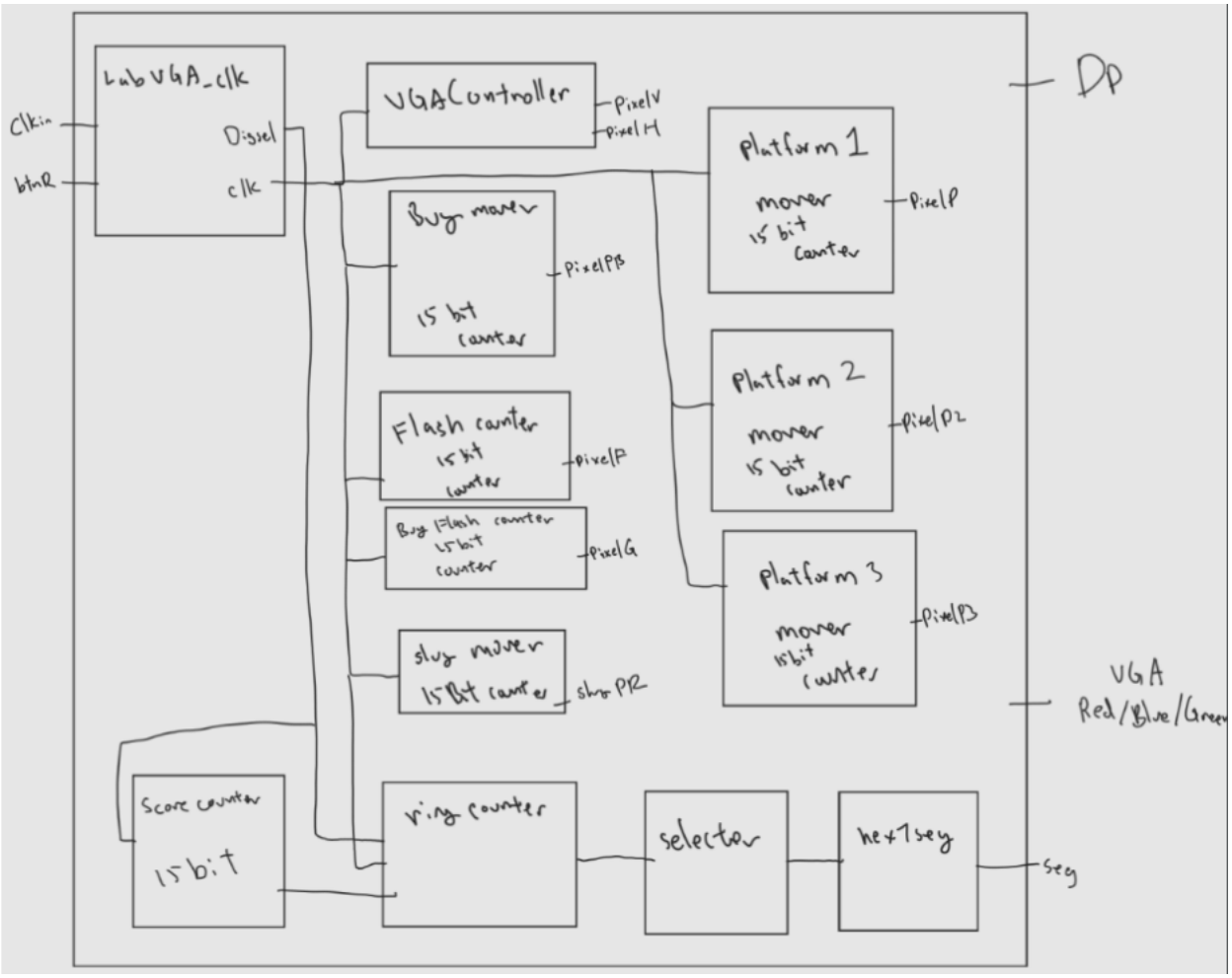


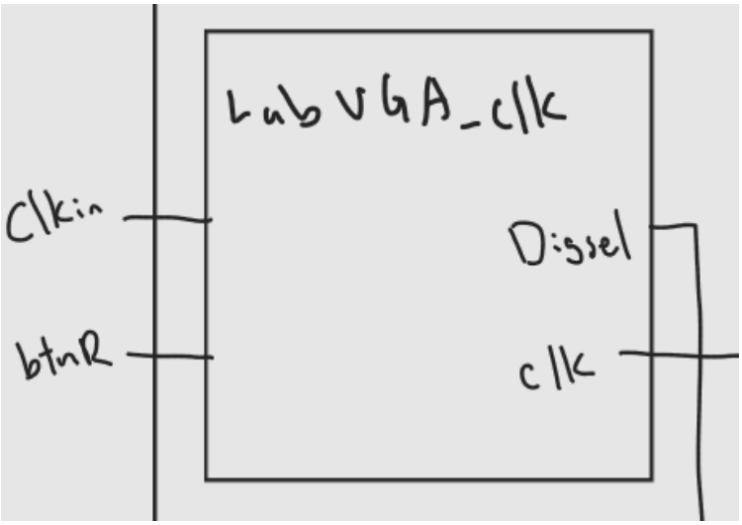
Figure 1 is a very simplified overview of the top module and helps illustrates the functionality of each component in the top module needed for the game to function normally. There are a total of 8 modules utilized. In the first module LabVGA_clk(), shown in figure 2, helps demonstrate its functionality with two inputs, clkin and btnR. This module provides two variables: clk and digsel. The clk variable provides a time measurement system, which is crucial when flashing the score for the winning player. The digsel variable assists us in determining the segments to be displayed on the baysys board which is then utilized to show the score of the game.

It can be implemented using the following module call in the top module:

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

This line takes in clkin and outputs a clk for the user to use and btnR and digsels functions are as described above.

Figure 2:



The clk outputted by the LabVGA_clk() function provides the ability for the operation of various other functions, one of which includes the 15 bit counter. This counter is critical for enabling actions such as movement, blinking, managing pixel addresses, and keeping track of scores. Among these interconnected elements is the VGA controller module. The VGA controller module's use is defined by the use of two of these counters. These counters generate pixelH and pixelV outputs. A detailed representation of the module's inputs and outputs is illustrated in figure 3. These two counters generate the addresses needed for each pixel. Specifically, pixelH represents the horizontal pixels, while pixelV denotes the vertical ones. The counter progresses by incrementing pixelH until it hits its limit. Following this, pixelV increases as pixelH resets back to 0. The resulting numbers can subsequently be utilized to accurately

locate different pixels, both on and off the screen. The code needed to implement this module is provided below:

```
module VGAController(
   input clk,

   output Hsync,
   output Vsync,
   output [14:0]pixelH,
   output [14:0]pixelV
   );
   wire resHor, resVer;
   wire Hs, Vs;

   assign resHor = (pixelH == 15'd799);
   assign resVer = (pixelV == 15'd524);
   assign Hsync = ~((pixelH < 15'd751) & (pixelH > 15'd654));
   assign Vsync = ~((pixelV > 15'd488) & (pixelV < 15'd491));
   counterUD15L C1 (.clk(clk),
                    .Up(1'b1),
                    .Dw(1'b0),
                    .inc(15'b000000000000000),
                    .Q(pixelH),
                    .Ld(resHor)
                    );
   counterUD15L C2(.clk(clk),
                    .Up(resHor),
                    .Dw(1'b0),
                    .inc(15'b000000000000000),
                    .Q(pixelV),
                    .Ld(resHor & resVer)
                    );
endmodule
```

This code gets us the Hsync and Vsync value, where we would check if it is in a specific range. And would be set to low if the condition evaluates to true. This would then be sent back to the top module to be used as an output.

Figure 3:

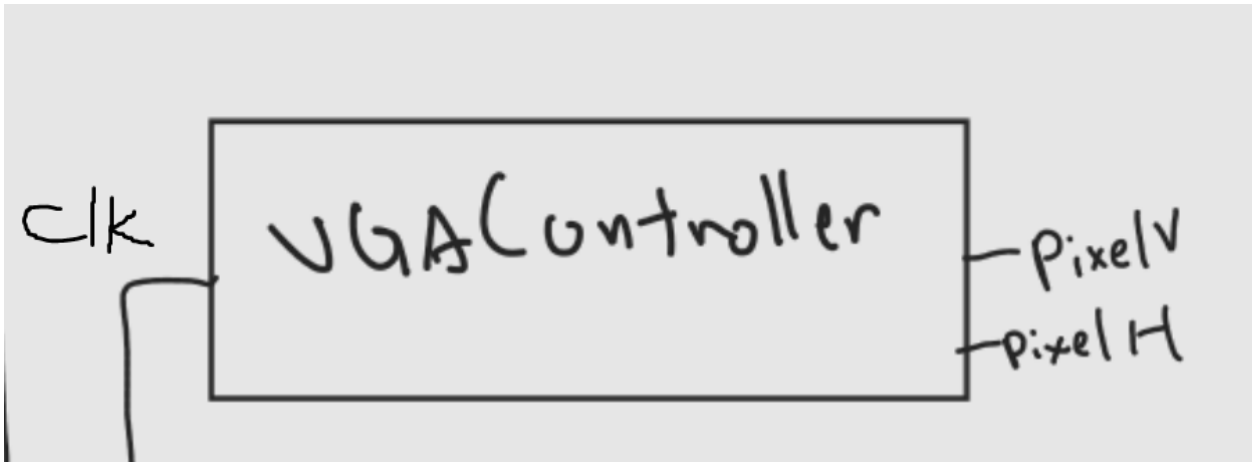
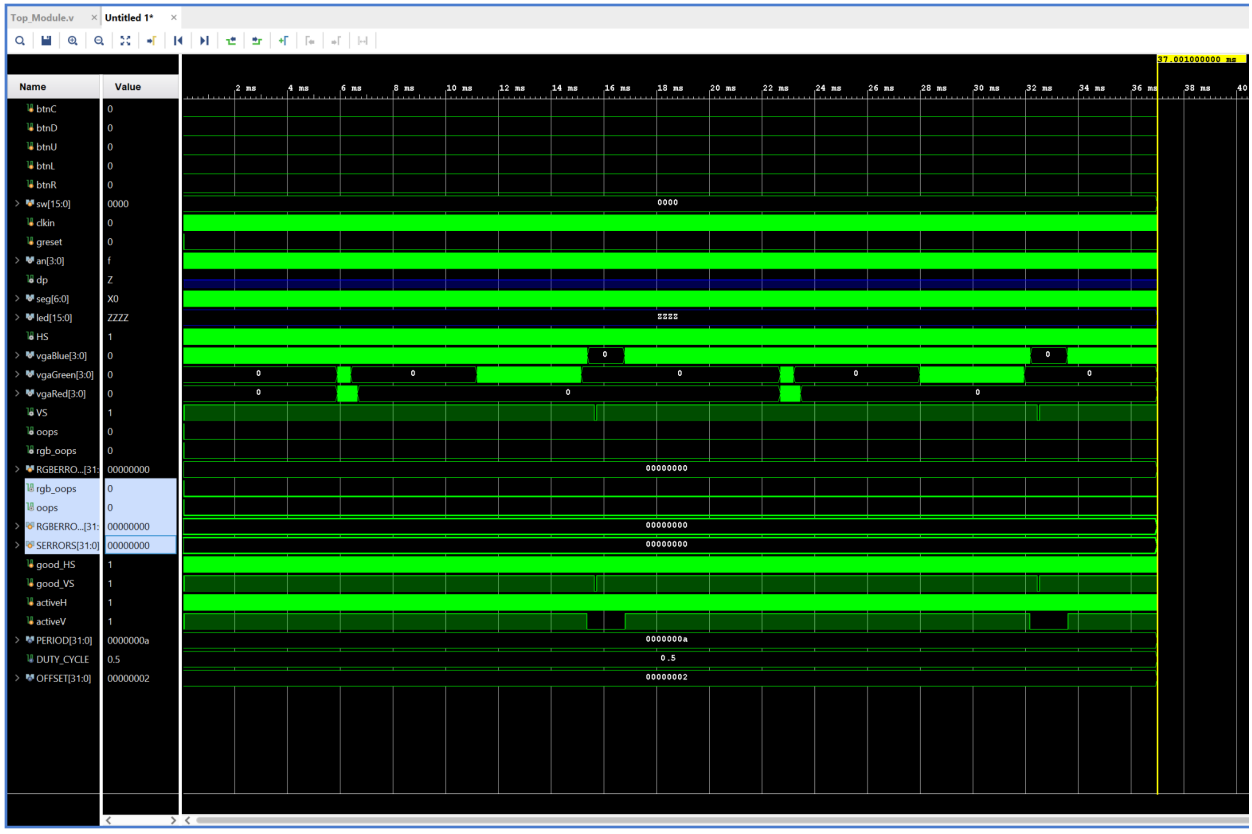


Figure 4:



Everything else takes place in the top module, as I have opted out of modularity for ease of access to everything. In the top module I have other boxes there to signify a simplified version of the functionalities and “sub modules” since they have their module calls and functionality do not intermingle or interfere with the others.

An example of this can be shown when displaying items on the screen, using the pixelH and pixelV from the VGAController, we can make the monitor display certain colors when the pixelH and pixelV are certain values. This can be done by setting a variable to be true when such conditions are met and assigning vgaBlue to be true on said conditions. This can be done through the following code.

```
assign vgaRed = ({4{active}} & {4{Border}} & {4'b0000}) |
            ({4{active}} & {4{platform}} & {4'b1111}) |
            ({4{active}} & {4{platform2}} & {4'b1111}) |
            ({4{active}} & {4{platform3}} & {4'b1111}) |
            ({4{active}} & {4{slug}} & {4'b1111})
            ;
assign vgaBlue = ({4{active}} & {4{Border}} & {4'b1111}) |
             ({4{active}} & {4{pond}} & {4'b1111}) |
             ({4{active}} & {4{platform}} & {4'b1111}) |
             ({4{active}} & {4{platform2}} & {4'b1111})
             ;
assign vgaGreen = ({4{active}} & {4{Border}} & {4'b0000}) |
              ({4{active}} & {4{pond}} & {4'b1111}) |
              ({4{active}} & {4{platform2}} & {4'b1111}) |
              ({4{active}} & {4{slug}} & {4'b1111}) |
              ({4{active}} & {4{bugger}} & {4'b1111})
                ;
```

The next thing is the slug movement, which can be implemented with the code below

```
counterUD15L #(.INIT(15'd199)) slugGo (.clk(clk),.Up((frameS | frame) & ~btnU & ~top_colli & ~bot_colli & ~left_colli_confirm & ~wet),.Dw((frameS |frame) & btnU & ~bot_colli & ~left_colli_confirm & ~wet & ~ceiling),.Q(slugP));

 counterUD15L #(.INIT(15'd140)) slugGoLR (.clk(clk),.Up(),.Dw((frame) & left_colli_confirm & ~pinned),.Q(slugPLR));
```

This code allows us to check which pixel address we want the slug to be displayed at which is then handled by the following code below. This is done with slugGo determining if we are allowed to move up or down with conditions and is it allowed to fall based on another set of conditions. slugGoLR handles if the slug will start forming from the right to left based off if the slug has collided with the side of the platform and will be squished at the edge of the screen.

assign slug = (pH >= slugPLR) & (pH < slugPLR + 10'd16) & (pV <= slugP) & (pV >= (slugP - 10'd15)) & ~flash;

Referencing the slug movement code the platform movements follow an almost identical format, shown in the code below:

```
 //platform 1
    assign frame = (pH == 10'd799) & (pV == 10'd485);
    counterUD15L #(.INIT(15'd300)) platsize (.clk(clk),.Dw(~bug_colli_confirm & frame & start & ~pinned),.Ld((pixelP <= 15'd0)), .inc(15'd900),.Q(pixelP));
    //252-128=124 / 31 = 4
    LFSR rando(.clk(clk),.Q(randowidth));
//    assign randowidth[7:4] = 1'b0;
    wire [3:0]randonum;
    FDRE #(.INIT(1'b1)) r1 [3:0] (.C(clk),.R(1'b0), .CE((pixelP <= 15'd0) & start), .D(randowidth), .Q(randonum));

    assign platwidth = (randonum[1:0] * 15'd32) + 15'd128;

    assign leftP = pH + platwidth;
    //The top vertical coordinate of the platforms is 200

    wire platform;
    assign platform = (pH + platwidth >= pixelP) & (pH <=  pixelP)& (pV <= 10'd207) & (pV >= 10'd200) & ~Border;
```

This platform section of the code generates a pixel position where we would then specify the height and match with the pixel addresses and the width which is randomly generated from an LFSR and that value is held in a flip flop to keep from generating a new width when it is still displayed in the active region. In the counter there is an INIT which loads an initial value this is so the platform spawns on the screen initially before being regenerated off screen as it exits the active region. With one platform done the next three platforms are essentially a copy and paste off this code and variables being changed. The code platform 2 and platform 3 is provided below.

```
    //platform 2
    wire [14:0]platwidth2;
    wire [3:0] randowidth2;
```

```
    wire frame2, topP2, bottomP2, leftP2, rightP2;
    wire [14:0]pixelP2;
    assign frame2 = (pH == 10'd799) & (pV == 10'd485);
    counterUD15L #(.INIT(15'd900)) platsize2 (.clk(clk),.Dw(~bug_colli_confirm & frame &
start & ~pinned),.Ld((pixelP2 <= 15'd0)), .inc(15'd900),.Q(pixelP2));
    //252-128=124 / 31 = 4
    LFSR rando2(.clk(clk),.Q(randowidth2));
//    assign randowidth[7:4] = 1'b0;
    wire [3:0]randonum2;
    FDRE #(.INIT(1'b1)) r2 [3:0] (.C(clk),.R(1'b0), .CE(pixelP2 <= 15'd0), .D(randowidth2),
.Q(randonum2));

    assign platwidth2 = (randonum2[1:0] * 15'd32) + 15'd128;

    assign leftP2 = pH + platwidth2;
    //The top vertical coordinate of the platforms is 200

    wire platform2;
    assign platform2 = (pH + platwidth2 >= pixelP2) & (pH <=  pixelP2)& (pV <= 10'd207) &
(pV >= 10'd200) & ~Border;

     //platform 3
    wire [14:0]platwidth3;
    wire [3:0] randowidth3;
    wire frame3, topP3, bottomP3, leftP3, rightP3;
    wire [14:0]pixelP3;
    assign frame3 = (pH == 10'd799) & (pV == 10'd485);
    counterUD15L #(.INIT(15'd600)) platsize3 (.clk(clk),.Dw(~bug_colli_confirm & frame &
start & ~pinned),.Ld((pixelP3 <= 15'd0)), .inc(15'd900),.Q(pixelP3));
    //252-128=124 / 31 = 4
    LFSR rando3(.clk(clk),.Q(randowidth3));
//    assign randowidth[7:4] = 1'b0;
    wire [3:0]randonum3;
    FDRE #(.INIT(1'b1)) r3 [3:0] (.C(clk),.R(1'b0), .CE(pixelP3 <= 15'd0), .D(randowidth3),
.Q(randonum3));

    assign platwidth3 = (randonum3[1:0] * 15'd32) + 15'd128;

    assign leftP3 = pH + platwidth3;
    //The top vertical coordinate of the platforms is 200

    wire platform3;
    assign platform3 = (pH + platwidth3 >= pixelP3) & (pH <=  pixelP3)& (pV <= 10'd207) &
(pV >= 10'd200) & ~Border;
```

The bug generator follows the same procedure except this time we would have an extra LFSR that dictates where it spawns top or bottom and another LFSR to dictate where to spawn in the vertical pixel range of 128 to 156, and 256 to 284. This can be done through the code below:

```
 //bug generator
    wire [14:0]bug;
    wire [2:0] position;
    wire frameS, topPB, bottomPB, leftPB, rightPB;
    wire [14:0]pixelPB;
    assign frameS = (pH == 15'd799) & (pV == 15'd480);
    counterUD15L #(.INIT(15'd642)) bugGo (.clk(clk),.Dw(~bug_colli_confirm & (frameS |
frame) & start),.Ld((pixelPB <= 15'd0) | (pixelG >= 15'd119)), .inc(15'd642),.Q(pixelPB));
    //252-128=124 / 31 = 4
    LFSR randoB(.clk(clk),.Q(position));
//    assign randowidth[7:4] = 1'b0;
    wire [2:0]randopos;
    FDRE #(.INIT(1'b1)) rB [2:0] (.C(clk),.R(((pixelPB <= 15'd0) | (pixelG >= 15'd199)) &
bug_colli_confirm), .CE((pixelPB <= 15'd0)), .D(position), .Q(randopos));


    wire [9:0] bugspot;
    //top gen
//    assign bugspot[7] = 1'b1;
    assign bugspot[4:2] = randopos[2:0];

    assign bugspot[6:5] = 1'b0;
    assign bugspot[1:0] = 1'b0;

    wire[1:0]opA, opB;

    assign opA[1:0] = 2'd1;
    assign opB[1:0] = 2'd2;

    wire tOb;
    wire holding_rtOb;

    LFSR rtOb(.clk(clk),.Q(holding_rtOb));
    wire why;
    assign why = ((pixelPB <= 15'd0) | (pixelG == 15'd119));
    FDRE #(.INIT(1'b1)) hrtOb [3:0] (.C(clk),.R(), .CE(why), .D(holding_rtOb), .Q(tOb));
    multiplex multi(.i0(opA),.i1(opB),.s(tOb),.o(bugspot[8:7]));

    wire bugger;
```

```
   assign bugger = (pH + 15'd8 >= pixelPB) & (pH <=  pixelPB) & (pV >= bugspot) & (pV <=
bugspot + 15'd8) & ~Border & ~bug_flash;
```

The code above functions as follows, after generating the pixel addresses for movement, we would make it 8 x 8 tall while the sides would be taken care of by the counter as it moves across the screen. The math to get the random range of each section that the bug can spawn goes as follows Max - Min = wiggle room / m = a = 2^n, what this equation essentially does is taking the difference of the lowest and highest point of that specific section, we are able to get the area in which the bug can spawn initially. M would an arbitrary whole number we would divide to get a number which conveniently is equal to 2 to the Nth power, and the Nth power would be the number of bits for the random number, what I did was 252-128=124 / 31 = 4 = 2 ^ 2 so my random position was 2 bits.

My collide logic checks if the slug overlaps with any of the other objects in the active region. This is done by checking the slugs position and seeing if another object is in the same position. A simplified example of this would be when we would check if the slug is going to collide with the top border or going to collide with the pond below. In order to do this we would check the vertical counter of the slug, if it matches the top border or the height of the pond you would set the variable to high and in the counter, you would not allow it to increment or decrement the counter more if the condition equates to true. This logic stays true with colliding with the platform and the bug with a few extra steps.

```
   //slug collison
   wire platcolli;
   assign platcolli = (((pixelP3 <= platwidth3) | ((pixelP3 - platwidth3) <= 10'd155)) & (pixelP3
>= 10'd140)) |
                  (((pixelP2 <= platwidth2) | ((pixelP2 - platwidth2) <= 10'd155)) & (pixelP2 >=
10'd140)) |
                  (((pixelP <= platwidth) | ((pixelP - platwidth) <= 10'd155)) & (pixelP >=
10'd140))
                   ;
```

```
   wire top_colli, bot_colli, left_colli, ceiling;
   assign wet = (slugP > 15'd350);
   assign ceiling = (slugP - 10'd15 < 15'd9);

   assign pinned = slugPLR == 8;
   assign stopper = slugPLR <= 8 & (((pixelP3 - platwidth3 - 10'd1) == 10'd6) |
                    ((pixelP2 - platwidth2 - 10'd1) == 10'd6) |
                    ((pixelP - platwidth - 10'd1) == 10'd6));

   assign top_colli = platcolli & (slugP == 10'd199);
   assign bot_colli = platcolli & ((slugP - 10'd15) == 10'd208);
   assign left_colli = (((pixelP3 - platwidth3 - 10'd1) == 10'd155) |
                    ((pixelP2 - platwidth2 - 10'd1) == 10'd155) |
                    ((pixelP - platwidth - 10'd1) == 10'd155)) &
                    (slugP >= 10'd200) & (slugP - 10'd15 <= 10'd207);

   assign bug_colli = (((pixelPB == 10'd140) | (pixelPB - 10'd8 == 10'd154)) & (bugspot - 10'd1
<= slugP) & ((bugspot + 10'd8) >= (slugP - 10'd15)))
                    | (((pixelPB + 10'd1) >= 10'd140) & (pixelPB - 10'd8 <= 10'd155)) & ((bugspot -
10'd1 == slugP) | ((bugspot + 10'd8) == (slugP - 10'd15)))
                    ;
```

Referencing the code above we can see how the collision for the platform works. With the top and bottom collide, we can see that it checks if the top of the platform is within the slugs movement range and the slug is at the vertical pixel height of 199 and the bottom one is the same except we would check if the top of the slug is at the vertical pixel height of 208. We are doing one pixel away from the actual platform to avoid any form of overlapping. Left collision is a little more work as we need to get the width of the platform - 1 to get the outer left side to use for detection of left collision. Once any of the platform collisions is detected with the exception of the collision that happens when the slug lands on the platform we would cease all vertical movement. With the left collision we would add the case of horizontal movement as it has been hit and will collide. The bug collision is similar as we just do the similar logic except movement would not be halted.

For flashing we made another counter to count the number of frames that have passed. We can see the code implementation below:

```
assign frame = (pH == 10'd799) & (pV == 10'd485);
   counterUD15L #(.INIT(15'd0)) framcounter (.clk(clk),.Up(frame & (wet | left_colli_confirm)),.Ld(pixelF == 15'd129), .inc(15'd0),.Q(pixelF));
   counterUD15L #(.INIT(15'd0)) bugblinker (.clk(clk),.Up(frame & (bug_colli_confirm)),.Ld(pixelG == 15'd119), .inc(15'd0),.Q(pixelG));
```

Using this frame counting code we are able to decide at which frames we want things to appear and disappear to make it result in a blinking object on the screen. The code to implement this is below:

```
wire FlashTiming, BugFlashTiming;

   assign BugFlashTiming = (((pixelG) > 0) & ((pixelG) <= 10)) | (((pixelG) >= 20) & ((pixelG) <= 30)) |
                           (((pixelG) >= 40) & ((pixelG) <= 50)) | (((pixelG) >= 60) & ((pixelG) <= 70)) |
                           (((pixelG) >= 80) & ((pixelG) <= 90)) | (((pixelG) >= 100) & ((pixelG) <= 110));

   assign FlashTiming = (((pixelF) > 0) & ((pixelF) <= 10)) | ((pixelF >= 10) & ((pixelF) <= 20)) |  ((pixelF >= 30) & (pixelF <= 40)) | ((pixelF >= 50) & (pixelF <= 60)) | ((pixelF >= 70) & (pixelF <= 80)) |
                           ((pixelF >= 90) & (pixelF <= 100))|
                           ((pixelF >= 110) & (pixelF <= 120));
```

We can see that whatever is specified is on and what's not specified in the range is effectively turning the object off. And for the bug flashing we can increment the point counter when the bug flashing counter has reached the end of flashing as the documentation video showcased.

Figure 5: Design Timing Summary

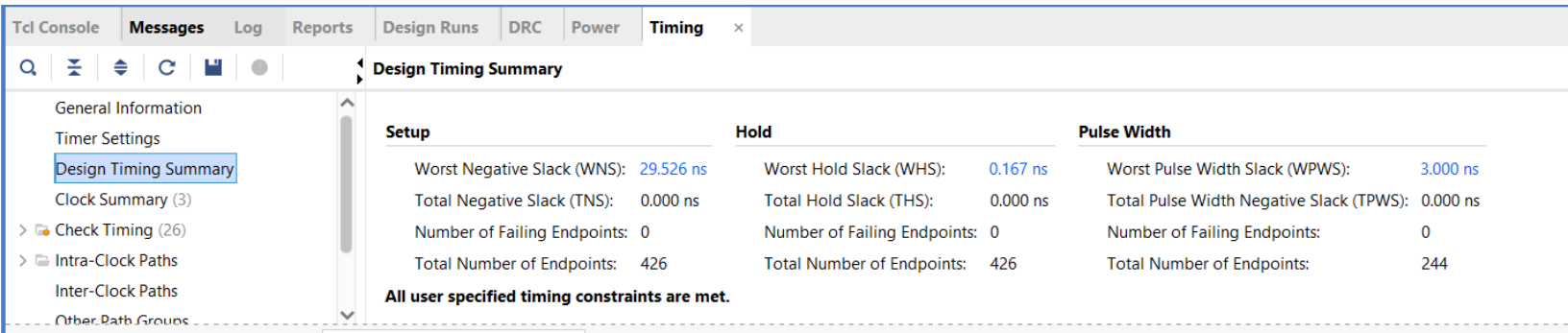
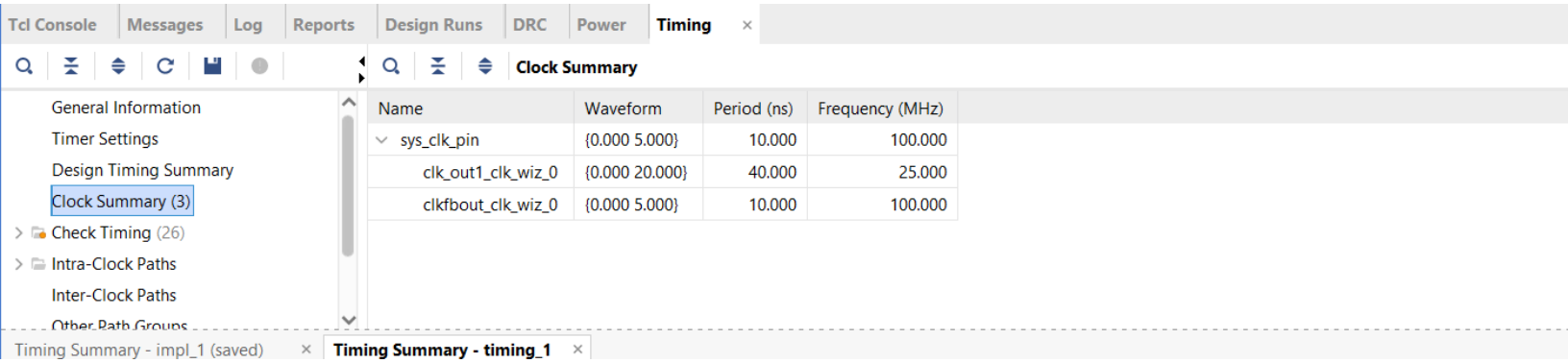


Figure 6: Clock Summary



Calculations for maximum clock frequency

40 - 29.526 = 10.474

1 / 10.474 = 0.095

Appendix:

Top Module:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2023 03:01:55 PM
// Design Name:
// Module Name: Top_Module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Top_Module(
    input btnU,
    input btnD,
    input btnC,
    input btnR,  //btnR is the global reset in S23 Lab6
    input btnL,
    input clkin,
```

```
    output [6:0]seg,
    output dp,
    output [3:0]an,
    output [3:0]vgaBlue,
    output [3:0]vgaRed,
    output [3:0]vgaGreen,
    output Vsync,
    output Hsync,
    input [15:0]sw,
    output [15:0]led
    );

    wire [14:0] pH;
    wire [14:0] pV;
    wire [3:0] vgaR;
    wire [3:0] vgaB;
    wire [3:0] vgaG;
    wire Top, Left, Right, Bottom, digsel, clk, active,
Border, pond, start, pinned, stopper,
bug_colli_confirm, frame, flash, wet;
    wire left_colli_confirm;
    wire bug_colli, bug_flash;
    wire [14:0]slugPLR;
    wire [14:0]platwidth;
    wire [3:0] randowidth;
    wire topP, bottomP, leftP, rightP;
    wire [14:0]pixelP;
    wire [14:0]pixelF, pixelG;


    //frame counter
```

```
    assign frame = (pH == 10'd799) & (pV == 10'd485);
    counterUD15L #(.INIT(15'd0)) framcounter
(.clk(clk),.Up(frame & (wet |
left_colli_confirm)),.Ld(pixelF == 15'd129),
.inc(15'd0),.Q(pixelF));
    counterUD15L #(.INIT(15'd0)) bugblinker
(.clk(clk),.Up(frame & (bug_colli_confirm)),.Ld(pixelG
== 15'd119), .inc(15'd0),.Q(pixelG));
    //start
    FDRE #(.INIT(1'b0)) starting (.C(clk),.R(1'b0),
.CE(btnC), .D(1'b1), .Q(start));

    //platform stopper
     //assign top left right and bottom border for VGA
Blue
    labVGA_clks not_so_slow (.clkin(clkin),
.greset(btnR), .clk(clk), .digsel(digsel));
    VGAController VGA
(.clk(clk),.pixelH(pH),.pixelV(pV),.Hsync(Hsync),.Vsync
(Vsync));



    assign active = (pH <= 639) & (pV <=479);

    //border
    assign Top = (pV < 8) & active;
    assign Left = (pH < 8) & active;
    assign Right = ((pH < 640) & (pH > 632)) & active;
    assign Bottom = (pV < 480) & (pV > 472) & active;

    assign Border = Top | Left | Right | Bottom;
```

```
    //pond

    assign pond = ((pV >= 350) & active) & ~Border;

    //platform 1
    assign frame = (pH == 10'd799) & (pV == 10'd485);
    counterUD15L #(.INIT(15'd300)) platsize
(.clk(clk),.Dw(~bug_colli_confirm & frame & start &
~pinned),.Ld((pixelP <= 15'd0)),
.inc(15'd900),.Q(pixelP));
    //252-128=124 / 31 = 4
    LFSR rando(.clk(clk),.Q(randowidth));
//    assign randowidth[7:4] = 1'b0;
    wire [3:0]randonum;
    FDRE #(.INIT(1'b1)) r1 [3:0] (.C(clk),.R(1'b0),
.CE((pixelP <= 15'd0) & start), .D(randowidth),
.Q(randonum));

    assign platwidth = (randonum[1:0] * 15'd32) +
15'd128;

    assign leftP = pH + platwidth;
    //The top vertical coordinate of the platforms is
200

    wire platform;
    assign platform = (pH + platwidth >= pixelP) & (pH
<=  pixelP)& (pV <= 10'd207) & (pV >= 10'd200) &
~Border;
```

```
    //platform 2
    wire [14:0]platwidth2;
    wire [3:0] randowidth2;
    wire frame2, topP2, bottomP2, leftP2, rightP2;
    wire [14:0]pixelP2;
    assign frame2 = (pH == 10'd799) & (pV == 10'd485);
    counterUD15L #(.INIT(15'd900)) platsize2
(.clk(clk),.Dw(~bug_colli_confirm & frame & start &
~pinned),.Ld((pixelP2 <= 15'd0)),
.inc(15'd900),.Q(pixelP2));
    //252-128=124 / 31 = 4
    LFSR rando2(.clk(clk),.Q(randowidth2));
//    assign randowidth[7:4] = 1'b0;
    wire [3:0]randonum2;
    FDRE #(.INIT(1'b1)) r2 [3:0] (.C(clk),.R(1'b0),
.CE(pixelP2 <= 15'd0), .D(randowidth2), .Q(randonum2));

    assign platwidth2 = (randonum2[1:0] * 15'd32) +
15'd128;

    assign leftP2 = pH + platwidth2;
    //The top vertical coordinate of the platforms is
200

    wire platform2;
    assign platform2 = (pH + platwidth2 >= pixelP2) &
(pH <=  pixelP2)& (pV <= 10'd207) & (pV >= 10'd200) &
~Border;
```

```
     //platform 3
    wire [14:0]platwidth3;
    wire [3:0] randowidth3;
    wire frame3, topP3, bottomP3, leftP3, rightP3;
    wire [14:0]pixelP3;
    assign frame3 = (pH == 10'd799) & (pV == 10'd485);
    counterUD15L #(.INIT(15'd600)) platsize3
(.clk(clk),.Dw(~bug_colli_confirm & frame & start &
~pinned),.Ld((pixelP3 <= 15'd0)),
.inc(15'd900),.Q(pixelP3));
    //252-128=124 / 31 = 4
    LFSR rando3(.clk(clk),.Q(randowidth3));
//    assign randowidth[7:4] = 1'b0;
    wire [3:0]randonum3;
    FDRE #(.INIT(1'b1)) r3 [3:0] (.C(clk),.R(1'b0),
.CE(pixelP3 <= 15'd0), .D(randowidth3), .Q(randonum3));

    assign platwidth3 = (randonum3[1:0] * 15'd32) +
15'd128;

    assign leftP3 = pH + platwidth3;
    //The top vertical coordinate of the platforms is
200

    wire platform3;
    assign platform3 = (pH + platwidth3 >= pixelP3) &
(pH <=  pixelP3)& (pV <= 10'd207) & (pV >= 10'd200) &
~Border;


   //bug generator
```

```
    wire [14:0]bug;
    wire [2:0] position;
    wire frameS, topPB, bottomPB, leftPB, rightPB;
    wire [14:0]pixelPB;
    assign frameS = (pH == 15'd799) & (pV == 15'd480);
    counterUD15L #(.INIT(15'd642)) bugGo
(.clk(clk),.Dw(~bug_colli_confirm & (frameS | frame) &
start),.Ld((pixelPB <= 15'd0) | (pixelG >= 15'd119)),
.inc(15'd642),.Q(pixelPB));
    //252-128=124 / 31 = 4
    LFSR randoB(.clk(clk),.Q(position));
//    assign randowidth[7:4] = 1'b0;
    wire [2:0]randopos;
    FDRE #(.INIT(1'b1)) rB [2:0] (.C(clk),.R(((pixelPB
<= 15'd0) | (pixelG >= 15'd199)) & bug_colli_confirm),
.CE((pixelPB <= 15'd0)), .D(position), .Q(randopos));


    wire [9:0] bugspot;
    //top gen
//    assign bugspot[7] = 1'b1;
    assign bugspot[4:2] = randopos[2:0];

    assign bugspot[6:5] = 1'b0;
    assign bugspot[1:0] = 1'b0;

    wire[1:0]opA, opB;

    assign opA[1:0] = 2'd1;
    assign opB[1:0] = 2'd2;
```

```
    wire tOb;
    wire holding_rtOb;

    LFSR rtOb(.clk(clk),.Q(holding_rtOb));
    wire why;
    assign why = ((pixelPB <= 15'd0) | (pixelG ==
15'd119));
    FDRE #(.INIT(1'b1)) hrtOb [3:0] (.C(clk),.R(),
.CE(why), .D(holding_rtOb), .Q(tOb));
    multiplex
multi(.i0(opA),.i1(opB),.s(tOb),.o(bugspot[8:7]));

    wire bugger;
    assign bugger = (pH + 15'd8 >= pixelPB) & (pH <=
pixelPB) & (pV >= bugspot) & (pV <= bugspot + 15'd8) &
~Border & ~bug_flash;

    //counter - lsfr to get the size
    //grab a frame to get the timing
    //decrement with another counter to move
    //Banana Tree Kicker
    wire slug, slugTop, slugBot, sluglef, slugrig,
slugH;
    wire [14:0]slugP;
//    assign slugS = (pH >= 15'd140) & (pH <= 15'd156);
////    assign slugL = slugH + 15'd16;
////    assign slugB = (pV == 15'd199);
//    assign slugH = (pV >= 15'd184) & (pV <= 15'd199);
//    assign slug = slugH & slugS;
```

```
    //slug collison
    wire platcolli;
    assign platcolli = (((pixelP3 <= platwidth3) |
((pixelP3 - platwidth3) <= 10'd155)) & (pixelP3 >=
10'd140)) |
                       (((pixelP2 <= platwidth2) |
((pixelP2 - platwidth2) <= 10'd155)) & (pixelP2 >=
10'd140)) |
                       (((pixelP <= platwidth) |
((pixelP - platwidth) <= 10'd155)) & (pixelP >=
10'd140))
                        ;

    wire top_colli, bot_colli, left_colli, ceiling;

    assign pinned = slugPLR == 8;
    assign stopper = slugPLR <= 8 & (((pixelP3 -
platwidth3 - 10'd1) == 10'd6) |
                        ((pixelP2 - platwidth2 - 10'd1)
== 10'd6) |
                        ((pixelP - platwidth - 10'd1)
== 10'd6));

    assign top_colli = platcolli & (slugP == 10'd199);
    assign bot_colli = platcolli & ((slugP - 10'd15) ==
10'd208);
    assign left_colli = (((pixelP3 - platwidth3 -
10'd1) == 10'd155) |
                        ((pixelP2 - platwidth2 - 10'd1)
== 10'd155) |
                        ((pixelP - platwidth - 10'd1)
```

```
== 10'd155)) &
                        (slugP >= 10'd200) & (slugP -
10'd15 <= 10'd207);

    assign bug_colli = (((pixelPB == 10'd140) |
(pixelPB - 10'd8 == 10'd154)) & (bugspot - 10'd1 <=
slugP) & ((bugspot + 10'd8) >= (slugP - 10'd15)))
                       | (((pixelPB + 10'd1) >=
10'd140) & (pixelPB - 10'd8 <= 10'd155)) & ((bugspot -
10'd1 == slugP) | ((bugspot + 10'd8) == (slugP -
10'd15)))
                        ;

    wire bugres;
    assign bugres = pixelG < 119;
    FDRE #(.INIT(1'b0)) left_BAM (.C(clk),.R(1'b0),
.CE(1'b1), .D(left_colli | left_colli_confirm ),
.Q(left_colli_confirm));
    FDRE #(.INIT(1'b0)) bug_smack (.C(clk),.R(1'b0),
.CE(1'b1), .D((bug_colli | bug_colli_confirm) &
bugres), .Q(bug_colli_confirm));
    assign wet = (slugP > 15'd350);
    assign ceiling = (slugP - 10'd15 < 15'd9);
//    assign led[0] = left_colli_confirm;
//    assign led[1] = top_colli;
//    assign led[2] = bot_colli;
//    assign led[15] = wet;
//    assign led[5] = bug_colli_confirm;

    wire FlashTiming, BugFlashTiming;
```

```
   assign BugFlashTiming = (((pixelG) > 0) & ((pixelG)
<= 10)) | (((pixelG) >= 20) & ((pixelG) <= 30)) |
                           (((pixelG) >= 40) & ((pixelG)
<= 50)) | (((pixelG) >= 60) & ((pixelG) <= 70)) |
                           (((pixelG) >= 80) & ((pixelG)
<= 90)) | (((pixelG) >= 100) & ((pixelG) <= 110));


   assign FlashTiming = (((pixelF) > 0) & ((pixelF) <=
10)) | ((pixelF >= 10) & ((pixelF) <= 20)) |  ((pixelF
>= 30) & (pixelF <= 40)) | ((pixelF >= 50) & (pixelF <=
60)) | ((pixelF >= 70) & (pixelF <= 80)) |
                             ((pixelF >= 90) & (pixelF <=
100))|
                             ((pixelF >= 110) & (pixelF
<= 120));


   // assign led[10] = FlashTiming;
    assign flash = (left_colli_confirm | wet) &
FlashTiming;
    assign bug_flash = bug_colli_confirm &
BugFlashTiming;




//////////////////////////////////////////////////////
//////////
    counterUD15L #(.INIT(15'd199)) slugGo
(.clk(clk),.Up((frameS | frame) & ~btnU & ~top_colli &
```

```
~bot_colli & ~left_colli_confirm & ~wet),
                    .Dw((frameS |frame) & btnU &
~bot_colli & ~left_colli_confirm & ~wet &
~ceiling),.Q(slugP));
    counterUD15L #(.INIT(15'd140)) slugGoLR
(.clk(clk),.Up(),
                    .Dw((frame) & left_colli_confirm &
~pinned),.Q(slugPLR));
//    assign slugrig = pH + 15'd16 >= slugP;
//    assign sluglef = pH <= slugP;
//    assign slugTop = slugH >= pV;
//    assign slugBot = slugH <= pV + 15'd16;
//    assign slug = slugrig & sluglef & slugTop &
slugBot;
    wire [3:0]ringOutput;
    wire [15:0]out;
    wire [3:0]H;
    counterUD15L #(.INIT(15'd0)) slugPoint
(.clk(clk),.Up((pixelG == 10'd119)),
                    .Dw(),.Q(out));
    ringCounter
rc(.Qput(ringOutput[3:0]),.Inc(digsel),.clk(clk));
    selector
sellout(.sel(ringOutput[3:0]),.N(out[15:0]),.H(H[3:0]))
;
    hex7seg s7eg(.n(H[3:0]),.seg(seg));
    assign slug = (pH >= slugPLR) & (pH < slugPLR +
10'd16) & (pV <= slugP) & (pV >= (slugP - 10'd15)) &
~flash;
    assign vgaRed = ({4{active}} & {4{Border}} &
{4'b0000}) |
```

```
                    ({4{active}} & {4{platform}} &
{4'b1111}) |
                    ({4{active}} & {4{platform2}} &
{4'b1111}) |
                    ({4{active}} & {4{platform3}} &
{4'b1111}) |
                    ({4{active}} & {4{slug}} &
{4'b1111})
                   ;
    assign vgaBlue = ({4{active}} & {4{Border}} &
{4'b1111}) |
                     ({4{active}} & {4{pond}} &
{4'b1111}) |
                     ({4{active}} & {4{platform}} &
{4'b1111}) |
                     ({4{active}} & {4{platform2}} &
{4'b1111})
                     ;
    assign vgaGreen = ({4{active}} & {4{Border}} &
{4'b0000}) |
                      ({4{active}} & {4{pond}} &
{4'b1111}) |
                      ({4{active}} & {4{platform2}} &
{4'b1111}) |
                      ({4{active}} & {4{slug}} &
{4'b1111}) |
                      ({4{active}} & {4{bugger}} &
{4'b1111})
                      ;

    assign an = ~ringOutput[1:0];
```

```
endmodule
```

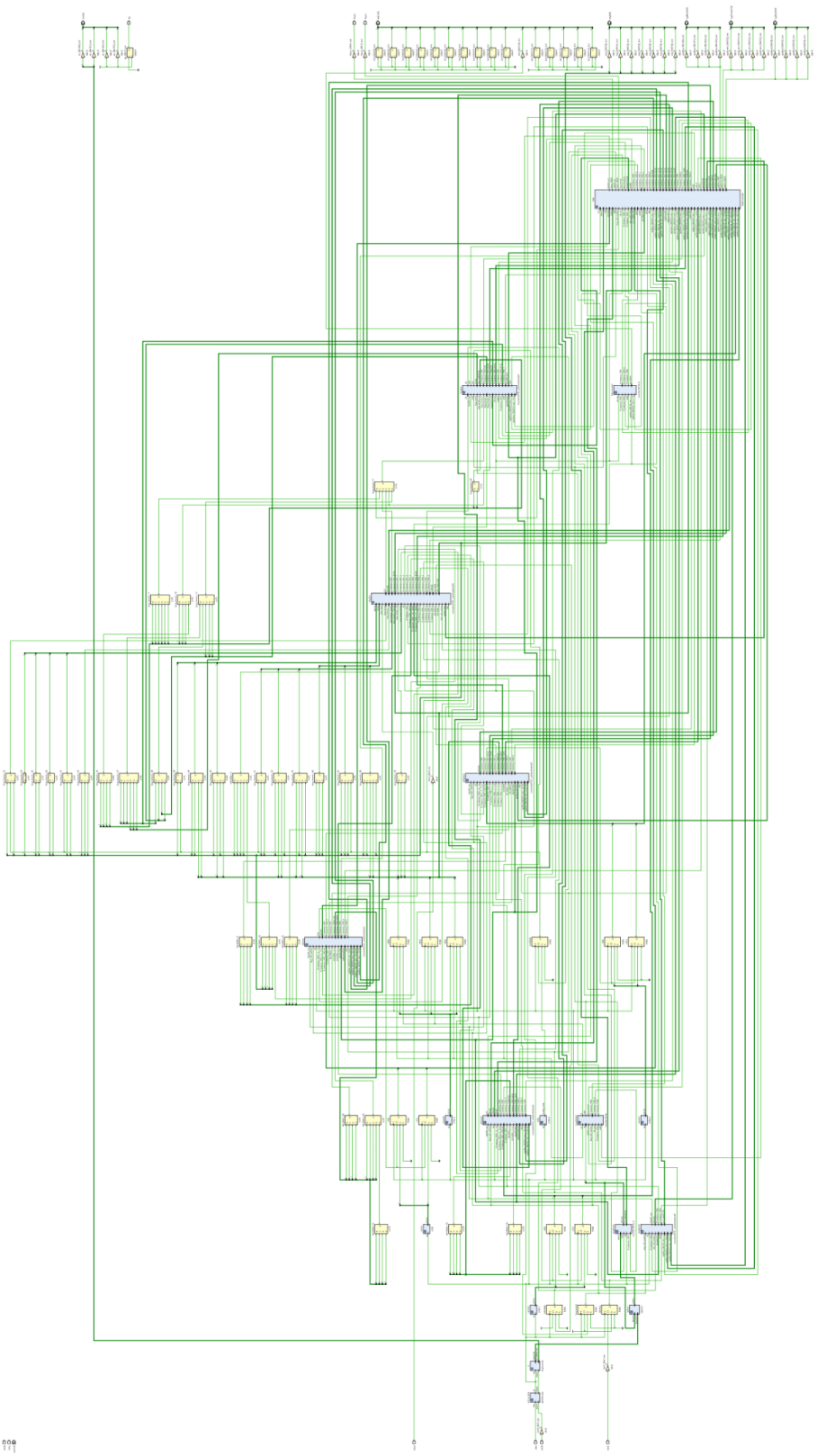
Vga Controller:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2023 12:46:05 PM
// Design Name:
// Module Name: VGAController
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module VGAController(
    input clk,

    output Hsync,
    output Vsync,
    output [14:0]pixelH,
    output [14:0]pixelV
    );
    wire resHor, resVer;
    wire Hs, Vs;

    assign resHor = (pixelH == 15'd799);
    assign resVer = (pixelV == 15'd524);
```

```
    assign Hsync = ~((pixelH < 15'd751) & (pixelH > 15'd654));
    assign Vsync = ~((pixelV > 15'd488) & (pixelV < 15'd491));
    counterUD15L C1 (.clk(clk),
                     .Up(1'b1),
                     .Dw(1'b0),
                     .inc(15'b000000000000000),
                     .Q(pixelH),
                     .Ld(resHor)
                     );
    counterUD15L C2(.clk(clk),
                     .Up(resHor),
                     .Dw(1'b0),
                     .inc(15'b000000000000000),
                     .Q(pixelV),
                     .Ld(resHor & resVer)
                     );
//    assign pH = pixelH;
//    wire [14:0]pixelV;
//    wire [14:0]pixelH;
//    assign pV = pixelV;
//    FDRE #(.INIT(1'b1) ) ff1 (.C(clk), .R(), .CE(), .D(Hs),
.Q(Hsync));
//    FDRE #(.INIT(1'b1) ) ff2 (.C(clk), .R(), .CE(), .D(Vs),
.Q(Vsync));


endmodule
```

Schematic:

Top Module:



VGAControl: