Gea Loro
June 9th, 2023
Section C

Lab 6 Write-Up

Description:

The objective of this lab report is to document the design and implementation of a customized version of the game "Bug Fest" using the BASYS3 board and the VGA monitor connected to it. The Bug Fest game involves controlling a player character called the "slug," which flies between moving platforms in an attempt to catch green bugs. The game ends if the slug falls into the pond below or is pushed against the left wall by a platform.

To control the slug, specific buttons on the BASYS3 board will be utilized. The "btnU" button allows the slug to fly upwards when pressed, and gravity takes effect when the button is released, causing the slug to fall unless it is on or clinging to a platform. The "btnC" button is used to initiate the game, starting the movement of platforms and the appearance of bugs.

Throughout the game, each intercepted bug adds a point to the player's score, which is constantly displayed on the right two digits of the 7-segment display. The goal is to score as many points as possible by intercepting bugs while avoiding falling into the pond or colliding with the left wall.

The implementation of this game requires programming and hardware integration skills. By designing and implementing our own version of Bug Fest, we can explore concepts such as input handling, platform and bug movement, collision detection, and score tracking.

In this lab report, we will provide a detailed account of our design process, explaining the various components and functionalities we incorporated into the game. We will also discuss the challenges encountered during the implementation and present a thorough analysis of the final game's performance and user experience.

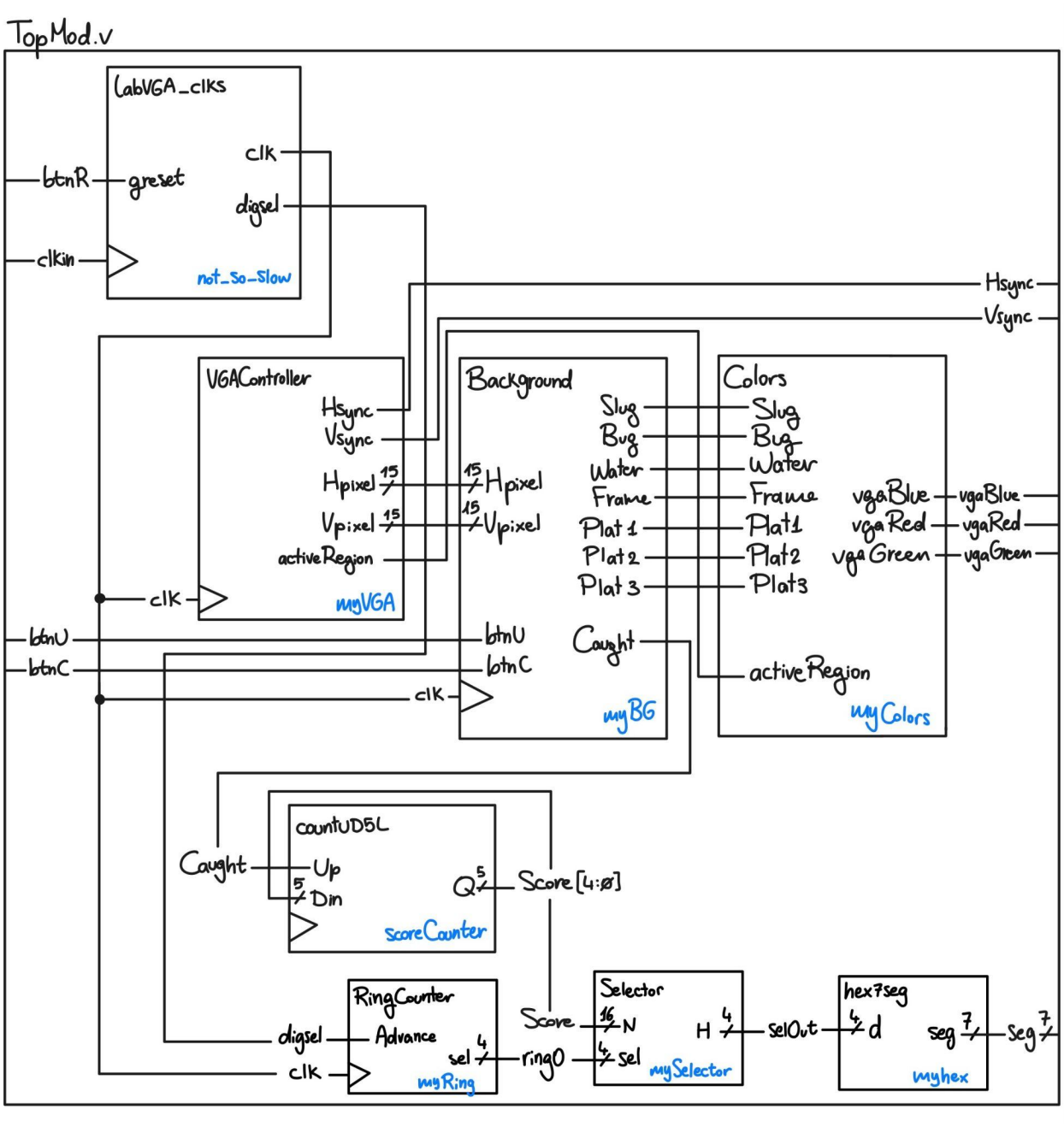
Design:

Top Mod

- Input Handling:
    - btnU: This input signal is used to control the slug's upward movement. When pressed, the slug moves up, defying gravity.
    - btnC: This input signal starts the game. When pressed, the platforms begin moving, and bugs appear from the right side of the screen.
    - btnR: This input signal is used as a reset signal.
- VGA Display Control:
    - Hsync and Vsync: These output signals are responsible for generating horizontal and vertical synchronization signals for the VGA monitor, ensuring proper display synchronization.
    - Hpixel and Vpixel: These output signals represent the current pixel position on the VGA monitor.
    - activeRegion: This output signal indicates whether the current pixel position is within the active display region.
- Background Rendering:
    - Water, Frame, Slug, Bug, Plat1, Plat2, Plat3: These output signals control the rendering of different elements in the game, such as water, frame, slug, bugs, and platforms.
- Score Tracking:
    - Score: This wire signal represents the current score in the game. It is incremented whenever a bug is intercepted by the slug.

    - Caught: This wire signal is triggered when a bug is intercepted by the slug, indicating a successful catch.
- Ring Counter and Selector:
    - ringO: This wire signal represents the output of a ring counter, which advances based on specific conditions
    - selOut: This wire signal represents the output of a selector module, which selects a value based on the value of ringO.
    - an[3:0]: These output signals control the segments of a 7-segment display, displaying the score.
- Colors:
    - vgaBlue, vgaRed, vgaGreen: These output signals control the RGB values for the VGA monitor, determining the color of various game elements based on specific conditions.
- Other Wires:
    - clk: This wire signal represents the clock signal used for synchronization and timing within the module.
    - digsel: This wire signal is used for digit selection in the 7-segment display.
    - moving, Start, SlugLeftPlat, SlugBottomPlat, SlugTopPlat: These wire signals are used for various game logic and conditions.

The functionalities described above collectively enable the Bug Fest game to be played on the BASYS3 board and VGA monitor. The code handles user input, controls VGA display rendering, tracks the score, updates the 7-segment display, and manages various game elements such as the slug, bugs, platforms, and background.

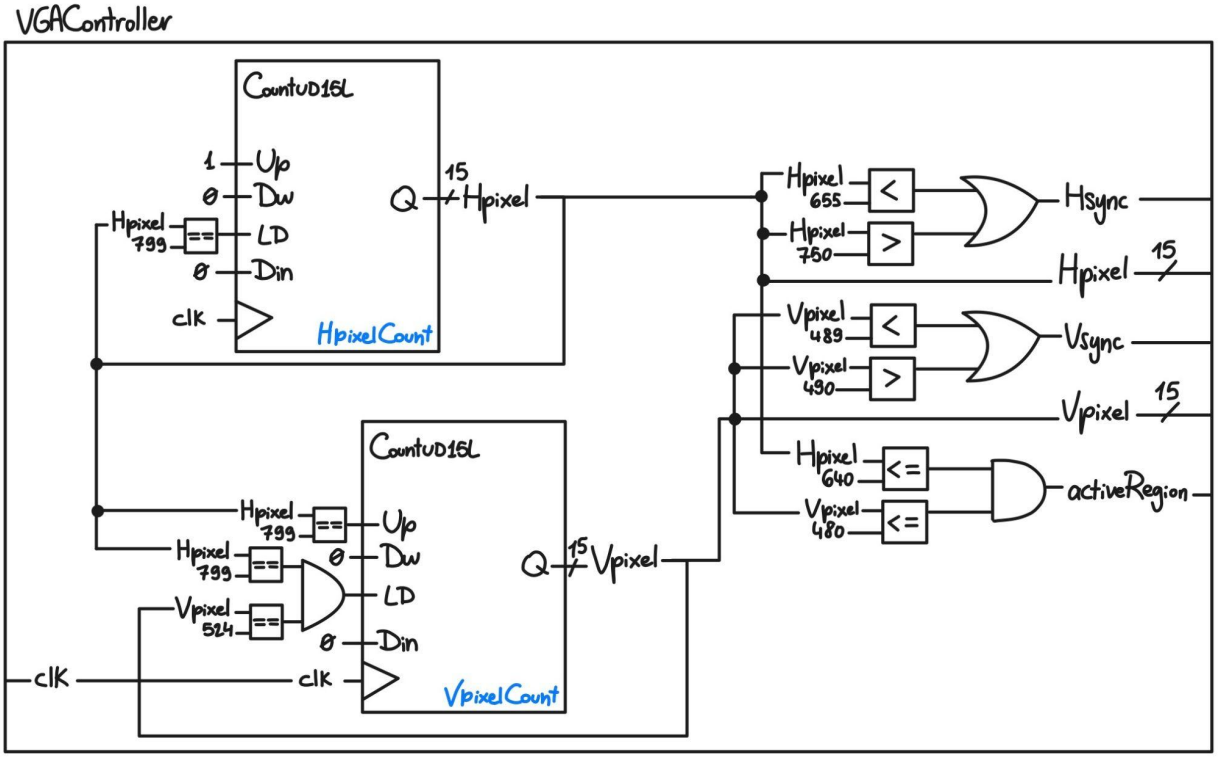

TopMod.v
labVGA_clks
clk
btnR
greset
digsel
clkin
not_So_Slow
Hsync
Vsync
VGAController
Hsync
Vsync
Hpixel
Vpixel
activeRegion
clk
myVGA
Background
Hpixel
Vpixel
btnU
btnC
clk
Caught
myBG
Slug
Bug
Water
Frame
Plat 1
Plat 2
Plat 3
Colors
Slug
Bug
Water
Frame
Plat1
Plat2
Plat3
activeRegion
myColors
vgaBlue
vgaRed
vgaGreen
countUD5L
Caught
Up
Din
Q
Score[4:0]
scoreCounter
RingCounter
digsel
Advance
sel
clk
myRing
Score
ring0
Selector
N
sel
H
selOut
mySelector
hex7seg
d
seg
myhex

VGAController

- Module Definition:
    - The code begins with the module definition statement module VGAController(...), which encapsulates the entire hardware module.
    - It has one input signal clk and several output signals: Hsync, Vsync, Hpixel, Vpixel, and activeRegion.
- Wires and Assignments:
    - The module declares two wire signals, Hactive and Vactive, which are used internally for determining the active display region.
    - The Hsync output signal is assigned based on the current Hpixel value. It becomes active (high) when the Hpixel value is less than 655 or greater than 750.
    - The Vsync output signal is assigned based on the current Vpixel value. It becomes active (high) when the Vpixel value is less than 489 or greater than 490.
    - The activeRegion output signal is assigned based on the current Hpixel and Vpixel values. It becomes active (high) when the Hpixel value is less than or equal to 640 and the Vpixel value is less than or equal to 480.
- CountUD15L Instantiations:
    - The code instantiates two instances of the CountUD15L module, named HpixelCount and VpixelCount, which are responsible for counting the horizontal and vertical pixel positions, respectively.
    - The HpixelCount module increments the Hpixel value on each clock cycle (clk) and resets it when it reaches 799 (indicating the end of a horizontal line).

- The VpixelCount module increments the Vpixel value when Hpixel reaches 799, and resets both Hpixel and Vpixel when they reach 799 and 524, respectively (indicating the end of a vertical frame).
- The VGAController module generates the necessary synchronization signals (Hsync and Vsync) for VGA display control based on the current horizontal and vertical pixel positions (Hpixel and Vpixel). It also determines the active display region (activeRegion) within the specified dimensions. The CountUD15L modules handle the counting and resetting of the pixel positions based on the clock signal (clk).

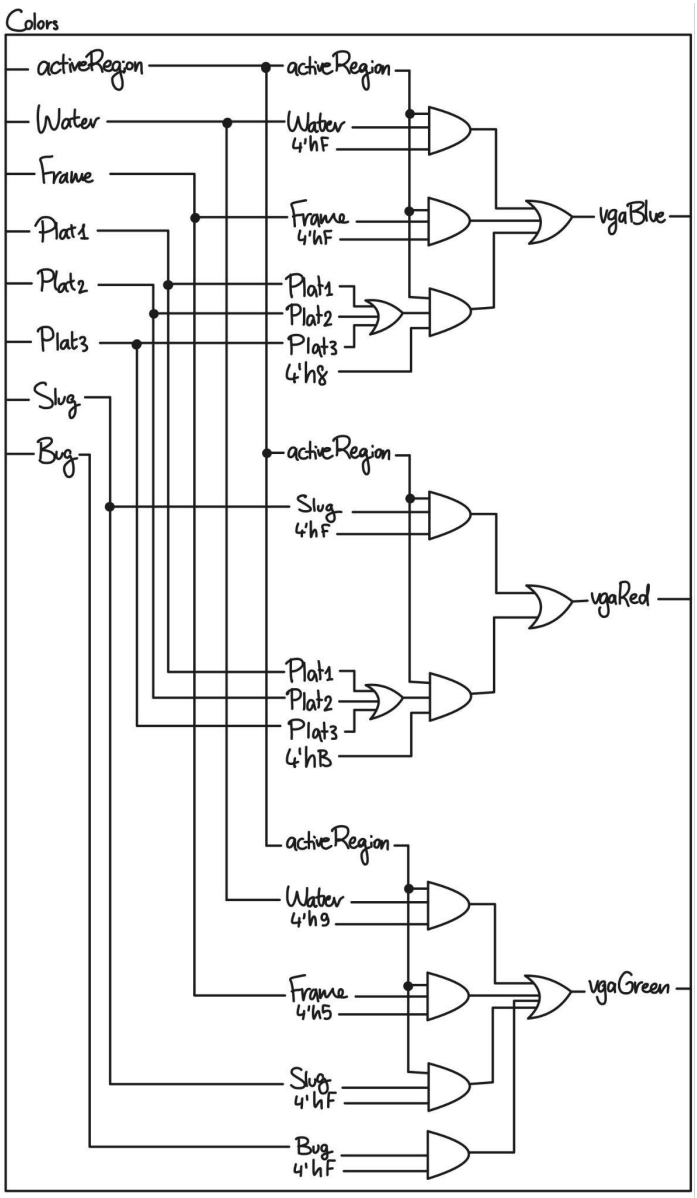
This code provides the foundational functionality to synchronize and control the VGA display, allowing for the proper rendering of graphics and images on a VGA monitor.



Colors

- vgaBlue Assignment:
    - The vgaBlue signal represents the blue component of the RGB color.
    - The assignment statement is a bitwise OR operation that combines several conditions to determine the value of vgaBlue.
    - The conditions are based on the activeRegion signal, which indicates whether the current pixel position is within the active display region.
    - Depending on the activeRegion value, different elements such as Water, Frame, Plat1, Plat2, Plat3, Slug, and Bug are considered.
    - The 4-bit hexadecimal values (4'hF, 4'h8, and 4'h0) represent specific blue color intensities.
    - The result of the bitwise OR operation determines the final value of vgaBlue.
- vgaRed Assignment:
    - The vgaRed signal represents the red component of the RGB color.
    - Similar to vgaBlue, the assignment statement combines multiple conditions using bitwise OR operations to determine the value of vgaRed.
    - The conditions involve the activeRegion signal and elements such as Water, Frame, Plat1, Plat2, Plat3, Slug, and Bug.
    - The 4-bit hexadecimal values (4'h0 and 4'hB) represent specific red color intensities.
    - The result of the bitwise OR operation determines the final value of vgaRed.
- vgaGreen Assignment:
    - The vgaGreen signal represents the green component of the RGB color.

- Similar to the previous assignments, the statement combines conditions using bitwise OR operations to determine the value of vgaGreen.
- The conditions involve the activeRegion signal and elements such as Water, Frame, Plat1, Plat2, Plat3, Slug, and Bug.
- The 4-bit hexadecimal values (4'h9 and 4'h5) represent specific green color intensities.
- The result of the bitwise OR operation determines the final value of vgaGreen.



Background

- Module Declaration: The code starts with the declaration of the module "Background" along with its input and output ports. The inputs include clk, Hpixel, Vpixel, btnU, and btnC, while the outputs include Water, Frame, Slug, Bug, Plat1, Plat2, Plat3, and Caught.
- Wire Declarations: Several wires are declared to facilitate the internal connections within the module. These wires are used to store intermediate values and connect different parts of the module together.
- Frame Confinements: The code assigns values to TopFrame, BottomFrame, LeftFrame, and RightFrame wires based on the specified frame confinements. These wires define the boundaries of the frame within the background.
- Water and Frame Assignment: The Water and Frame outputs are assigned based on the frame confinements. Water represents the water region within the background, while Frame represents the frame region.
- Platform Definitions: Three platforms (Plat1, Plat2, and Plat3) are defined based on the frame confinements and specific platform dimensions. These platforms are regions within the background where certain game elements can interact.
- Slug Collisions: The code defines conditions for the collision of a slug (game element) with the platforms. It checks for collisions with the left side of each platform (SlugLeftPlat), the top side of each platform (SlugTopPlat), and the bottom side of each platform (SlugBottomPlat).
- Slug and Bug States: The Slug and Bug outputs are assigned based on the presence of the slug and bug within specific regions of the background.

- Random Number Generation: The code includes a random number generator that generates a random value (Vrand) based on the provided clock signal.
- Frames Per Second (FPS): The code checks for specific pixel coordinates to determine the frames per second (FPS1 and FPS2) at which the background is being rendered.
- Flashing Logic: The code defines conditions for flashing effects within the background. It includes a flashing timer (flashing) and assigns values to slug_flash and bug_flash based on specific conditions.
- Counters: The code includes several counters (CountUD15L instances) that are used for various purposes such as counting the position of the slug (VSlugCounter and HSlugCounter), the position of the bug (HBugCounter), and the position of the platforms (Plat1VCounter, Plat1HCounter, Plat2VCounter, Plat2HCounter, Plat3VCounter, Plat3HCounter). These counters are driven by the clock signal and other control signals.
- Flip-Flops: The code includes several flip-flops (FDRE instances) that store and update different states within the module. These flip-flops include StartFF for the start state, MovingFF for the moving state, and SlugBugCollision for the collision between the slug and bug.
- Random Value Generation: The code includes another flip-flop (RDFF) that generates random values (RD and RQ) based on the clock signal and other control signals.
- Overall Functionality: The module combines all the defined inputs, outputs, wires, counters, and flip-flops to create

TopFrame : (0 <= Hpixel <= 639) & (0 <= Vpixel <= 7)

BottomFrame : (0 <= Hpixel <= 639) & (472 <= Vpixel <= 479)

LeftFrame : (0 <= Hpixel <= 7) & (8 <= Vpixel <= 471)

RightFrame : (632 <= Hpixel <= 639) & (8 <= Vpixel <= 471)

PlatTop = 201
PlatBottom = 208

flash_water = (VSlug + 17 == 375)

slug_border = (HSlug == 8)

Water : (8 <= Hpixel <= 631) & (375 <= Vpixel <= 471)

Frame : TopFrame | BottomFrame | LeftFrame | RightFrame

LeftPlat1 = Plat1H - Plat1V
LeftPlat2 = Plat2H - Plat2V
LeftPlat3 = Plat3H - Plat3V

Plat1 = ~Frame & (LeftPlat1 <= Hpixel <= Plat1H) & (200 <= Vpixel <= 207)
Plat2 = ~Frame & (LeftPlat2 <= Hpixel <= Plat2H) & (200 <= Vpixel <= 207)
Plat3 = ~Frame & (LeftPlat3 <= Hpixel <= Plat3H) & (200 <= Vpixel <= 207)

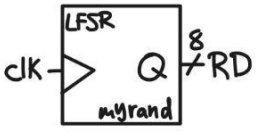
SlugLeftPlat = (LeftPlat1 <= HSlug + 17 <= LeftPlat1 + 17) or
(LeftPlat2 <= HSlug + 17 <= LeftPlat2 + 17) or
(LeftPlat3 <= HSlug + 17 <= LeftPlat3 + 17) or
(VSlug + 16 >= PlatTop) or
(VSlug + 1 <= PlatBottom)

SlugTopPlat = *see modules*

SlugBottomPlat = *see modules*

Slug = ~slug_flash & (HSlug <= Hpixel <= HSlug + 16) & (VSlug <= Vpixel <= VSlug + 16)

Bug = ~Frame & ~bug_flash & (HBug <= Hpixel <= HBug + 7) & (Vrand <= Vpixel <= Vrand + 8)



Vrand = RQ[5] & RQ[4:0] + 128 | ~RQ[5] & RQ[4:0] + 256

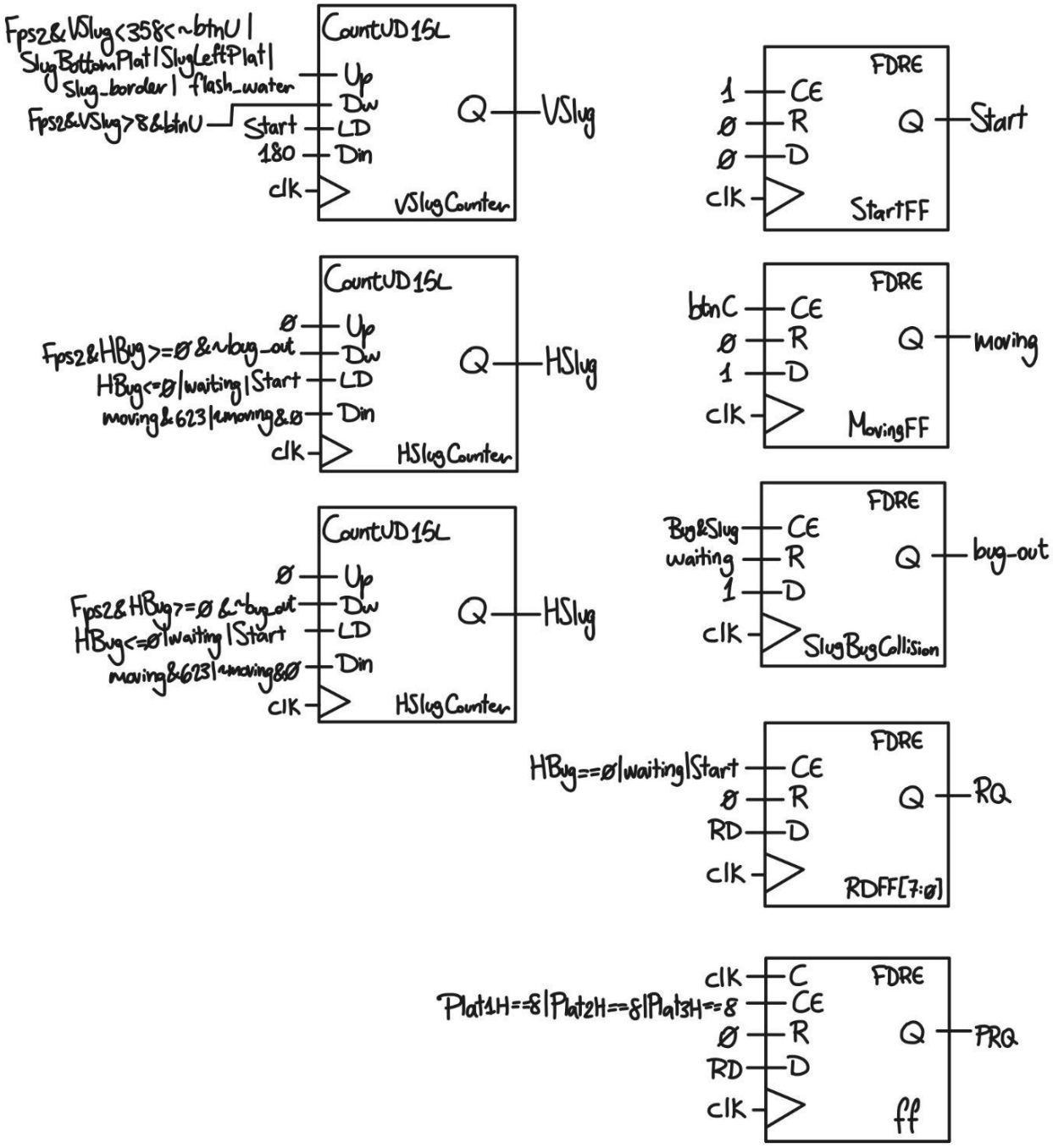
Fps1 = (Hpixel == 799) & (Vpixel == 524)
Fps2 = Fps1 | (Hpixel == 798) & (Vpixel == 424)

flashoff = (flashing<=8)

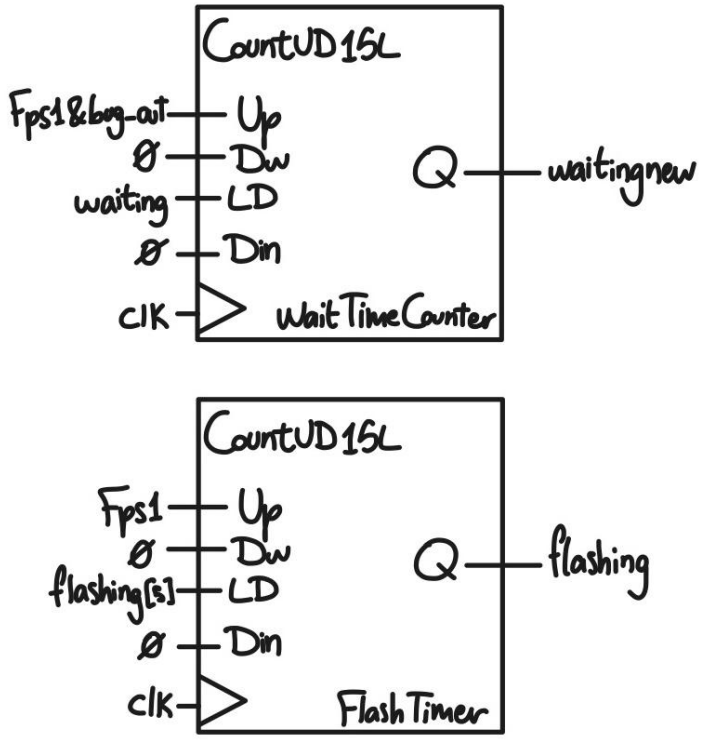
waiting = (waitingnew>=128)

slug_flash = (flash_water|slug_border|SlugLeftPlat)& flashoff

bug_flash = bug_out &(flashing<=8)



Caught = bug_out & waiting



Platform Counters : Plat1V , Plat1H , Plat2V , Plat2H , Plat3V , Plat3H

see module

Testing & Simulation

Testing and simulation were conducted to verify the functionality of the design before implementation. An incremental design approach was followed, implementing and testing the design in phases.

The components of the design, including the Hsync/Vsync signals, pixel address counters, and the VGA controller, were mapped out to understand their inputs, outputs, and functionalities.

A dedicated testbench was created to evaluate the design. It ensured the correctness of the Hsync and Vsync signals and validated that the RGB outputs were low outside the active region. This test was performed before adding further logic to the design to save simulation time.

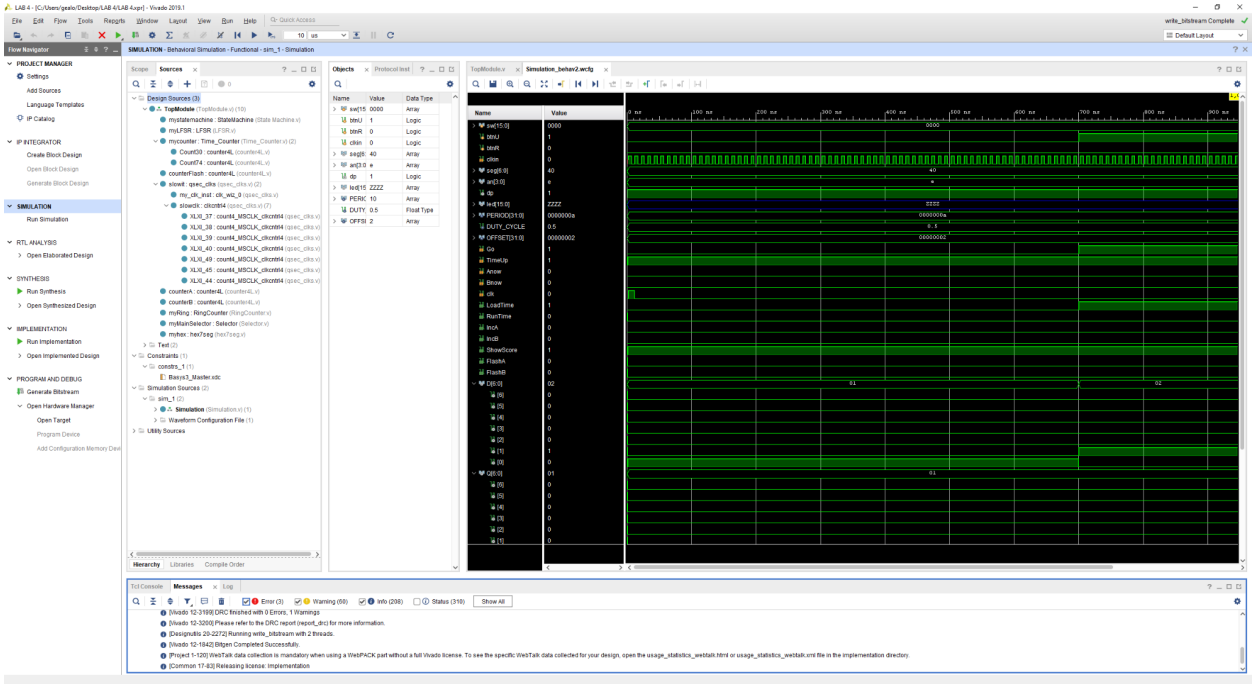
The implementation began with the VGA controller, responsible for generating Hsync and Vsync signals, providing the pixel address, and indicating the active region. The functionality of the VGA controller was verified using the testbench.

Subsequently, other components were implemented in phases. The borders and pond region were displayed, followed by the introduction of a platform initially positioned at a fixed location.

The platform was then programmed to move to the left, going off the screen completely, and respawning. The testbench was utilized to ensure the platform's correct behavior.

Additional components were gradually integrated into the design, with each implementation accompanied by testing using the testbench to ensure adherence to the desired specifications.

Throughout the testing and simulation process, attention was given to timing and synchronization, ensuring the accurate generation of Hsync, Vsync, and RGB signals. Monitor feedback was utilized to observe the behavior of the design, and LEDs and 7-segment displays were employed for displaying relevant signals and values to aid in debugging.



Conclusion:

In conclusion, the testing and simulation phase played a crucial role in verifying the functionality and correctness of the design before its implementation. An incremental design approach was followed, allowing for system integration and testing of different components.

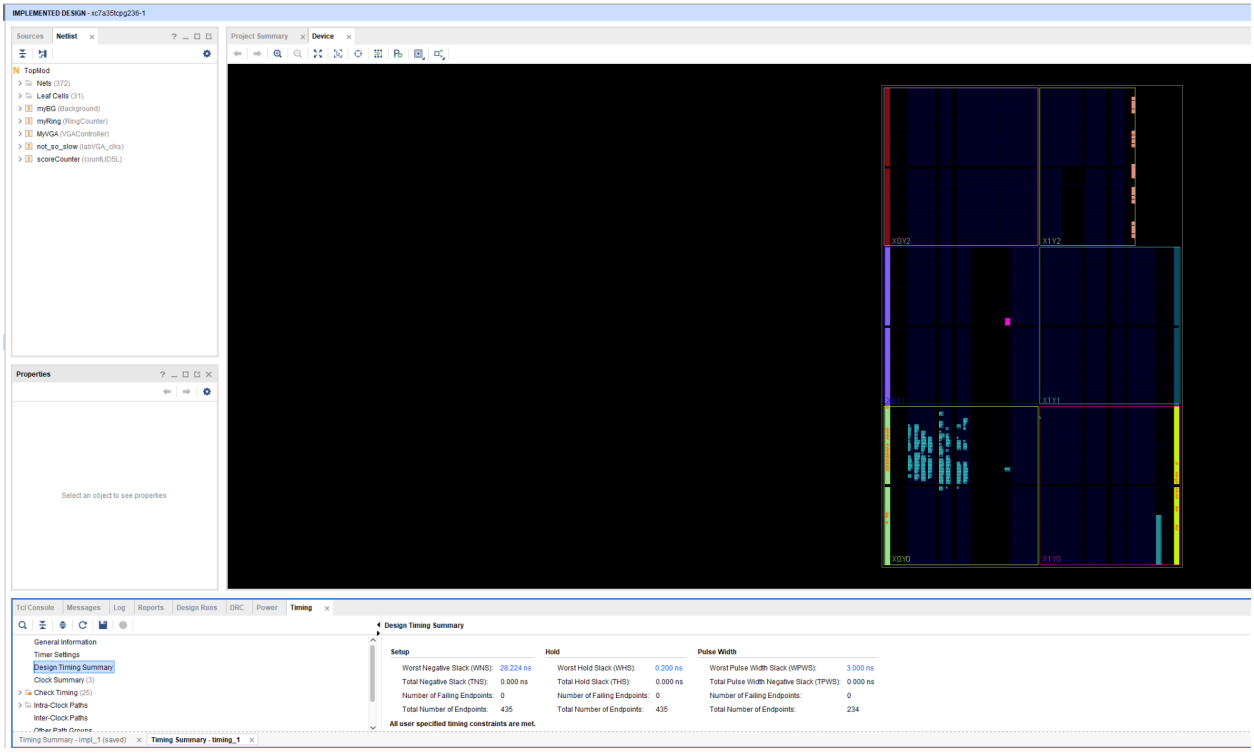
By mapping out the inputs, outputs, and functionalities of each component, a clear understanding was gained of their roles within the overall design. This facilitated the identification and resolution of any potential issues or conflicts early on.

The dedicated testbench proved invaluable in validating the behavior of critical elements, such as the Hsync/Vsync signals and RGB outputs. By conducting these tests before incorporating additional logic, simulation time was optimized, enabling more efficient development.

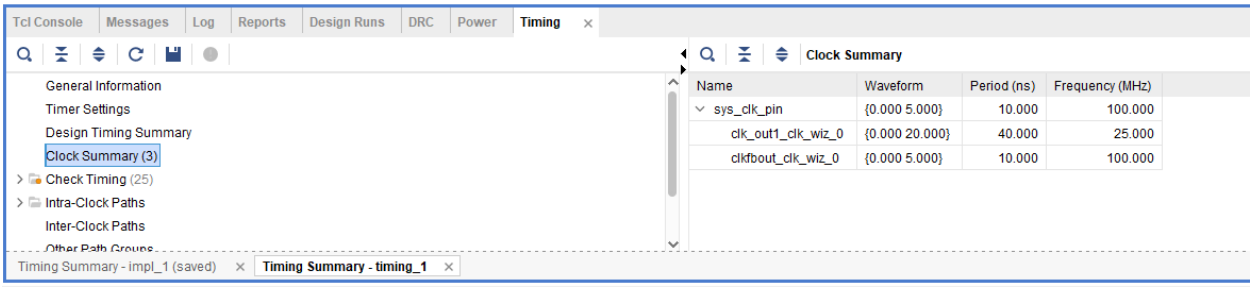
Throughout the process, meticulous attention was given to timing and synchronization to ensure the accurate generation of signals. Monitor feedback served as a valuable tool for monitoring and

assessing the design's behavior, while LEDs and 7-segment displays aided in debugging by providing visual representations of key signals and values.

The successful completion of the testing and simulation phase instilled confidence in the design's functionality and laid a solid foundation for its subsequent implementation. Any issues or discrepancies that arose during testing were identified and addressed promptly, contributing to an overall robust and reliable design. By adhering to best practices and employing thorough testing methodologies, the design is well-positioned for the next phase of development. The insights gained from testing and simulation will inform the subsequent steps, ensuring a smooth transition from design to implementation and ultimately leading to a successful end product.



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


Tcl Console
Messages
Log
Reports
Design Runs
DRC
Power
Timing
General Information
Timer Settings
Design Timing Summary
Clock Summary (3)
Check Timing (25)
Intra-Clock Paths
Inter-Clock Paths
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
Timing Summary - impl_1 (saved)
Timing Summary - timing_1

Appendix:

Top Mod:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/18/2023 02:15:15 PM
// Design Name:
// Module Name: TopMod
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module TopMod(

    input btnU,
    input btnC,
    input btnR,
    input clkin,

    output dp,
    output [3:0] an,
    output [6:0] seg,
    output [3:0] vgaRed,
    output [3:0] vgaBlue,
    output [3:0] vgaGreen,
    output Hsync,
    output Vsync

    );

    wire clk;
    wire digsel;
    wire Water;
    wire Frame;
    wire Platform;
    wire Slug;
    wire Bug;
    wire [14:0] Hpixel;
    wire [14:0] Vpixel;
    wire [14:0] HSlug;
    wire [14:0] VSlug;
    wire [14:0] HBug;
    wire activeRegion;
    wire Plat1, Plat2, Plat3;
    wire Caught;
    wire [3:0] ringO, selOut;
    wire [14:0] Score;
    wire bug_out;
    wire moving;
    wire Start;
    wire SlugLeftPlat;
    wire SlugBottomPlat;
    wire SlugTopPlat;

    assign an[3:2] =  2'b11;
    assign an[1:0] =  ~ringO[1:0];

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

    VGAController MyVGA (
        .clk(clk),
        .Hsync(Hsync),
        .Vsync(Vsync),
        .Hpixel(Hpixel),
        .Vpixel(Vpixel),
        .activeRegion(activeRegion)
        );

    Background myBG (
        .clk(clk),
        .btnU(btnU),
        .btnC (btnC),
        .Hpixel(Hpixel),
        .Vpixel(Vpixel),
        .Water(Water),
        .Frame(Frame),
        .Slug(Slug),
        .Bug(Bug),
        .Plat1(Plat1),
        .Plat2(Plat2),
        .Plat3(Plat3),
        .Caught(Caught)
        );

    countUD5L scoreCounter(
        .clk(clk),
        .Up(Caught),
        .Din(Score[4:0]),
        .Q(Score[4:0])
    );

    //Ring Counter logic
    RingCounter myRing (.Advance(digsel), .clk(clk), .sel(ringO));
    Selector myMainSelector(.sel(ringO), .N(Score), .H(selOut));
    hex7seg myhex (.d(selOut), .seg(seg));

    //Colors
    assign vgaBlue = (({4{activeRegion}} & ({4{Water}} & 4'hF)) | ({4{activeRegion}} & ({4{Frame}} & 4'hF)) | ({4{activeRegion}} & ({4{Plat1|Plat2|Plat3}} & 4'h8)) | ({4{activeRegion}} & ({4{Slug}} & 4'h0)) | ({4{activeRegion}} & ({4{Bug}} & 4'h0)));
    assign vgaRed = (({4{activeRegion}} & ({4{Water}} & 4'h0)) | ({4{activeRegion}} & ({4{Frame}} & 4'h0)) | ({4{activeRegion}} & ({4{Plat1|Plat2|Plat3}} & 4'h0)) | ({4{activeRegion}} & ({4{Slug}} & 4'hF)) | ({4{activeRegion}} & ({4{Bug}} & 4'h0)));
    assign vgaGreen = (({4{activeRegion}} & ({4{Water}}&4'h9)) | ({4{activeRegion}} & ({4{Frame}} & 4'h5)) | ({4{activeRegion}} & ({4{Plat1|Plat2|Plat3}} & 4'h0)) | ({4{activeRegion}} & ({4{Slug}} & 4'hF)) | ({4{activeRegion}} & ({4{Bug}} & 4'hF)));

endmodule
```

VGAController:

```
                                                `timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/30/2023 09:43:47 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] Hpixel,
    output [14:0] Vpixel,
    output activeRegion
    );

    wire Hactive;
    wire Vactive;

    assign Hsync = (Hpixel < 15'd655) | (Hpixel > 15'd750);
    assign Vsync = (Vpixel < 15'd489) | (Vpixel > 15'd490);
    assign activeRegion = ((Hpixel <= 15'd640) & (Vpixel <= 15'd480));

    CountUD15L HpixelCount(
        .Up(1'b1),
        .Dw(1'b0),
        .LD(Hpixel==15'd799), //Reset value
        .clk(clk),
        .Din({15{1'b0}}),
        .Q(Hpixel)
    );

    CountUD15L VpixelCount(
        .Up(Hpixel==15'd799), //Reset value
        .Dw(1'b0),
        .LD((Hpixel == 15'd799) & (Vpixel == 15'd524)), //Reset values
        .clk(clk),
        .Din({15{1'b0}}),
        .Q(Vpixel)
   );

endmodule
```

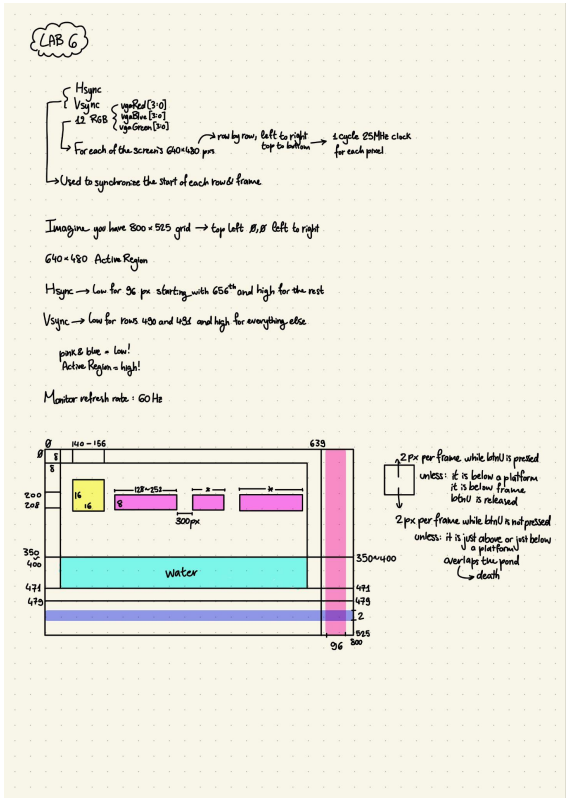
Background:

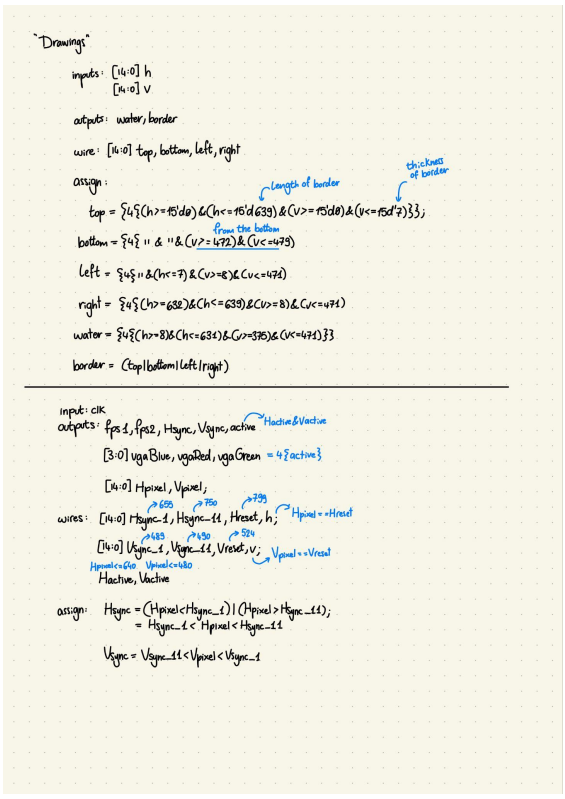
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/31/2023 05:53:34 PM
// Design Name:
// Module Name: Background
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module Background(

    input clk,
    input [14:0] Hpixel,
    input [14:0] Vpixel,
    input btnU,
    input btnC,

    output Water,
    output Frame,
    output Slug,
    output Bug,
    output Plat1,
    output Plat2,
    output Plat3,
    output Caught
    );

    wire [14:0] TopFrame;
    wire [14:0] BottomFrame;
    wire [14:0] LeftFrame;
    wire [14:0] RightFrame;

    wire [7:0] RD;
    wire [7:0] RQ;
    wire [7:0] PRQ;
    wire [14:0] Vrand;

    wire [14:0] VSlug;
    wire [14:0] HSlug;
    wire [14:0] HBug;

    wire FPS1;
    wire FPS2;

    wire bug_out;

    wire [14:0] PlatTop;
    wire [14:0] PlatBottom;

    wire Start;
    wire Playing;
    wire [14:0] Plat1V, Plat1H, Plat2V, Plat2H, Plat3V, Plat3H;
    wire [14:0] LeftPlat1, LeftPlat2, LeftPlat3;

    wire SlugLeftPlat;
    wire SlugTopPlat;
    wire SlugBottomPlat;

    wire moving;
    wire waiting;
    wire [14:0] flashing;
    wire slug_flash, bug_flash;

    wire slug_border;
    wire flash_water;
    wire [7:0] waitingnew;

    //Assigning the confinements of the frame
    assign TopFrame = {4{(Hpixel>=15'd0) & (Hpixel<=15'd639) & (Vpixel>=15'd0) & (Vpixel<=15'd7)}};
    assign BottomFrame = {4{(Hpixel>=15'd0) & (Hpixel<=15'd639) & (Vpixel>=15'd472) & (Vpixel<=15'd479)}};
    assign LeftFrame = {4{(Hpixel>=15'd0) & (Hpixel<=15'd7) & (Vpixel>=15'd8) & (Vpixel<=15'd471)}};
    assign RightFrame = {4{(Hpixel>=15'd632) & (Hpixel<=15'd639) & (Vpixel>=15'd8) & (Vpixel<=15'd471)}};
```

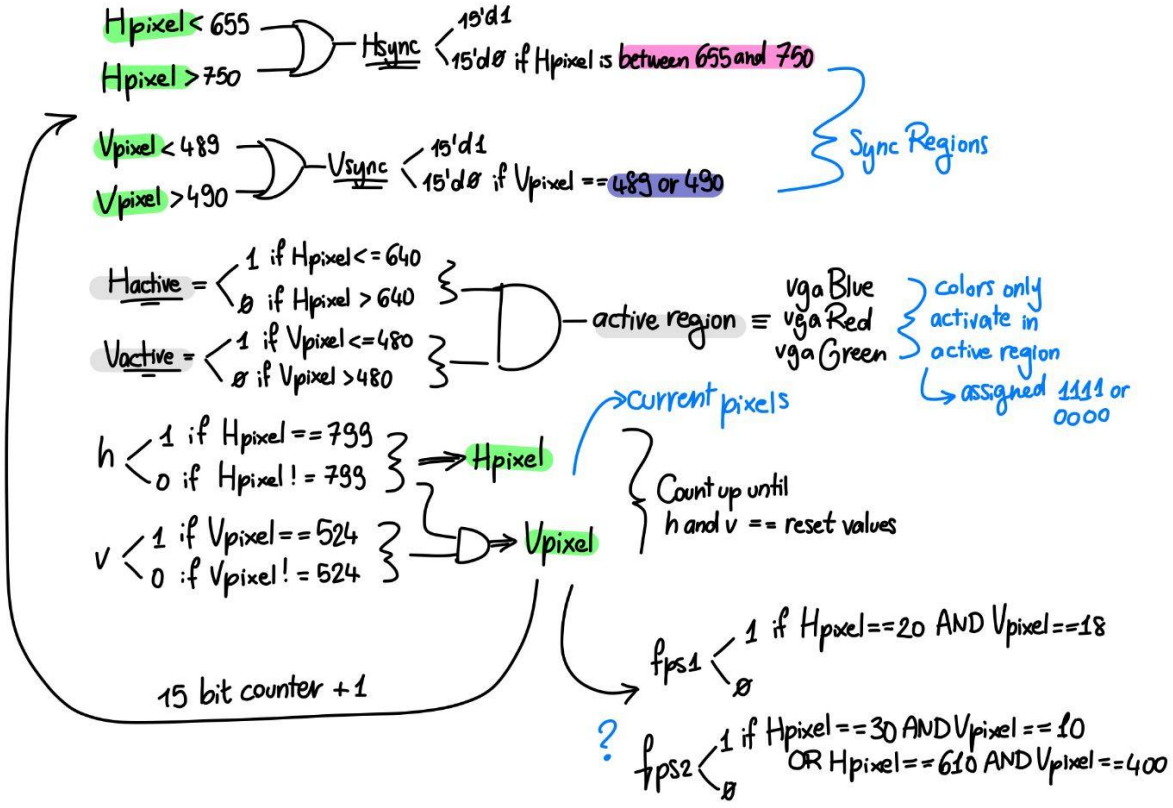
```
//Assigning Water and Frame: Background
assign Water = {4{(Hpixel>=15'd8) & (Hpixel<=15'd631) & (Vpixel>=15'd375) & (Vpixel<=15'd471)}};
assign Frame = (TopFrame | BottomFrame | LeftFrame | RightFrame);

//Assigning three platforms for the game
assign LeftPlat1 = Plat1H - Plat1V;
assign LeftPlat2 = Plat2H - Plat2V;
assign LeftPlat3 = Plat3H - Plat3V;

//Defining the platforms
assign Plat1 = ~Frame&({4{(Hpixel >= (LeftPlat1)) & (Hpixel <= Plat1H) & (Vpixel >= 15'd200) & (Vpixel <= 15'd207)}});
assign Plat2 = ~Frame&({4{(Hpixel >= (LeftPlat2)) & (Hpixel <= Plat2H) & (Vpixel >= 15'd200) & (Vpixel <= 15'd207)}});
assign Plat3 = ~Frame&({4{(Hpixel >= (LeftPlat3)) & (Hpixel <= Plat3H) & (Vpixel >= 15'd200) & (Vpixel <= 15'd207)}});

//Slug's collision with Left side of Platform
assign SlugLeftPlat = (((
    ((HSlug + 15'd17) >= (LeftPlat1))
    & (HSlug + 15'd17) <= ((LeftPlat1) + 15'd17)
    | ((HSlug + 15'd17) >= (LeftPlat2))
    & (HSlug + 15'd17) <= ((LeftPlat2) + 15'd17)
    | ((HSlug + 15'd17) >= (LeftPlat3))
    & (HSlug + 15'd17) <= ((LeftPlat3) + 15'd17))
    & ((VSlug + 15'd1 <= PlatBottom)
    & (VSlug + 15'd16 >= PlatTop)
    )));

assign PlatTop = 15'd201;
assign PlatBottom = 15'd208;
assign flash_water = VSlug + 15'd17 ==  15'd375;
assign slug_border = HSlug == 15'd8;

//Slug's collision with Top side of Platform
assign SlugTopPlat = (VSlug + 15'd17 == PlatTop)
     & ((LeftPlat1 < 15'd140 + 15'd16 & Plat1H > 15'd140 - 15'd2)
     | (LeftPlat2 < 15'd140 + 15'd16 & Plat2H > 15'd140 - 15'd2)
     | (LeftPlat3 < 15'd140 + 15'd16 & Plat3H > 15'd140 - 15'd2));

//Slug's collision with Bottom side of Platform
assign SlugBottomPlat =
    (VSlug == PlatBottom)
    & ((LeftPlat1 < 15'd140 + 15'd16 & Plat1H > 15'd140 - 15'd2)
    | (LeftPlat2 < 15'd140 + 15'd16 & Plat2H > 15'd140 - 15'd2)
    | (LeftPlat3 < 15'd140 + 15'd16 & Plat3H > 15'd140 - 15'd2));

//Slug and Bug States
assign Slug = ~slug_flash & {4{(Hpixel >= HSlug) & (Hpixel <= HSlug + 15'd16) & (Vpixel >= VSlug) & (Vpixel <= (VSlug + 15'd16))}};
assign Bug = ~Frame & ~bug_flash & {4{(Hpixel >= HBug) & (Hpixel <= HBug + 15'd7) & (Vpixel >= Vrand) & (Vpixel <= Vrand + 15'd8)}};

//Random number generator
LFSR myVrand (.clk(clk), .Q(RD));
assign Vrand = ({15{RQ[5]}}&(RQ[4:0]+15'd128)) | ({15{~RQ[5]}}&(RQ[4:0]+15'd256)) ;

//Frames Per Second
assign FPS1 = Hpixel == 15'd799 & Vpixel == 15'd524;
assign FPS2 = FPS1 | Hpixel == 15'd798 & Vpixel == 15'd424;

//Flashing Logic
wire flashoff;
assign flashoff = flashing <= 15'd8;
assign waiting = (waitingnew >= 15'd128);
assign slug_flash = (flash_water | slug_border | SlugLeftPlat) & flashoff;
assign bug_flash = bug_out & ((flashing <= 15'd8));

//Slug Up and Down Counter
CountUD15L VSlugCounter(
    .Up((FPS2 & VSlug < 15'd358 & ~btnU) | SlugBottomPlat | SlugLeftPlat | slug_border | flash_water),
    .Dw((FPS2 & VSlug > 15'd8 & btnU) | SlugTopPlat | SlugBottomPlat | SlugLeftPlat | slug_border | flash_water),
    .LD(Start), //Reset value
    .clk(clk),
    .Din(15'd180), //Starting Value
    .Q(VSlug)
);

//Slug Horizontal Counter
CountUD15L HSlugCounter(
    .clk(clk),
    .Up(1'b0),
    .Dw((FPS1 & HSlug > 15'd8) & SlugLeftPlat),
    .LD(Start),
    .Din(15'd140),
    .Q(HSlug)
);
```

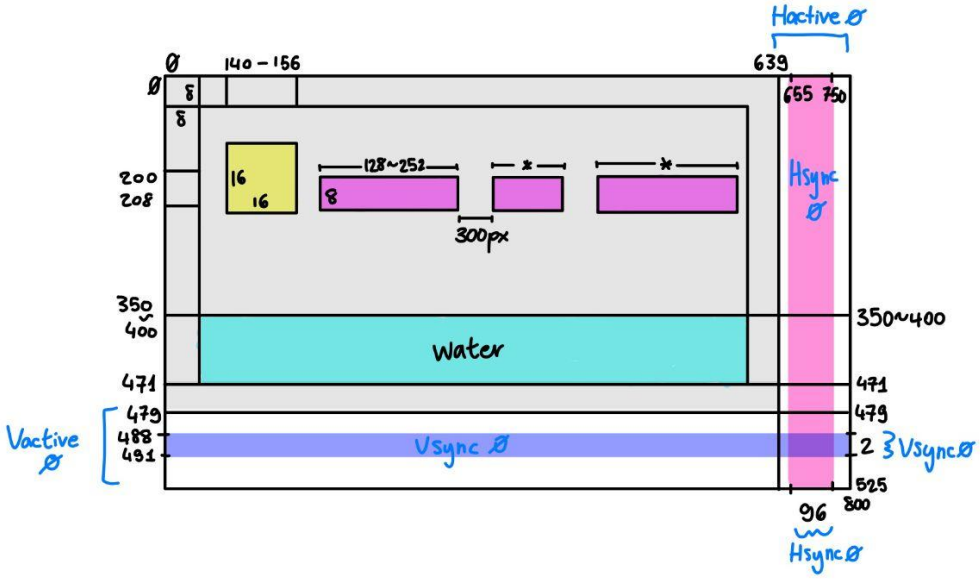
```
    //Bug Horizontal Counter
    CountUD15L HBugCounter(
        .Up(1'b0),
        .Dw((FPS2 & HBug >= 15'd0) & ~bug_out),
        .LD(HBug<=15'd0 | waiting | Start), //Reset value
        .clk(clk),
        .Din({{15{moving}} & 15'd623) | ({15{~moving}} & 15'h0)), //Starting Value
        .Q(HBug)
    );

    //My FlipFlop values for each State
    FDRE #(.INIT(1'b1)) StartFF (.C(clk), .CE(1'b1), .R(1'b0), .D(1'b0), .Q(Start));
    FDRE #(.INIT(1'b0)) MovingFF (.C(clk), .R(1'b0), .CE(btnC), .D(1'b1), .Q(moving));
    FDRE #(.INIT(1'b0)) SlugBugCollision (.C(clk), .R(waiting), .CE(Bug & Slug), .D(1'b1), .Q(bug_out));

    //Generating random value for Bug
    FDRE #(.INIT(1'b0)) RDFF [7:0] (.C({8{clk}}), .CE(HBug==15'd0 | waiting | Start), .R(8'b0), .D(RD), .Q(RQ));

    //Random value for Platforms
        FDRE #(.INIT(1'b0)) ff [7:0](
        .C({8{clk}}),
        .R({8{1'b0}}),
        .CE(Plat1H == 15'd8 | Plat2H == 15'd8 | Plat3H == 15'd8),
        .D(RD[7:0]),
        .Q(PRQ[7:0])
    );

    //One frame counter
    CountUD15L WaitTimeCounter (.clk(clk), .Up(FPS1 & bug_out), .Dw(1'b0), .LD(waiting), .Din(15'b0), .Q(waitingnew));

    //Keeping track of Caught for Scoring
    assign Caught = bug_out & waiting;

    //My counter for the Slug Flashing
    CountUD15L FlashTimer (.clk(clk), .Up(FPS1), .Dw(1'b0), .LD(flashing[5]), .Din(15'b0), .Q(flashing));

    //My Counters for the 3 Platforms
    CountUD15L Plat1VCounter(.Up(1'b0), .Dw((FPS1 & (LeftPlat1 <= 15'd7) & moving & ~slug_border) & ~bug_out), .LD(Plat1V == 0 | Start), .clk(clk), .Din({{15{~Start}} & {1'b1, PRQ[4:0], 2'b00}) | ({15{Start}} & 15'd200)), .Q(Plat1V));
    CountUD15L Plat1HCounter(.Up(1'b0), .Dw((FPS1 & moving & ~slug_border) & ~bug_out), .LD(Plat1H <= 15'd7 | Start), .clk(clk), .Din({{15{~Start}} & 15'd950) | ({15{Start}} & 15'd625)), .Q(Plat1H));
    CountUD15L Plat2VCounter(.Up(1'b0), .Dw((FPS1 & (LeftPlat2 <= 15'd7) & moving & ~slug_border) & ~bug_out), .LD(Plat2V == 0 | Start), .clk(clk), .Din({{15{~Start}} & {1'b1, PRQ[5:1], 2'b00}) | ({15{Start}} & 15'd200)), .Q(Plat2V));
    CountUD15L Plat2HCounter(.Up(1'b0), .Dw((FPS1 & moving & ~slug_border) & ~bug_out), .LD(Plat2H <= 15'd7 | Start), .clk(clk), .Din({{15{~Start}} & 15'd950) | ({15{Start}} & 15'd325)), .Q(Plat2H));
    CountUD15L Plat3VCounter(.Up(1'b0), .Dw((FPS1 & (LeftPlat3 <= 15'd7) & moving & ~slug_border) & ~bug_out), .LD(Plat3V == 0 | Start), .clk(clk), .Din({{15{~Start}} & {1'b1, PRQ[6:2], 2'b00}) | ({15{Start}} & 15'd200)), .Q(Plat3V));
    CountUD15L Plat3HCounter(.Up(1'b0), .Dw((FPS1 & moving & ~slug_border) & ~bug_out), .LD(Plat3H <= 15'd7 | Start), .clk(clk), .Din({{15{~Start}} & 15'd950) | ({15{Start}} & 15'd25)), .Q(Plat3H));

endmodule
```

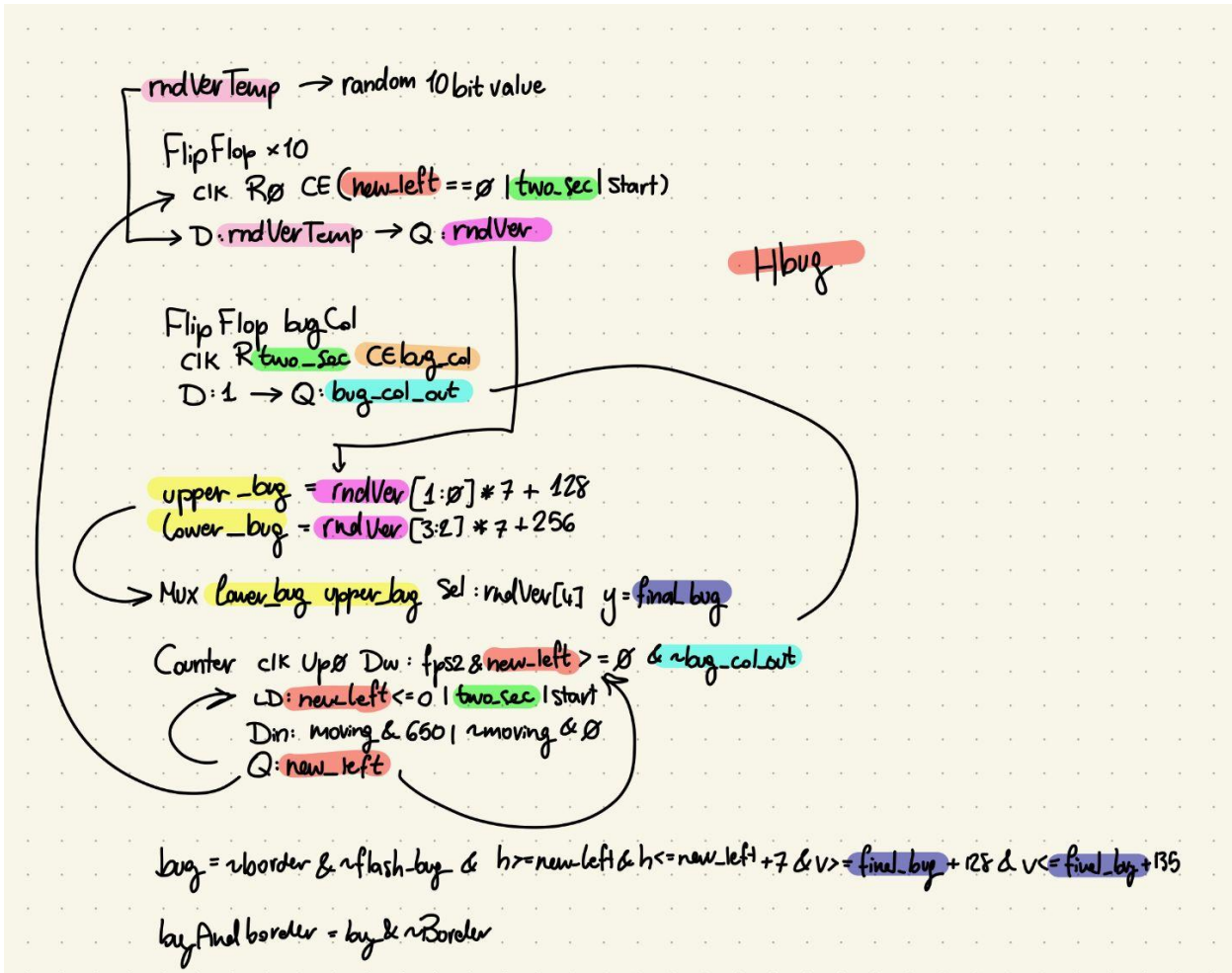
Lab Notebook:






Hpixel < 655
Hpixel > 750
Hsync
15'd1
15'd0 if Hpixel is between 655 and 750
Vpixel < 489
Vpixel > 490
Vsync
15'd1
15'd0 if Vpixel == 489 or 490
Sync Regions
Hactive = 1 if Hpixel <= 640, 0 if Hpixel > 640
Vactive = 1 if Vpixel <= 480, 0 if Vpixel > 480
active region ≡ vga Blue, vga Red, vga Green
colors only activate in active region
assigned 1111 or 0000
h 1 if Hpixel == 799, 0 if Hpixel != 799
Hpixel
current pixels
v 1 if Vpixel == 524, 0 if Vpixel != 524
Vpixel
Count up until h and v == reset values
15 bit counter +1
fps1 1 if Hpixel == 20 AND Vpixel == 18
fps2 1 if Hpixel == 30 AND Vpixel == 10 OR Hpixel == 610 AND Vpixel == 400


Hactive 0
0
140 - 156
639
655 750
8
16
16
128~252
8
300px
200
208
Hsync 0
350
400
350~400
water
471
471
Vactive 0
479
488
491
Vsync 0
479
2
3 Vsync 0
525
800
96
Hsync 0


rndVerTemp → random 10 bit value
Flip Flop ×10
clk R0 CE (new_left == 0 | two_sec | start)
D: rndVerTemp → Q: rndVer
Hbug
Flip Flop bug_Col
clk R two_sec CE bug_col
D: 1 → Q: bug_col_out
upper_bug = rndVer [1:0] * 7 + 128
lower_bug = rndVer [3:2] * 7 + 256
Mux lower_bug upper_bug Sel: rndVer[4] y = final_bug
Counter clk Up0 Dw: fps2 & new_left >= 0 & ~bug_col_out
LD: new_left <= 0 | two_sec | start
Din: moving & 650 | ~moving & 0
Q: new_left
bug = ~border & ~flash_bug & h >= new_left & h <= new_left + 7 & v >= final_bug + 128 & v <= final_bug + 135
bug_final border = bug & ~Border