Project Progress Report

Flappy Bird

Ali Asghar Yousuf - ay06993
Shahid Mehmood - mm06600
Azeem Haider - mh06858
Musab Sattar - ms06651

November 2021

Digital Logic and Design
Supervisors: Junaid Ahmed Memon - Hafsa Amanullah

Contents

1 Introduction

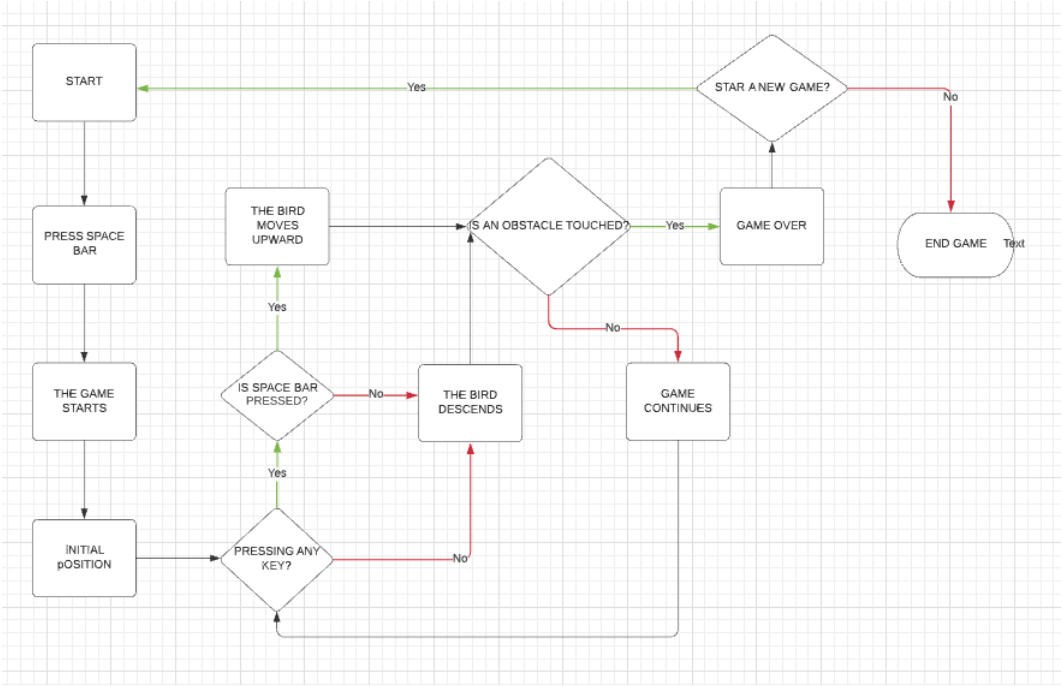
Flappy bird exploded in popularity when it was first introduced in the apple Appstore and the google play store. The game quickly was rose to popularity and in the short span of time became a fan favorite game that most wanted to play in their spare time. This crazy popularity and boost prompted a lot of market competitors to enter the market to try and replicate the game and somehow mimic the popularity and monetary gain that the developers of flappy bird experienced. This could not happen and the mimickers and fake developers of the game could never copy the success. The huge success of the game meant, it was soon taken out of the play store and the app store sent behind a pay wall that no one appreciates. The game was taken away from us, the childhood game that made us want to beat the high scores of our peers. Therefore, years later, the group has decided to make our very own flappy bird and compete.

The project is simple and subtle in its delivery physically, but emotionally it has a pretty big impact. The bird on the x co-ordinate will not move, however, on the y-axis the bird will be moving up and down to navigate between the tubes. The game will be played with the help of the input which will be our keyboard, the bird will jump upward in the sky when the spacebar is pressed, while the bird will slowly but surely descend to the ground. The user has to make sure that he/she masterfully navigates the bird to glory by missing the tubes and creating a high score that no one else can beat. The project will be a great trip down the memory lane for the users that have played and enjoyed this game in the past but this in no way means that new users or users that haven't played this game before will not enjoy this. The target user is anyone and everyone having a liking for casual games but they still have the competitive attitude and desire to do great.

2 User-Flow Diagram and Division of tasks between blocks

2.1 User-Flow diagram

The following is the user flow diagram of our project:



2.2 Explanation:

The user-flow diagram is elegantly made indicating all the possibilities for a user in this game. The user is controlling one key, the space bar. Extremely essential for the success of the user to properly navigate the bird and not hit the tubes while controlling the bird. The user can either move up by pressing space bar or descend downwards by not pressing the key for a small time.The tubes will keep randomly generating until the game is over and the user crashes the bird with a tube. The user will be granted ten points each time it accurately navigates through one set of tubes.

2.3 Division of Task:

The task division between the three blocks of our project are as followed:

1. The tasks of the input block of our project is when we press space bar to provide the control block with an input/task for it to perform.

2. The task of the control block of our project is to check the entry of the input and make sure the output is being performed. In simple words, the bird moving upward and downward is a control block but since it also a function of the input we are providing, it can also be understood to be an output block.

3. The task of the output block of our project is that when the space bar is pressed, the flappy bird will ascend(move up) for a unit time and then descend(move down) until the space bar is pressed again. If the bird collides with any of the walls of the tubes or touches the upper or lower frame of the screen the game finishes and output is generated as 'GAME OVER'

3 Input Block

3.1 Introduction to the input peripheral: The Keyboard

The Artix 7 FPGA board is powered by connecting it to the pc via micro USB. After turning the switch on we are required to configure the board. Bit streams generated (in .bit files) for the configuration using Vivado software. We program them using the hardware language, Verilog.
The keyboard is attached to the FPGA board at the USB connector J2. The host capability of USB HID in Basys 3 is provided by microcontroller which switches to the USB host and turns on application mode

3.2 Inputs in the Project

In order to move our bird, we press 'spacebar'. Pressing the space bar moves the bird upwards for an instance after which it continues its downwards trajectory. In order to exit/reset the game, we press 'x'. Pressing the reset button (x) resets the game to its idle state regardless of the game progress.

3.3 Configuring the keyboard

We have interfaced the USB keyboard to give scan codes of PS/2 keyboard as the FPGA only takes PS/2 keyboard inputs. When we press the space bar on the keyboard, the unique scan code is generated and the game starts or continues meaning if it's the initial press then the game starts but if it is not the first press then we are already in the game and the bird is navigated by using these presses. Similarly we have an exit key 'x'. This is a reset

button which resets the game and goes back to the idle state no matter where the player currently is. Pressing x also generates a unique hexadecimal scan code that we interpret similar to the space bar hexadecimal scan code.

Scan codes:
SPACE BAR = 29
X= 22

3.4 Input translation to Output

The bird moves up when the space bar is pressed. The clock of the keyboard sends the data on the rising edges. An important thing to note here is that the space bar's functionality will be designed such that at every press the bird jumps up rather than just escalating up with a constant direction and angle

3.5 Input Code

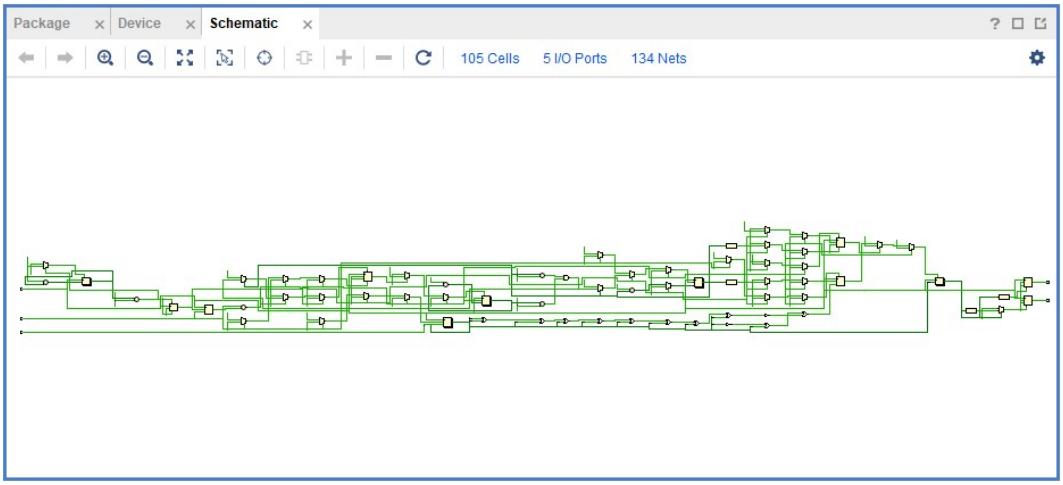
```
module Keyboard(
        input CLK,      //board clock
    input PS2_CLK,      //keyboard clock and data signals
    input PS2_DATA,
    output reg Jump, //output when spacebar is pressed
    output reg Reset //output when X is pressed
    );

        wire [7:0] SPACE_BAR = 8'h29;
        wire [7:0] X = 8'h22;
|
        reg read;                           //this is 1 if still waits to receive more bits
        reg [11:0] count_reading;           //this is used to detect how much time passed since it received the previous codeword
        reg PREVIOUS_STATE;                 //used to check the previous state of the keyboard clock signal to know if it changed
        reg scan_err;                       //this becomes one if an error was received somewhere in the packet
        reg [10:0] scan_code;               //this stores 11 received bits
        reg [7:0] CODEWORD;                 //this stores only the DATA codeword
        reg TRIG_ARR;                       //this is triggered when full 11 bits are received
        reg [3:0]COUNT;                     //tells how many bits were received until now (from 0 to 11)
        reg TRIGGER = 0;                    //This acts as a 250 times slower than the board clock.
        reg [7:0]DOWNCOUNTER = 0;           //This is used together with TRIGGER - look the code

        //Set initial values
        initial begin
                PREVIOUS_STATE = 1;
                scan_err = 0;
                scan_code = 0;
                COUNT = 0;
                CODEWORD = 0;
                Jump = 0;
                Reset = 0;
                read = 0;
                count_reading = 0;
        end

        always @(posedge CLK) begin                     //This reduces the frequency 250 times
                if (DOWNCOUNTER < 249) begin            //and uses variable TRIGGER as the new board clock
                        DOWNCOUNTER <= DOWNCOUNTER + 1;
                        TRIGGER <= 0;
                end
                else begin
                        DOWNCOUNTER <= 0;
                        TRIGGER <= 1;
                end
        end

        always @(posedge CLK) begin
                if (TRIGGER) begin
                        if (read)                       //if it still waits to read full packet of 11 bits, then (read == 1)
```

```
	always @(posedge CLK) begin
		if (TRIGGER) begin
			if (read)                                  //if it still waits to read full packet of 11 bits, then (read == 1)
				count_reading <= count_reading + 1;     //and it counts up this variable
			else                                       //and later if check to see how big this value is.
				count_reading <= 0;                     //if it is too big, then it resets the received data
		end
	end


	always @(posedge CLK) begin
	if (TRIGGER) begin                                 //If the down counter (CLK/250) is ready
		if (PS2_CLK != PREVIOUS_STATE) begin            //if the state of Clock pin changed from previous state
			if (!PS2_CLK) begin                        //and if the keyboard clock is at falling edge
				read <= 1;                              //mark down that it is still reading for the next bit
				scan_err <= 0;                          //no errors
				scan_code[10:0] <= {PS2_DATA, scan_code[10:1]}; //add up the data received by shifting bits and adding one new bit
				COUNT <= COUNT + 1;                     //
			end
		end
		else if (COUNT == 11) begin                    //if it already received 11 bits
			COUNT <= 0;
			read <= 0;                                 //mark down that reading stopped
			TRIG_ARR <= 1;                             //trigger out that the full pack of 11bits was received
			//calculate scan_err using parity bit
			if (!scan_code[10] || scan_code[0] || !(scan_code[1]^scan_code[2]^scan_code[3]^scan_code[4]
				^scan_code[5]^scan_code[6]^scan_code[7]^scan_code[8]
				^scan_code[9]))
				scan_err <= 1;
			else
				scan_err <= 0;
		end
		else  begin                                    //if it yet not received full pack of 11 bits
			TRIG_ARR <= 0;                             //tell that the packet of 11bits was not received yet
			if (COUNT < 11 && count_reading >= 4000) begin  //and if after a certain time no more bits were received, then
				COUNT <= 0;                             //reset the number of bits received
				read <= 0;                              //and wait for the next packet
			end
		end
	PREVIOUS_STATE <= PS2_CLK;                         //mark down the previous state of the keyboard clock
	end
	end


	always @(posedge CLK) begin
		if (TRIGGER) begin                             //if the 250 times slower than board clock triggers
			if (TRIG_ARR) begin                        //and if a full packet of 11 bits was received
				if (scan_err) begin                     //BUT if the packet was NOT OK
```

```
	always @(posedge CLK) begin
		if (TRIGGER) begin                                  //if the 250 times slower than board clock triggers
			if (TRIG_ARR) begin                             //and if a full packet of 11 bits was received
				if (scan_err) begin                          //BUT if the packet was NOT OK
					CODEWORD <= 8'd0;                        //then reset the codeword register
				end
				else begin
					CODEWORD <= scan_code[8:1];              //else drop down the unnecessary  bits and transport the 7 DATA bits to CODEWORD reg
				end                                         //notice, that the codeword is also reversed! This is because the first bit to received
			end                                             //is supposed to be the last bit in the codewordâ€¦
			else CODEWORD <= 8'd0;                          //not a full packet received, thus reset codeword
		end
		else CODEWORD <= 8'd0;                              //no clock trigger, no dataâ€¦
	end

	always @(posedge CLK) begin
//      if (TRIGGER) begin
//              if (TRIG_ARR) begin
//              LED<=scan_code[8:1];                    //You can put the code on the LEDs if you want to, thatâ€™s up to you
//              if (CODEWORD == ARROW_UP)                   //if the CODEWORD has the same code as the ARROW_UP code
//                      LED <= LED + 1;                     //count up the LED register to light up LEDs
//              else if (CODEWORD == ARROW_DOWN)            //or if the ARROW_DOWN was pressed, then
//                      LED <= LED - 1;                     //count down LED register
		if (CODEWORD == SPACE_BAR)
		    Jump = 1;
		else if (CODEWORD == X)
		    Reset = 1;

		end

endmodule
```

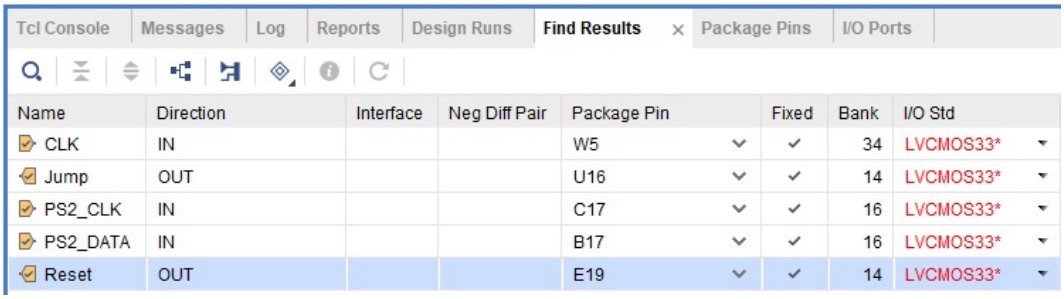
3.6 Input Elaborated Diagram

Here is the elaborated diagram of the input:



3.7 Input I/O Ports

Port configuration:
$PS2\_CLK : C17$
$PS2\_DATA : B17$
$CLK : W5$



Name	Direction	Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
CLK	IN			W5	✓	34	LVCMOS33*
Jump	OUT			U16	✓	14	LVCMOS33*
PS2_CLK	IN			C17	✓	16	LVCMOS33*
PS2_DATA	IN			B17	✓	16	LVCMOS33*
Reset	OUT			E19	✓	14	LVCMOS33*

4 Output Block

4.1 Introduction to the Output Block

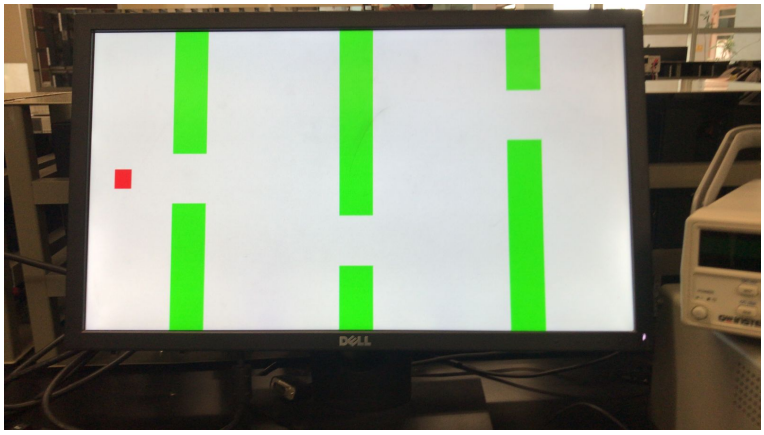
The output block of our project is the VGA display. The most important part of our project is the output and the VGA display. What appears on the screen makes the user excited for the game that they are about to play. The following is how the group decided to map the pixels in the output display of our project.

4.2 Pixel Mapping

The gap between the tubes in our output display will be 80 pixels. This will remain constant throughout the game.

The tube height of both the sides will be varying because the obstacles have to be made in a challenging manner so the user has to use the key to navigate the flappy bird up and down. The width of the tube height is 40 pixels of the screen.

The dimensions of the flappy bird are 20x25. This means that the x component of the bird is 20 pixels wide and the y component is 25 pixels long. Each input press will jump the bird 30 pixels upward in the y direction. Every time the user provides an input for a jump, the bird will have 6 transitions of 5 pixels each in order to complete the 30 pixel jump, and the transitions will complete regardless of the input from the user at those instances. The game will read the input after all 6 transitions have been completed. Where as during the downfall the bird will have transitions of 5 pixels each (downwards) for as long as its in the downfall state.



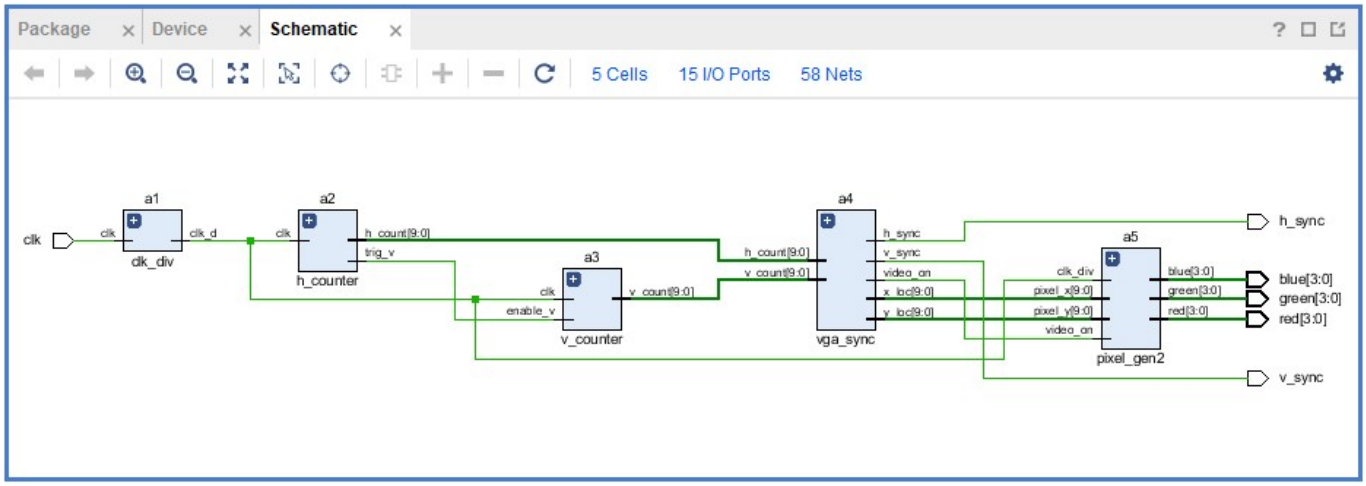
4.3 Output Demo Code

Following is the display output and the output code for the demo display:

```
module pixel_gen2(
  input [9:0] pixel_x,
  input [9:0] pixel_y,
  input clk_div,
  input video_on,
  output reg [3:0] red = 0,
  output reg [3:0] blue = 0,
  output reg [3:0] green = 0
);

  always @ (posedge clk_div)
      begin
      if ((pixel_y >= 225 && pixel_y <= 250) && (pixel_x >= 30 && pixel_x < 50))
//        drawing the bird
        begin
        red <= 4'hf;
        blue <= 4'h0;
        green <= 4'h0;
        end
      else if (((pixel_x >= 300 && pixel_x < 340) && ((pixel_y >= 380 && pixel_y < 480) || (pixel_y >= 0 && pixel_y < 300))) ||
        ((pixel_x >= 500 && pixel_x < 540) && ((pixel_y >= 0 && pixel_y < 100) || (pixel_y >= 180 && pixel_y < 480))))
//        drawing the pipes
        begin
        red <= 4'h0;
        blue <= 4'h0;
        green <= 4'hf;
        end
      else
        begin
//        painting the rest of the background white / also drawing 1 more pipe
        red <= video_on ? ( ((pixel_y >= 0 && pixel_y < 200) || (pixel_y >= 280 && pixel_y < 480)) ? ( ((pixel_x >= 100 && pixel_x < 140)) ? 4'h0 : 4'hf) : 4'hf) : 4'h0;
        green <= video_on ? ( ((pixel_y >= 0 && pixel_y < 200) || (pixel_y >= 280 && pixel_y < 480)) ? ( ((pixel_x >= 100 && pixel_x < 140)) ? 4'hf : 4'hf) : 4'hf) : 4'h0;
        blue <= video_on ? ( ((pixel_y >= 0 && pixel_y < 200) || (pixel_y >= 280 && pixel_y < 480)) ? ( ((pixel_x >= 100 && pixel_x < 140)) ? 4'h0 : 4'hf) : 4'hf) : 4'h0;
        end
      end
endmodule
```

4.4 Output Elaborated Diagram

Here is the elaborated diagram of the sample output:



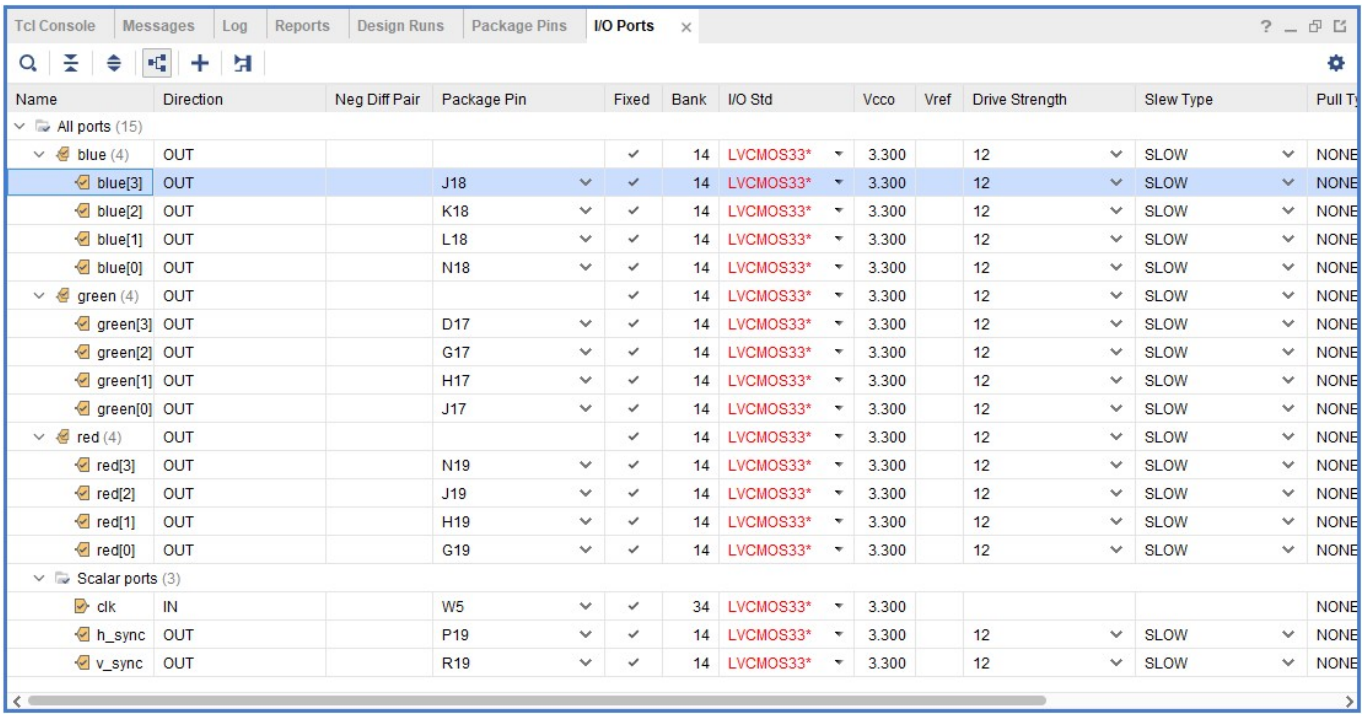
4.5 Output I/O Ports

Port configuration:
$blue[3] : J18$
$blue[2] : K18$
$blue[1] : L18$
$blue[0] : N18$

$green[3] : D17$
$green[2] : G17$
$green[1] : H17$
$green[0] : J17$

$red[3] : N19$
$red[2] : J19$
$red[1] : H19$
$red[0] : G19$

$clk : W5$
$h\_sync : P19$
$v\_sync : R19$



Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Ty
All ports (15)											
blue (4)	OUT			✓	14	LVCMOS33*	3.300		12	SLOW	NONE
blue[3]	OUT		J18	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
blue[2]	OUT		K18	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
blue[1]	OUT		L18	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
blue[0]	OUT		N18	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
green (4)	OUT			✓	14	LVCMOS33*	3.300		12	SLOW	NONE
green[3]	OUT		D17	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
green[2]	OUT		G17	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
green[1]	OUT		H17	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
green[0]	OUT		J17	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
red (4)	OUT			✓	14	LVCMOS33*	3.300		12	SLOW	NONE
red[3]	OUT		N19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
red[2]	OUT		J19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
red[1]	OUT		H19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
red[0]	OUT		G19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
Scalar ports (3)											
clk	IN		W5	✓	34	LVCMOS33*	3.300				NONE
h_sync	OUT		P19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE
v_sync	OUT		R19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE

5 Control Block

The description of the states of our project are as followed. In addition, the state transition diagram of the project is also attached below.

5.1 States of our project

1. Reset: Before starting the game, the bird is in the reset state and so is the game. If the space bar is pressed the position of the bird is changed and the bird moves up and the game begins. If the space bar is not pressed, the game remains at the idle/reset state.

2. Up: When the bird is in an upward motion, it is our discretion if we want to make it jump more by pressing the space bar or let it descend by choosing to not press the space bar.

3. Down: When the bird in the downward position, it is our discretion if we want to make it descend more by not pressing the space bar or make it jump by pressing the space bar.

5.2 State Transition Diagram

Here is the state transition diagram:

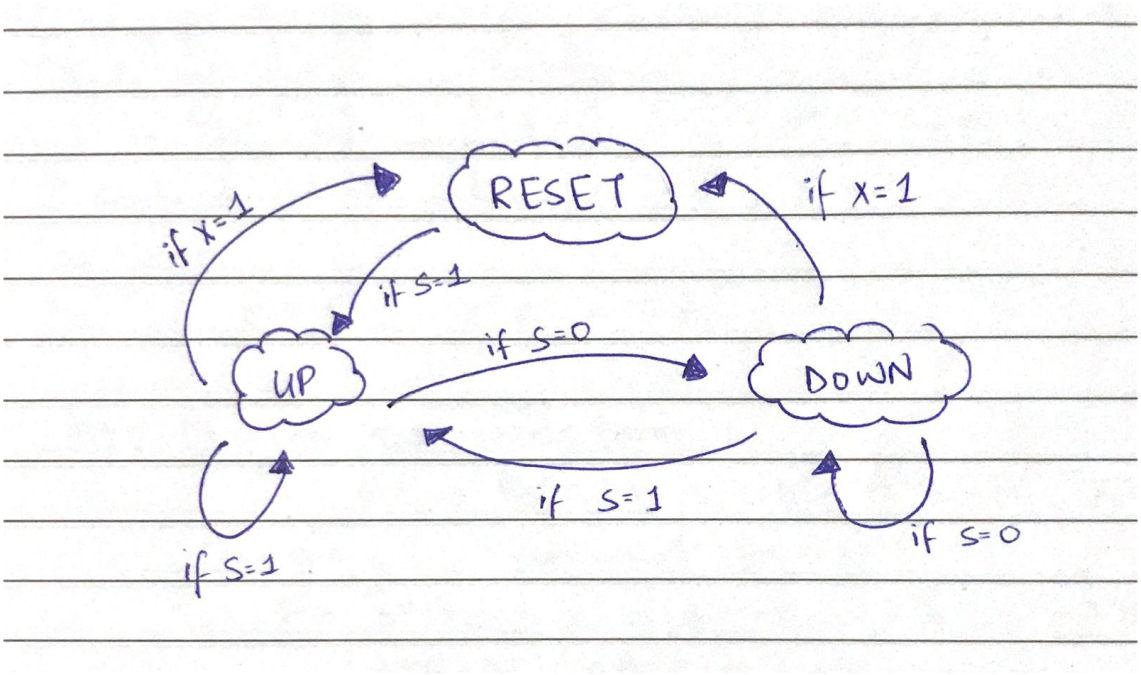


Figure 1: X = reset, S = spacebar

6 Conclusion

The following report introduces the readers to the game flappy bird that is made by the group. In the report, the readers are being introduced to the user-flow diagram of the project and the division of blocks in the user-flow diagram. In addition, the report introduces the users to the detailed input block, control block and the output block of the project. The input block focuses on the aspect of understanding the IO pins and the input that will be the keyboard in this project. The output block explains the the pixel division of the components that are made in the project i.e., the flappy bird and the tubes. The control block of the project explains the states of the project and the state transition diagram of the project.

7 References

Keyboard Input: `https://www.instructables.com/PS2-Keyboard-for-FPGA/`
Key scan codes and IO pins: `https://digilent.com/reference/programmable-logic/basys-3/start`