Chungchhay Kuoch
004843308
Karthik Ramesh
104687637

# Lab 4 Report: Flappy Bird

(LIST OF THINGS LEFT TO DO:
Figures for each of the .v files. Add the design implementation for these
Add pictures to show that we met the requirements. )

**Introduction:**

The purpose of the lab is to design flappy bird using Verilog and the Nexys3 board. The player can control the bird by using three buttons on the Nexys3 board. The right (D9) button is for the bird to jump, the bottom (C9) button to pause the game, and the left (C4) button to restart the game. When a player scores a point, we increase the score counter, make a beep sound with the piezo buzzer, and display the score on the seven segment display. We increase the bird's speed when the score is greater than 10 and increase it further when the score is greater than 30. We only increase twice because we think it moves too fast past that speed. The bird is dead when it hits either the top/bottom wall, or the edges of the screen.

We use the VGA port to connect to a monitor to display the game's visuals.
By showing the graphical of the game, we also use VGA to connect the Nexys3 board with the monitor. We reached all of the goals on our final project proposal, as well as the extra credit for making the bird move faster.

**Game Design Implementation:**

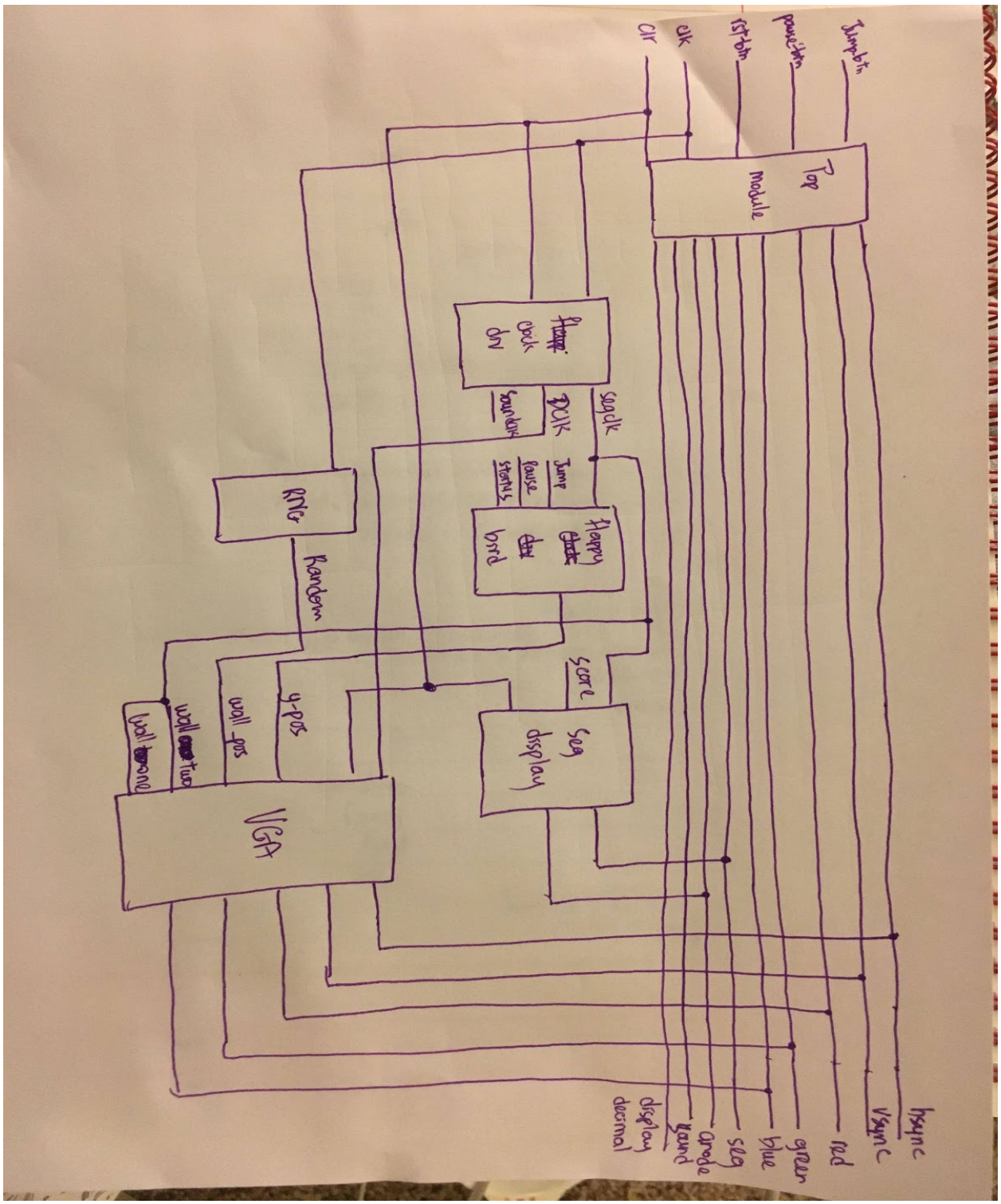


***Figure 1: This is the overview block diagrams of the game design.***

RNG.v: In this file, we get an RNG value for the initial wall's position, so that we could vary the height at which the walls show  up, so that the player can play without recognizing a pattern in the manner at which the walls show up. This file was inspired by the code in this link: http://simplefpga.blogspot.com/2013/02/random-number-generator-in-verilog-fpga.html
From the code in this link, I modified it a little bit more in order to get it to work. When I did it at first, the walls did not change in height because I did not pass in a reset state, so I created an initial state myself, wherein the rand number was set to all 1's so that when we XOR'd it with other random bits in the number, it generated a pseudo random number. This is an LSFR RNG.

This file's module was used by the top file in order to create a random position at which the wall will be positioned at.

Nexys3.ucf: This has all of our FPGA board's PINs, buttons, switches, connectors, etc. In this file, we enabled the master clock, D9 (jump), C9 (pause), and C4(reset) buttons, the VGA PINs, seven segment display, and a ping for the sound to output through the piezo buzzer. This was taken from src file from lab1, or lab 0 in the CCLE list of items.

Clockdiv.v: This was one of the three files taken from the NERP demo folder, and modified. This was modified to add one more clock--the frequency at which the sound plays. It has the other clocks inside it too, such as the dclk, for the rate at which the pixels on the screen are changed, and the segclk, for the seven segment display's refresh rate, both of which were already given from beforehand. I used the segclk for not only the seven segment display's refresh rate, but also the rate at which I can press the buttons, and changing the game's status based on collisions, and the restart/pause buttons being pressed.

This file's module is used by the top file. It uses the different clocks from this module to run different jobs at different speeds. For example, the walls have to move to the left at a certain speed relative to the bird, and when you press the jump button, the bird has to jump up not too slowly, but at the same time, not too quickly, so at just the right pace. The pixels on the screen, however, have to refresh exceptionally quickly, while the seven segment display has to refresh not fast, but not too fast because that is just overkill (it would be ok to do that, but it is not necessary.)

Segdisplay.v: This file was used to show the score on the seven segment display for the current game. It is also a file that was already used in the NERP demo folder. We used the logic for making all four letters NERP showing up on the display to cycle through the four different analog displays and show the corresponding score, with overflow properly. For example, the score:15 would show up as 0015 on the display in seven segments, properly with overflow. The number 9999 overflows over to 0000, so the max score that one can achieve is 9999.

This file's module is used by the top file. The top file feeds in the segclk, a clr button (which I kept from the NERP top demo, but do not actually use for this project), and a score. It outputs the proper analog with the corresponding score in seven segment display-form to the top file.

Vga640x480.v: This file was used to display the game through the VGA connector onto the monitor. This was originally also from the NERP top demo, as well. We changed it to take in the positions of the bird, as well as the walls in order to display them onto the screen. We had to make sure that the coordinates we used for collision in the top file matched up with the coordinates here for the bird and the wall, so that the collisions and gameplay are accurate with the design implementation that was running in the background.

The top file called this function with all the regular commands that it normally did in the NERP top demo, but also added on the bird and walls' coordinates. It outputted hsync, vsync, and RGB. I am not too sure how the logic in this file works exactly, but it is mostly somewhere along the lines of a vertical and horizontal pixel component sweeping through the screen pixel by pixel, and depending on the position of the horizontal and vertical pixels, you decide what color to light it up. Something worth noting in this file that I learned from looking at the original NERP top demo's vga file is that in the series of if-else-if statements that are used to set the color for the bars, the earlier you draw the color of the object, the more precedence it has in how it shows up on the screen. This may be hard to explain, so let me attempt to explain it with an example: we colored in our bird in our first if statement, so the things that show up in the if-else statements afterwards show up "behind" the bird relative to us looking at the screen. So if I were to switch the positions of the if-else-if statements at which I color the bird, with for example the background, then the background would take precedence and the bird would not show up on the screen.

In this file, our bird is yellow, the walls are blue, the background green, and the ground pink. We got these color codes by looking at the NERP demo's vga file. Our bird is 30x30 units, and the wall is 40 units wide, with 100 units of gap between the first and second segment of the walls. The ground is 11 units off from the bottom edge of the screen.

We calculated the bird's, and walls' positions in the flappybird.v and top.v files.

Flappybird.v: In this file, we took in from the top file, the segclk, pause/jump states, and status of the game, and outputted the bird's coordinates, labelled y_pos because the user can only move the bird in the y direction by pressing the jump button. In this file, we check the status of the game, and as long as it is on, we allow the bird to "jump". The bird's clock for jumping is actually about 380/32 = 12 Hz. I used this clock because when I tried to jump while running at the regular segclk, it jumped to the ceiling too fast--it looked like it teleported to the ceiling, so I decided to make it slower by making a new clock and triggering the jump mechanism only when the new clock was at the positive edge of [4]. I got to setting the positive edge to [4] on the new clock only after trial and error, by seeing what was too fast or too slow. I came upon the logic of creating clocks like this from the clockdiv file, where they did it for us with the given code beforehand from the NERP demo (for example, by setting the segclk to q[16], equivalent to 50Mhz / 2^17 = 381.47Hz).

For the design of this file, at the positive edges of the new clock[4], if the status is game over, I set the velocity to 0, but if the status is at reset, I set the initial height to an arbitrary value of 250, and the velocity to 0. However, upon starting the game at first, the flappy bird's y position is set to 300, and velocity to 0. The flappy bird I created takes into account acceleration and deceleration. (There is no explicit reasoning for choosing these values for these situations.) When the status shows that the game is on, and the state of the game is not in pause, then I went ahead and started the "physics engine" part of my code.

In the physics engine, I first check to see if the bird is out of bounds by hitting the bottom and top of the screen. If it is, then it should make the game end in the top file. If that does not happen, then the bird is in bounds. Then, the bird checks if its current position plus the velocity is within the bounds from 0 to 480 ( according to the vga's horizontal video: 511 - 31 = 480). I then subtract 15 from 480 to check if it is within 465 because the size of the bird is

(15+15)x(15+15). If it is within the bounds, then it updates the new y position to be the old y position plus the velocity. Then, I check to see if the jump button is pressed and the current velocity is less than -3 (to see if the bird is falling down), and if those two cases are met, then I increment the velocity by 10 to first offset the negative three, then to add a bit more to actually accelerate the bird in the positive direction, too. Lastly, I check to see if the bird's velocity is greater than -5 to decelerate the bird while the jump button is not pressed, but at the same time limit it to greater than -5 because after that, the bird would start falling down too fast and not float down. A lot of these values were found from trial and error. After this, the y coordinate (y_pos) was outputted to the top file.

Top.v: This is the file that puts it all together. All the other files mentioned earlier are called by this file, and joined together to form some more core logic for this project. I first initialize a bunch of registers that I later assign stuff to. I got this file from the NERP top demo folder, too. I kept everything the same from that project, except that I also added inputs for the jump_btn, pause_btn, and rst_btn that come from the Nexys3.ucf file. I output a wire called vcc for the sound to beep whenever I score a point by setting the vcc to the soundclock that is bitwise and-ed with a flag that turns on when a point is scored, and off a certain time afterwards to allow the sound to actually audibly play for a short duration.

First, I initialized another clock called wallspeed in order to move the wall at a certain speed. I set the walls to do stuff when the wallspeed was at a positive edge of [17]. I originally had it at the segclk speed, but that was too fast, so then, I created this new one that incremented itself at the posedge of dclk, and went slower than the segclk by 4 times. This offered a decent enough speed for the moving walls. At the posedge of wallspeed[17], if the status was game reset, then I assigned initial values for the walls, and reset the score. If the walls reached a position of 400 units along the x-coordinate, I increase the score counter, set the flag to play the beep from the piezo buzzer to one, and set wallOne to wallTwo, and wallTwo to a random height. This then gets passed onto the vga file, where the gaps between the wall segments, and distances between the walls are created properly. However, if the wall has not reached an x-coordinate of 400, then I continue incrementing the wall's position, and a count to stop the sound. If the count to stop the sound is greater than 63, then I set the flag to play the sound and count to stop the sound to zero. This indicates the duration of the beep. If I had not set the flag to be on for a certain count like this, then the beep would have been too short in duration to hear.

In the posedge of the dclk, I also increased the speed of the game as people reached certain scores, in order to meet the extra credit part of the assignment: at 11 points, the speed doubles, and at 31 points, the speed doubles again. After that, we did not increase the speed because it would make the game unplayable.

Additionally, I use the segclk's posedge to check the statuses and update them based on the current situation. For example, if there is a collision, then set the status to game over. The coordinates I use for this are related to the coordinates I use in the vga file to fill in the pixels for the bird and walls. Something else that happens in the posedge of the segclk in this top file is that it sets the jump and pause states when the respective buttons are pressed, which are later fed into the bird module. The clock used here is again arbitrary; I could have used another clock, but this was something I tested in the beginning, and used segclk, which was already defined early on in the project.

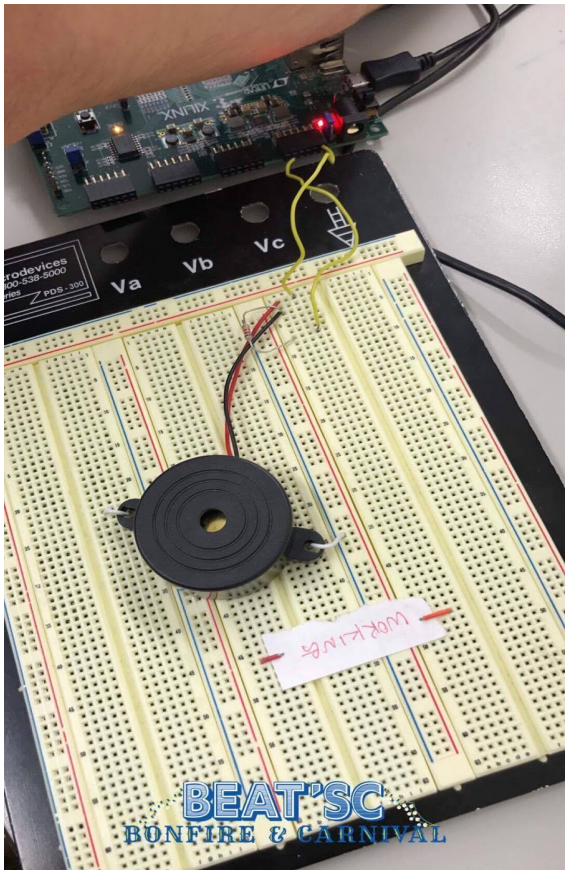


***Figure 2: This is the board that we use for the sound. Whenever we score, we use this as a beep sound.***

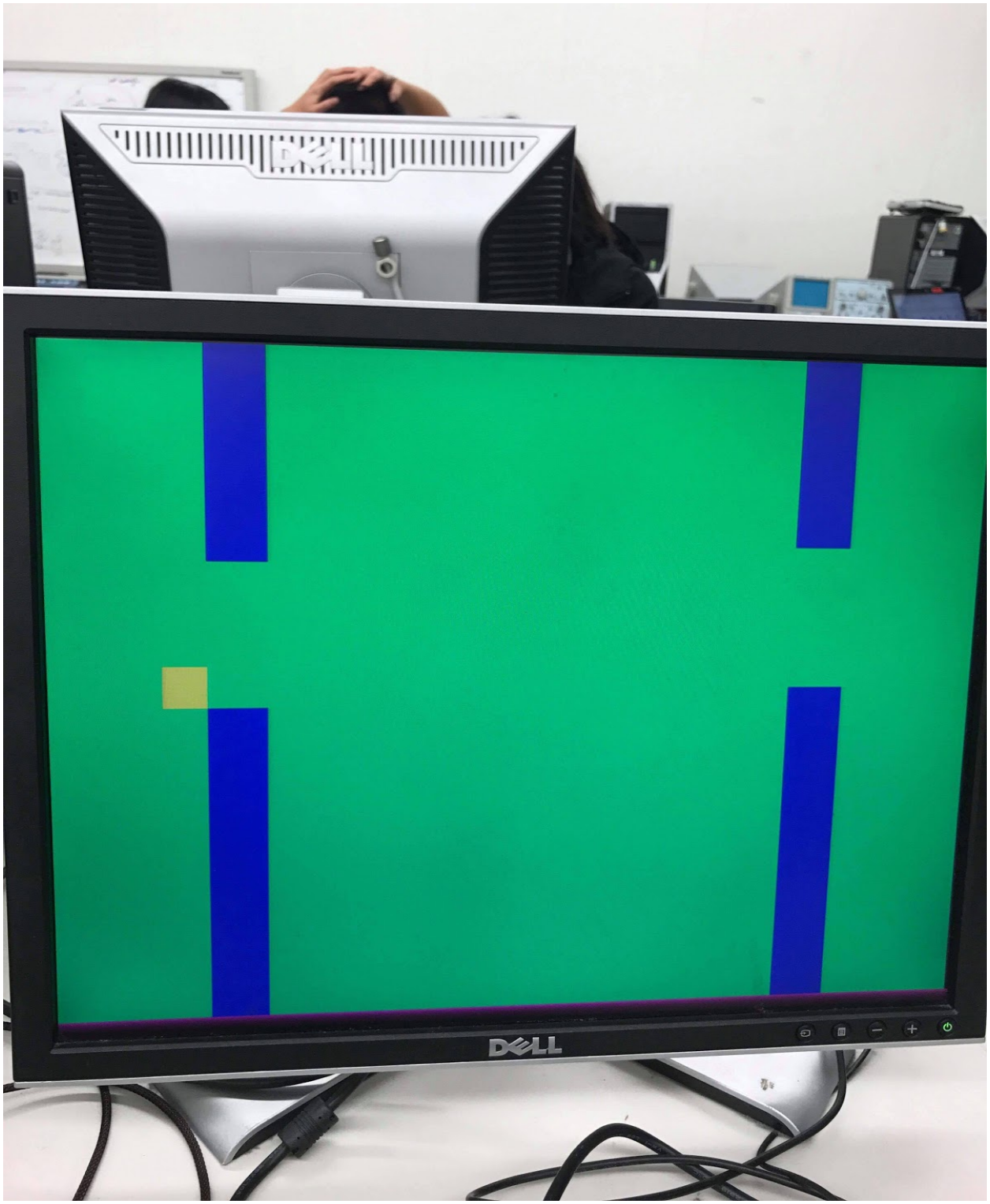


***Figure 3: When the bird hits the edge, it's dead. This shows that we have reached the perfect edge for the bird to die***

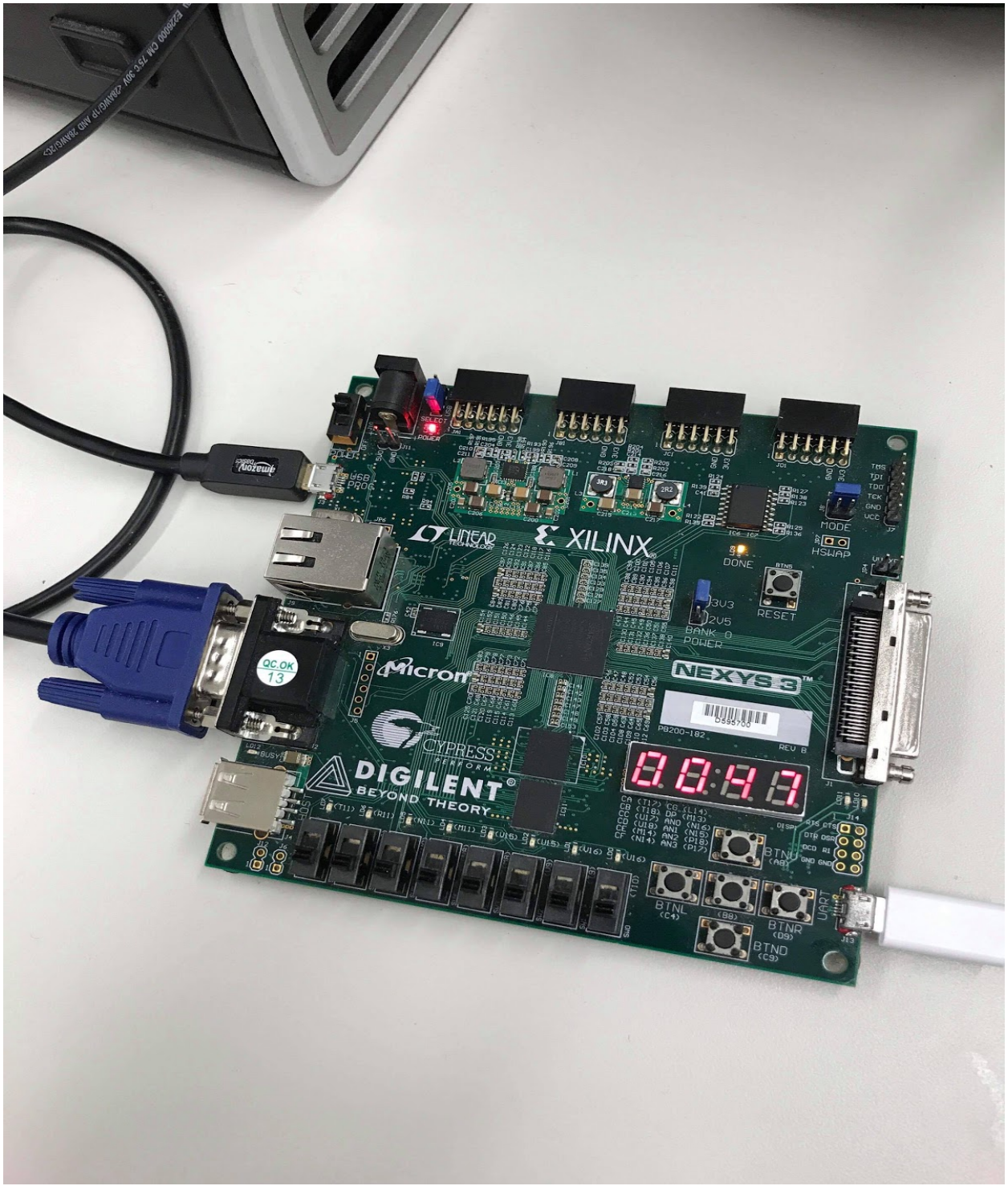


***Figure 4: We use the seven segmentations display on the board to display the score***

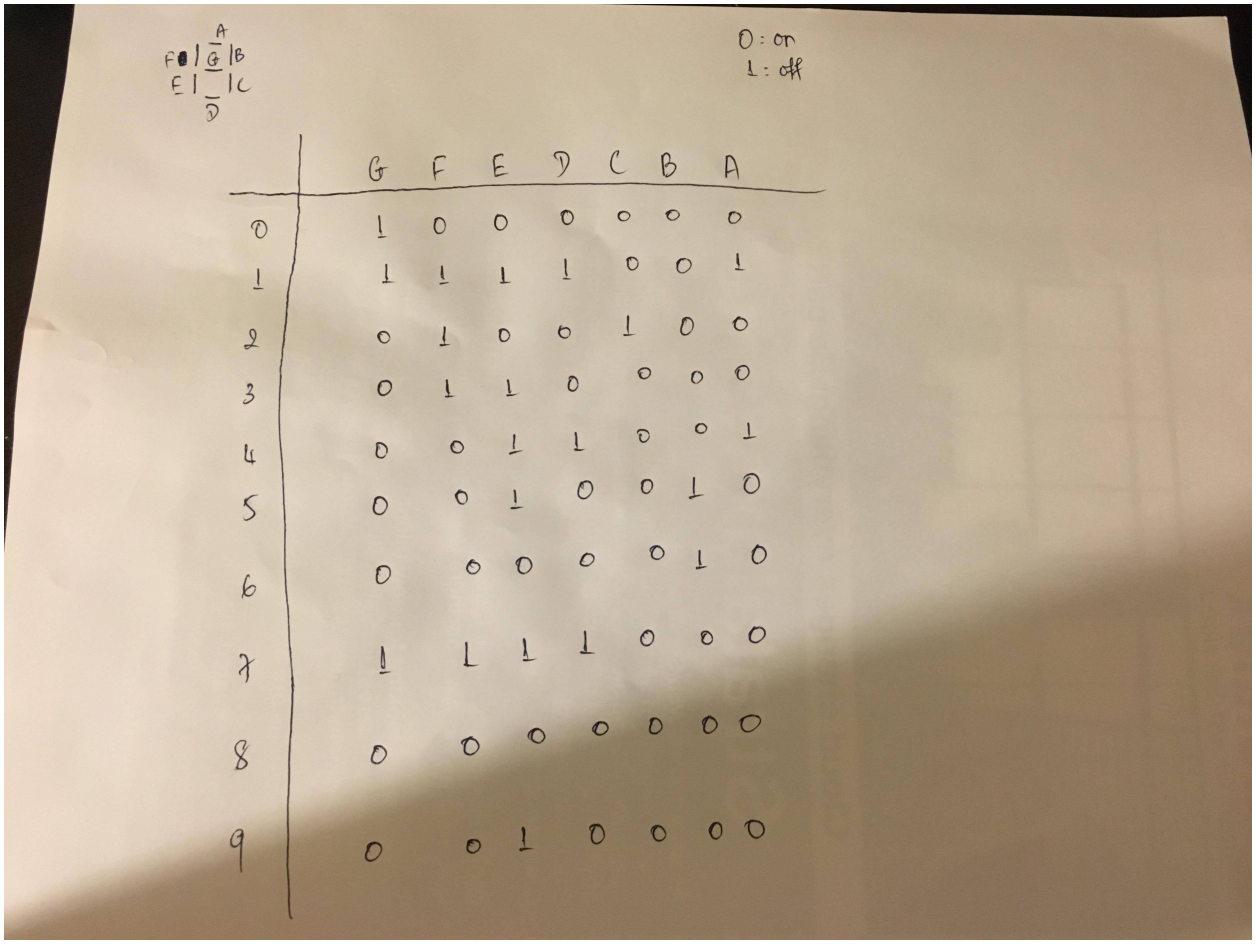


***Figure 5: Seven segment representation of digits 0-9***

**Approach to Project:**

For this project, we first looked at the NERP demo files, and recreated that demo, which did not take too long. Afterwards, we worked on creating our bird and walls on the screen, without implementing the bird's physics, or collisions; we just had the visuals. Next, we created the status checks in the posedge of the segclk, which we mentioned shortly before this (however, at this point, we left out collisions). Then, we created the flappybird.v file to see the physics of the bird, and fed that into the top module, which put it into the vga file. Now, we were able to see the bird jump when we pressed the jump button, the game reset when we pressed the reset button, and pause when we pressed the pause button. The walls, however, were not moving. So, we then implemented the walls to move. This was one of the toughest part of the project, only next to the bird's physics. We had a lot of trouble trying to figure out how to make both of the walls

show up on the screen properly. For example the wall had to slide into the screen. This took a bit of time to implement. Later on, we decided to create two different walls, and set wallOne equal to wallTwo after the distance between them exceeded 400, which in our case was the distance that was supposed to be between the walls. This handoff expression we created allowed us to hand off wallTwo into wallOne , then assign wallTwo a value from the RNG and print it accordingly in our VGA properly. However, figuring this out took us a long time. Also, in the RNG, we had to implement an initial state instead of the reset state that they mentioned in the code example because we did not pass in a reset state to begin with, but still needed a way to initialize the value of the rand number to all 1's so they we can XOR it, then shift it to get a random number. Then, we fixed our seven segment display to show the points scored in four seven segment analog numbers. The sound also took us a while to implement, as we did not know of anyone else in our class using the piezo buzzer we were using, so we had to figure it out pretty much by ourselves. For the extra credit part of our lab, it was pretty easy due to the way in which we implemented this. We made the walls move based on a clock, so in order to make the game's speed increase, we just had to increase the speed of the clock at certain checkpoints: at 11 points, the speed doubles, and at 31 points, the speed doubles again. After that, we did not increase the speed because it would make the game unplayable.

Due to such an approach of the project, wherein we first tried to implement the NERP demo, we did not really have to create testbenches to test our project. The only case wherein I could have seen using testbenches was maybe the seven segment display, but that worked pretty quickly for us. (Note: You may remember that we said early on in our project that it was not working, but that was because we purposely held it off until later so that we could get some of the more important parts of the project done! It was a strategic move that, in the end, did not really matter, but it was more for safety that we approached the project in such a way.)

**Conclusion:**

From this project, we learned how to use our creativity and previous knowledge we gained from labs 0-3 to combine them and put them to use to create something that was out of our comfort zone. It challenged us, and tested our limits of using verilog code, but we approached the big picture module by module, part by part, and slowly checked stuff off of our checklist and ultimately ended up with our project. This project was similar in difficulty to project three, but

maybe a little bit harder. Project three was about managing a lot of clocks and displaying it on the seven segment display, but this was more about using the vga, and the game logic behind it. The core focus off the projects were different, but similar in the amount of work required, just that this was definitely a few levels tougher.

There are some things I wish I could improve on this project further, such as adding cooler backgrounds with moving distractions, finer details on the walls, and making the yellow block look more like a bird. Functionality-wise, the project is complete, and pretty much perfect, though. In that department, there is not much I can do to make it better.

The toughest parts of this project were figuring out how the NERP demo worked, and then recreating it ourselves. Then, it was figuring out the bird's physics, how to make two different walls slide into and out of the screen, and the collisions. Once we figured out how to make the walls both appear on the screen properly, the collisions became much easier automatically.