Final Project: Space Invaders

by
Tom Faulconer

A Technical Report Submitted to the Faculty of Computer Science and Engineering

University of Nebraska-Lincoln

Submitted in partial fulfillment for the requirements of
CSCE 436 - Advanced Embedded Systems

April 24, 2020

Contents

List of Figures

1 Design Goals

1.1 Need Statement

I was discussing the state of the engineering college with our senior design professor and he noted that the electrical engineering enrollment has been declining and was asking us for ways to enhance enrollment. I propose the way to get kids into electrical engineering (or computer engineering) is to show them something they are interested in and excited about (i.e. Showing them they can create classic video games on an FPGA). Therefore, this tool will be used as a recruiting tool to show potential students and get them enrolled in the electrical engineering program. This would be an easy tool to set up and use for marketing to prospective students.

1.2 Marketing Requirements

The FPGA re-created Space Invaders will tried to be recreated as closely as possible. The design will be powered by wall outlet, and output on 640x480 VGA resolution via an HDMI cable.

1.3 Level-0 Graphic Diagram

The Level-0 Diagram is the first attempt at thinking through the system level design for my final project. What modules are required, how things are connected together, etc. This is the first pass at it. It can be seen if Figure 1 below:

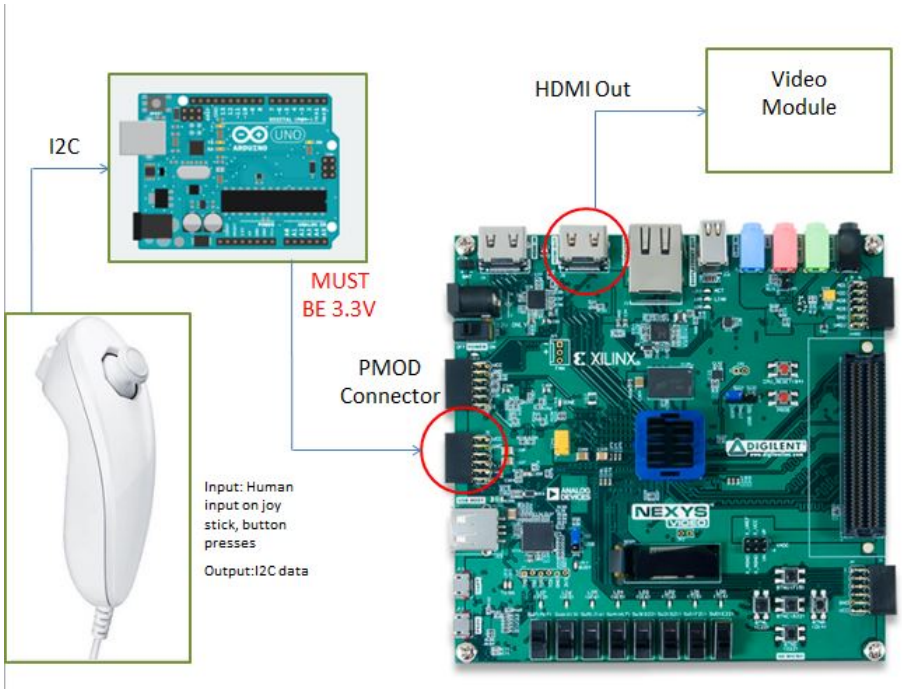


Figure 1: Level-0 Block Diagram

2 Detailed Design

2.1 Level-1 Diagram

After doing some further research on the project and the modules involved. This tasks required me to look deeper at each component and revise the Level-0 diagram if I needed to. As it happens, the Level-0 diagram got pretty close to actual Level-1 Diagram. Each module describes the inputs, outputs, and behavior for each module in the design. The Level 1 block diagram can be seen in Figure 2, while the tables can be seen in Figures 3-5.

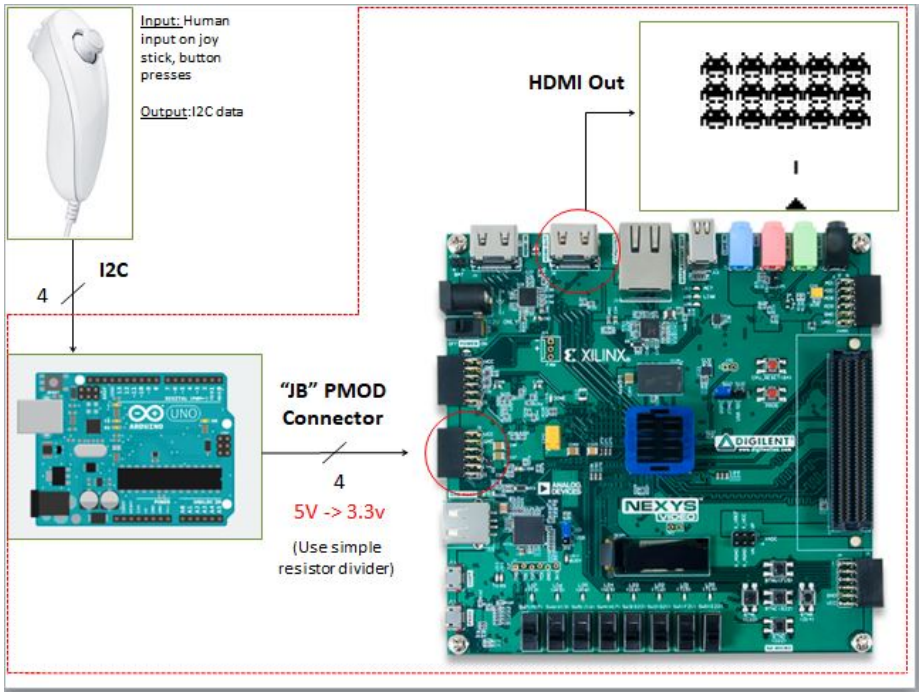


Figure 2: Level-1 Block Diagram

Module	Video (on FPGA)
Inputs	HDMI
Outputs	TMDS Signals
Behavior	This module will be used from our previous labs and take HDMI signals and turn them into TMDS signals to display on a screen

Module	Wii Nunchuk
Inputs	Player inputs
Outputs	I2C data (Z button, C button, left and right movement)
Behavior	This module will decode Arduino I2C values, decode them and use them as the human input to the system

Module	Arduino
Inputs	I2C Data from Wii Nunchuk
Outputs	Transformed input from Arduino to the FPGA (Z button, C button, left and right movement)
Behavior	Take input from the Wii nunchuk controller and send them to the FPGA for processing

Figure 3: Level-1 Table - Part 1

Module	Scope Face
Inputs	Player inputs
Outputs	I2C data (Z button, C button, left and right movement)
Behavior	Draw game boundary, player ship, and aliens

Module	Video
Inputs	Row, column locations, TMDS signals
Outputs	I2C data (Z button, C button, left and right movement)
Behavior	Draw game boundary, player ship, and aliens

Module	TMDS
Inputs	VGA signals
Outputs	HDMI Signals
Behavior	Draw game boundary, player ship, and aliens

Figure 4: Level-1 Table - Part 2

Module	DVID
Inputs	Clock, VGA signals
Outputs	Red, Green, Blue values
Behavior	Helps draw game boundary, player ship, and aliens

Module	Counters
Inputs	Control Words
Outputs	Column/row count
Behavior	Helps keep track of the rows and columns for the system

Figure 5: Level-1 Table - Part 3

2.2 Datapath and Control

The datapath and control section will describe the inner logic workings of the design. The Datapath provides feedback to the control unit with status words, while the control unit takes those status words and determines what to do. This work was heavily influenced by reference (2) below.

The Datapath can be seen in Figure 6 and Figure 7.

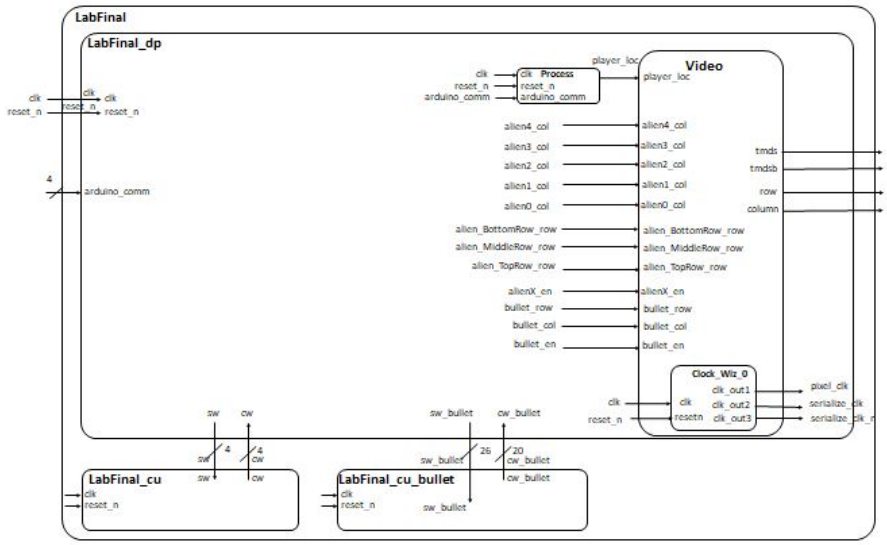


Figure 6: Datapath - High Modules

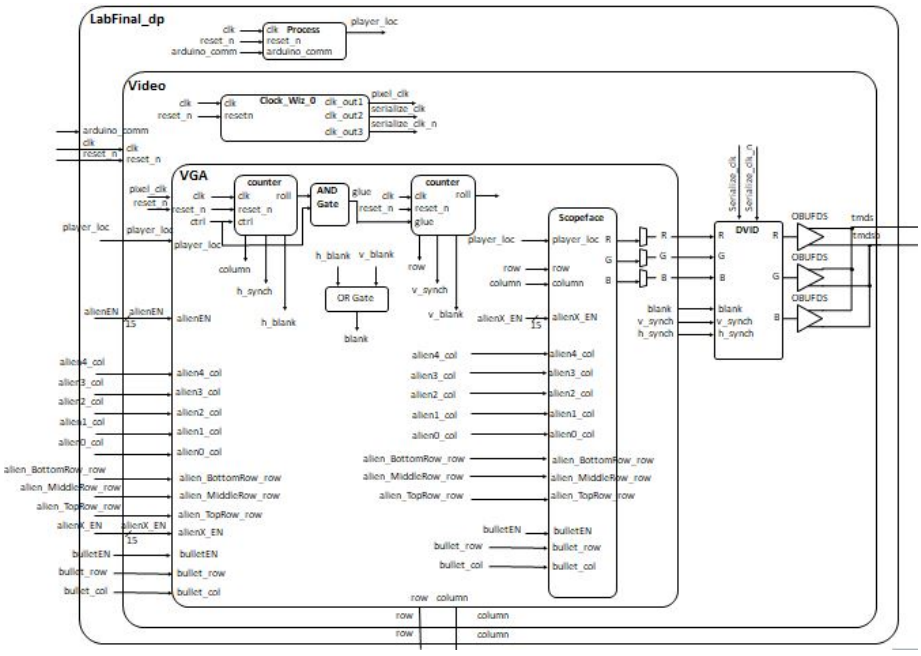
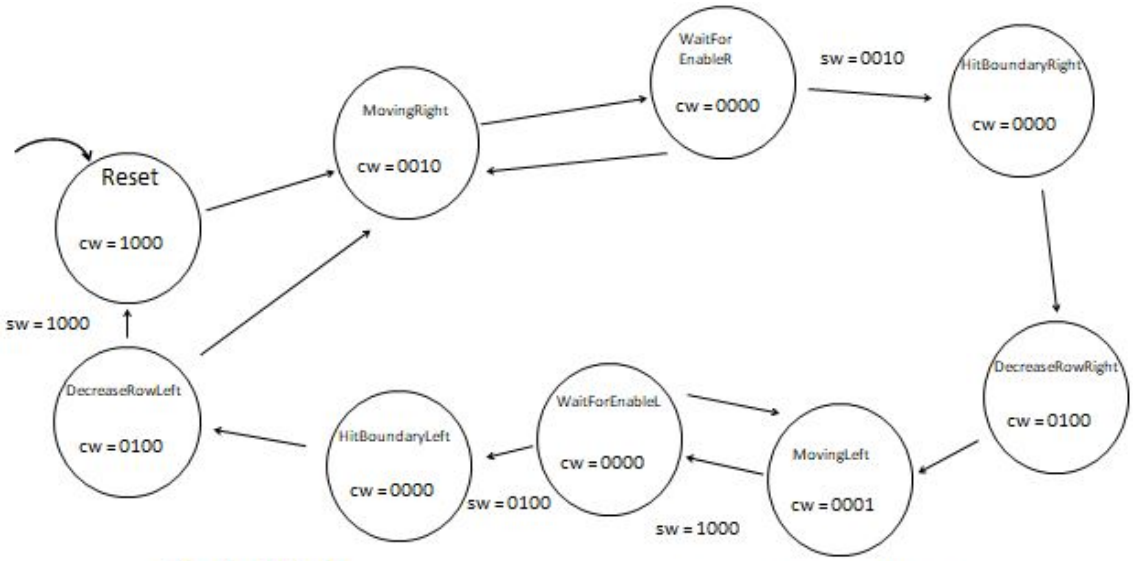


Figure 7: Datapath - Lower Modules

In my design, I decided to use two control units. One control unit controls the aliens moving and hitting boundarys. For this, all we needed was 4 status words and 4 control words. The control unit utilizes a Moore Finite State Machine (FSM). This can be seen in Figure 8.



Control Words

Bit 0	Move Left
Bit 1	Move Right
Bit 2	Decrease Row
Bit 3	Reset to Defaults

Status Words

Bit 0	Alien_clk_en
Bit 1	Alien Column = 440
Bit 2	Alien Column = 220
Bit 3	Alien Bottom Row hits boundary

Figure 8: Control Unit - Alien Movement

This control unit moves the aliens to the right, checks to see if they hit the game board boundary, decreases a row, moves to the left, checks to the if they hit the game board boundary from the left, decreases a row.

Figure 9 shows the bullet control unit and Figure 10 shows the status and control words used in it. The control unit for the bullet movement is much more complicated than the alien movement control unit. It has 27 status words and 21 control words. Most of these are used toggle the alienX_en. Since there are 15 aliens there are 15 enables to keep track of. Then there is the piece that we have to keep track of alien collisions with the bullet. I noticed early on that I didn't need to keep a row and a column on each alien (resulting in 15 rows, 15 columns and 15 enables = 45 signals). Instead, each top row alien has one row signal and since there are 5 columns, that's all I needed to keep track of all the aliens. So I now have 3 row signals (top row, middle row and bottom row) and 5 column signals (for a total of 3 row + 5 columns + 15 enables = 23 signals). This is much easier to keep track of and resulted in many less states in my FSM. In a nutshell, the FSM waits for the fire command, moves the bullet upwards, and then waits to see if the bullet hits an alien or the top boundary, and then resets. Quite simple actually.

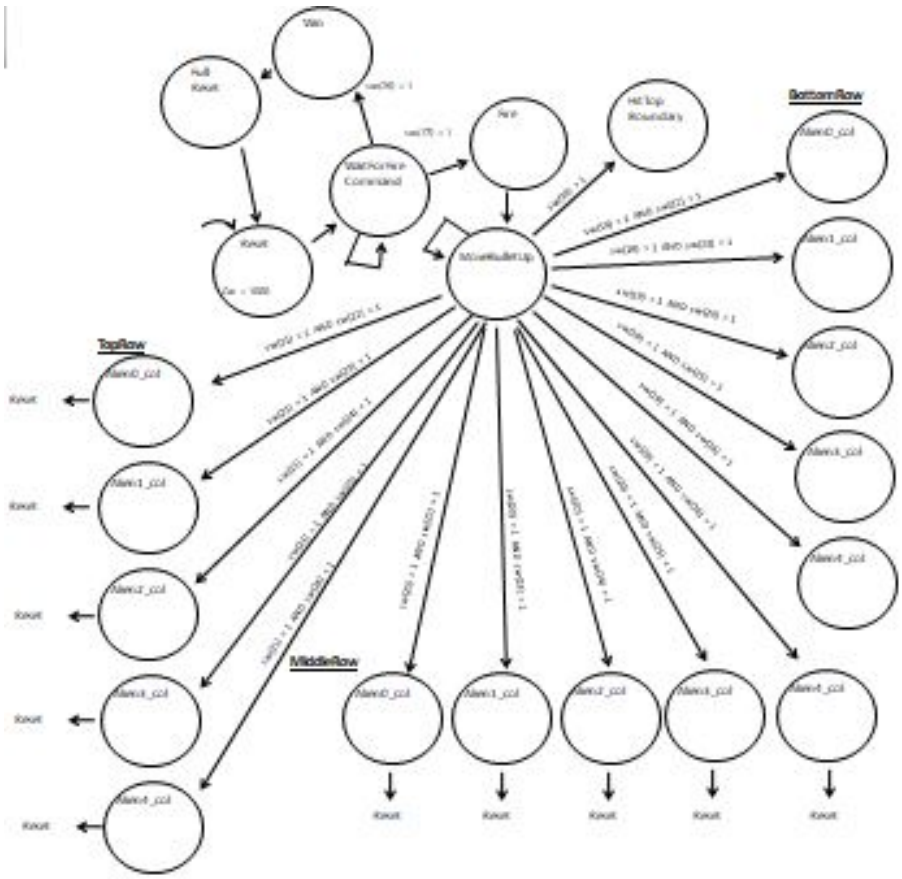


Figure 9: Control Unit - Bullet Movement

Status Words	
Bit 0	Z_Button = '1'
Bit 1	Bullet_clk_en
Bit 2	BottomRow_collision
Bit 3	MiddleRow_collision
Bit 4	TopRow_collision
Bit 5	Col0_collision
Bit 6	Col1_collision
Bit 7	Col2_collision
Bit 8	Col3_collision
Bit 9	Col4_collision
Bit 10	Top Boundary hit
Bit 11	Alien0_en
Bit 12	Alien1_en
Bit 13	Alien2_en
Bit 14	Alien3_en
Bit 15	Alien4_en
Bit 16	Alien5_en
Bit 17	Alien6_en
Bit 18	Alien7_en
Bit 19	Alien8_en
Bit 20	Alien9_en
Bit 21	Alien10_en
Bit 22	Alien11_en
Bit 23	Alien12_en
Bit 24	Alien13_en
Bit 25	Alien14_en
Bit 26	Win_counter

Control Words	
Bit 0	Alien0_en
Bit 1	Alien1_en
Bit 2	Alien2_en
Bit 3	Alien3_en
Bit 4	Alien4_en
Bit 5	Alien5_en
Bit 6	Alien6_en
Bit 7	Alien7_en
Bit 8	Alien8_en
Bit 9	Alien9_en
Bit 10	Alien10_en
Bit 11	Alien11_en
Bit 12	Alien12_en
Bit 13	Alien13_en
Bit 14	Alien14_en
Bit 15	Bullet_en
Bit 16	Move bullet up
Bit 17	Set bullet_column
Bit 18	Reset bullet row
Bit 19	Win
Bit 20	Full Reset

Figure 10: Control Unit - Bullet Movement - Status and Control Words

2.3 Calculations

There were several calculations required to complete this project. First things first, I needed to use the Arduino (5V logic) to capture the I2C signals sent from the Wii Nunchuk. The PMOD connectors on the Nexys Video use 3.3V level logic. I utilized a simple resistor divider to achieve 3.0V. This ensured that I did not fry an hardware on the FPGA. This can be seen in Figure 11.

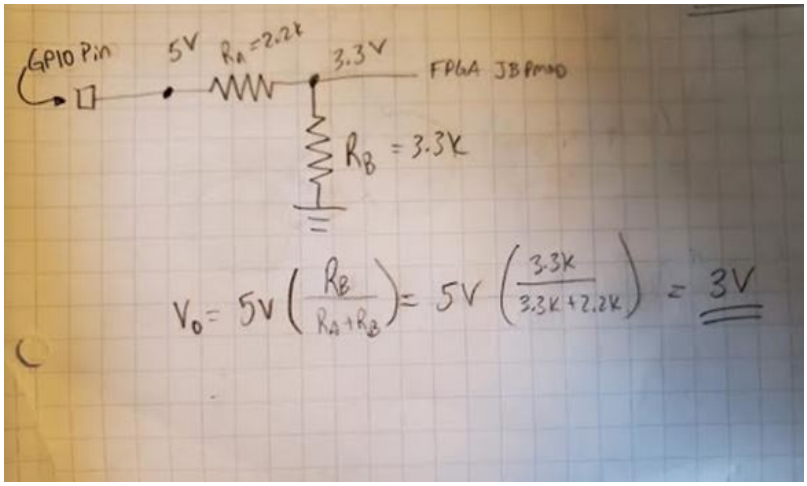


Figure 11: Resistor Divider Calculation

The next thing I needed to do was verify that the VGA timing for the front porch, back porch and sync were working properly, which can bee seen in Figure 12. The HDMI signals were taken directly from source (3) and past labs.

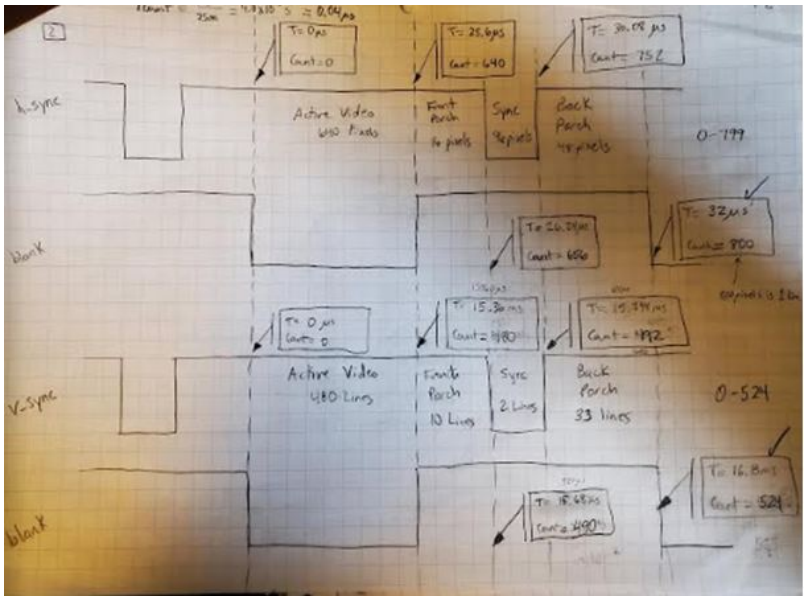


Figure 12: VGA Timing

The next calculation I found myself in need of was to determine the alien hitbox for the alien which can be seen in Figure 13. The yellow dot is the location of the alien column

and the alien row. Therefore, the hitbox is the bolded black box around the alien. So the far left is alienColumn-5 and the far right is alienColumn+5.

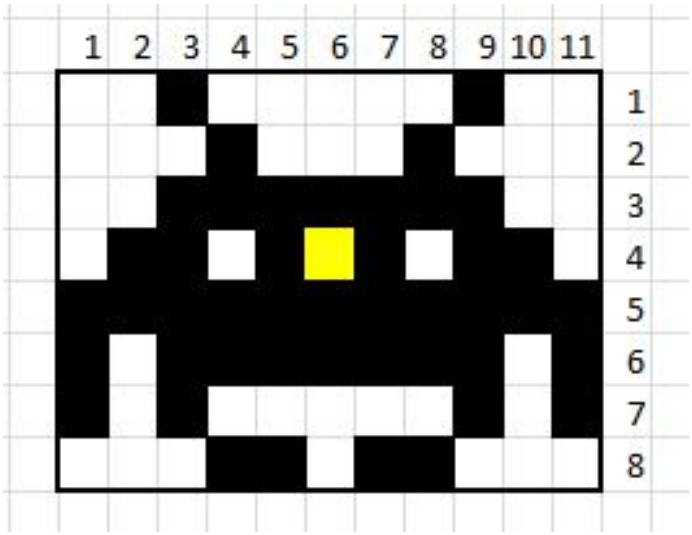


Figure 13: Alien Hitbox

Lastly, one of the calculations that I'm very proud of is using a clock divider to slow the movements of the alien, bullet, and player down to manageable speeds (1). For example, if I ran the aliens movement on the clock speed of the Nexys Video, that would be impossible for the eye to see. The internal clock is 100 MHz. Therefore, we have to derive a slower clock for the aliens to move off of. At first, I thought about using the Clocking Wizard IP, however the slowest clock it could generate was around 4 MHz, still far to fast for my application. I needed a clock around 25 Hz.

What I did was use variable of X bits to determine how clock speed. It's easier to explain by example, so let me start that. I decided that I wanted the aliens to at 25 Hz.

$$\frac{100MHz}{2^x} = 25Hz$$

$$4000000 = 2^x$$

$$log_2(4000000) = log_2(2^x)$$

$$21.93 = x$$

The results here show that we need to have our 100 MHz clock counter from 0 up to 3995653.366 ($2^{21.93}$), and every time it gets to the top toggle an enable for the clock. This will get us a 25 Hz clock. Obviously, we can't have a 21.93 bit number so we can choose to round up or round down. I chose to round up to a 22 bit number. This would leave us with slightly slower clock speed at 23.84 Hz. But this is totally fine and we don't need something so exact. It just has to be slow enough so that our eyes can detect it.

This same process was used to determine clocks to control the bullets and used to "debounce" the player movement. Similar to a mechanical button, the Wii Nunchuck would have move left or move right (in my implementation). Obviously, a the reading on

the potentiometer would determine the speed the player ship is moving left or right, but for simplicity's sake, I decided to only move left (or right) when the potentiometer read greater than (or less than) a certain value. This made it easier to control for me. It's either moving left (or right) or not at all. With that out of the way, I found that when I would move left, the player ship would move from the middle of the screen to the left side of the boundary. This would happen because everytime there was a clock rising edge (remember the Nexys video clock is moving at 100 Mhz), the player location would move left one pixel. With how fast the clock was moving, to the human eye, it moved all the way left instantly. To prevent this, I had to use a slower clock (using the exact method described above). Doing this, allowed the player to move left slowly and control the ship. The exact clock speed was 42.66 Hz and is something I tweaked based on feeling.

2.4 Technical Requirements

REQUIRED FUNCTIONALITY:

- Game boundary is present and clearly visible
- Aliens, player, and projectiles may not move outside the boundary.
- Aliens must move across the screen (left to right or right to left) then descend a row repeat, when alien's reach bottom of screen they will go back up to the top
- Player can move left and right (not up or down)
- Player will shoot when a button (C button) is pressed and the projectile will move upwards and either go to top of boundary
- Use a Wii Nunchuck Keyboard for player control (connected with Arduino)

Behavior:
The system should display a game boundary (white rectangle) with aliens moving (left and right) across the screen. The player should be able move left and right and shoot bullets. The bullet should travel straight up from the point it was fired and no alien collision should be present at the moment.

Testing:
Simulation will be used to verify simple concepts, but the main test will be displaying it on a monitor to see it function

B-LEVEL FUNCTIONALITY:

- Player can switch (Z button) between two different types of shot (single shot and burst shot)
- NEW: Add Score counter

Behavior:
The system should be able to let the player switch between different types of ammunition.

Testing :
Simulation will be used to verify simple concepts, but the main test will be displaying it

on a monitor to see it function.

A-LEVEL FUNCTIONALITY:

- Individual Alien collision (when bullet makes contact with Alien, Alien should disappear)

Behavior:
The system should behave much closer to the actual space invaders game. The player should be able to shoot alien's and have them disappear if touched by a bullet.

Testing:
Simulation will be used to verify simple concepts, but the main test will be displaying it on a monitor to see it function.

2.5 Bill of Materials

The materials required for this project are as follows:

1. Nexys Video FPGA
2. Arduino UNO
3. Jumper Wires
4. (4) - 2.2kΩ Resistors
5. (4) - 3.3kΩ Resistors
6. Wii NunChuck
7. HDMI Cord
8. Breadboard
9. Monitor (with HDMI Input)

3 Implementation

3.1 Milestone I

I was not able to achieve the technical requirements to complete Milestone I, however, I do think that I am close.
Please refer to Figure 6 - 9 for the Datapath and Control unit used for this milestone.

I changed the way the alien's are showing up. Originally, I thought I would have 25 aliens (5 rows of 5) and then as I thought more and more about the implementation I figured out that would be a huge undertaking. So I decided to tone it down to 15 aliens (3 rows of 5). Originally, I thought that I would have an alien0_row, alien1_col, and alien2_en parameter for each alien. That way I could track each alien individually. However, as I sat

and thought about it and plotted everything in excel, I discovered that I would only need 3 row parameters to keep track of all the aliens. There is a top of aliens, a middle row of aliens and a bottom row of aliens. This allowed me to keep track of each alien with less signals. In fact, as I'm writing this, I could probably cut down even more on the signals and have 5 columns. That way, each alien can be identified by one of three rows, and one of 5 columns. That would cut down my signals further. I will look into this.

This was something that ate up a lot of my time. I kept thinking, then rethinking things and never really started moving forward on one idea. This week, I'm going to be working more on this and I hope to get it nailed down early-mid week this week.
The big chunk of work that I completed was obtaining a Wii Nunchuk controller and writing code to decode (over I2C) the values from the accelerometer and potentiometer from the controller. I also set up the resistor divider network as my "poormans" voltage level shifter. This will work with no issues because the speeds we're dealing with are so slow. If we started to get up to 1 MHz, we'd probably need to use a transistor level shifter or just buy a dedicated IC for it. But for human inputs on a controller, we're seeing things like 1 Hz – 5 Hz(1 input per second – 5 inputs per second) for normal usage. So a resistor divider will be perfect. See the Calculations section for the resistor calculations.

Unit Test Plan:

My test plan isn't all that formal because I have a screen to test things on (due to the nature of the project), I will be able to see what is wrong and what isn't. So I haven't done any simulation testing as of now.

I did test out the Arduino code to make sure that I am correctly reading values from the Wii Nunchuck correctly. I was able to find a library that helped with most of this, but to get the outputs I needed, there were some modifications that were necessary. I decided to take a video and upload it to my google drive which can be found at this link:

https://drive.google.com/file/d/16tmsaV8-UFDhHS6zvEOEDo7J1TTvMIEI/view?usp=sharing

The Wii nunchuk runs off of 3.3V (which is powered off the of the 3.3V Arduino power output pin). It uses I2C to communicate data values:

- **AnalogX**
- AnalogY (Don't need)
- AccelX (Don't need)
- AccelY (Don't need)
- AccelZ (Don't need)
- **Button_Z**
- Button_C (Didn't use)

This blog helped out a lot (https://todbot.com/blog/2008/02/18/wiichuck-wii-nunchuck-adapter-available/). I am determined to figure out the logic of the library, which is something that there is very little documentation on. So it's eating up a lot of my time and

it's probably something I should forgo right now, but I would like to include in my report at a later date. Additionally, I was able to modify the code to output high or low signals to input into the FPGA based on reference (4). Figure 14 shows my test setup during Milestone I:

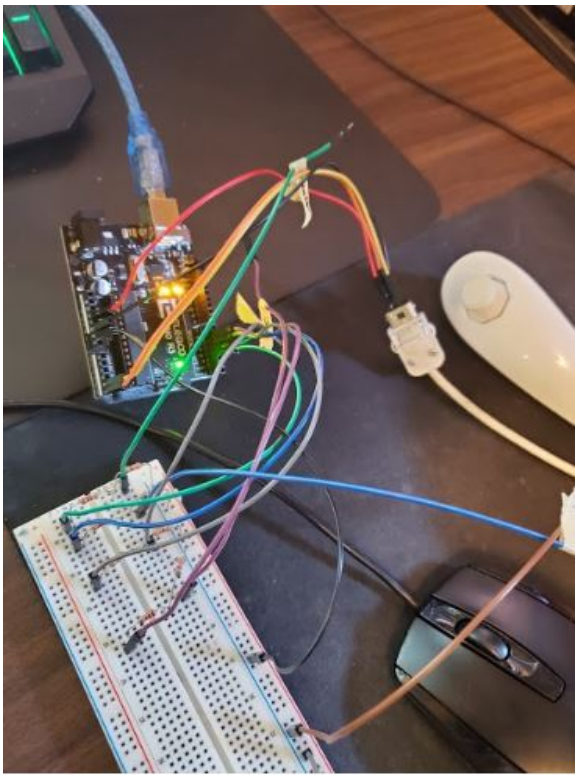


Figure 14: Milestone I Setup

3.2 Milestone II

The datapath and control units are referenced in Figure 6 - 9 above.

The biggest change that I incorporated into my design was to add another Finite State Machine (FSM). I now have one to move the aliens across the screen. The new one that I added moves the bullet up the screen and is in charge of individual alien collision. One way that I could have gone about it, and probably the most intuitive way, is to just have a column, a row and an enable for each individual alien. There isn't necessarily anything wrong with this approach, and the logic is pretty straight forward. The reason I chose not to go down this route is that it was so many signals. For my game, I chose to have 15 aliens, which would mean that I have 15 row signals, 15 column signals, and 15 enable signals for a total of 45 signals.

Instead, I chose to make three row signals (AlienTopRow, AlienMiddleRow, and AlienBottomRow), and five column signals (alien0_column, alien1_column, alien2_column, alien3_column, and alien4_column). To make sure that I have individual alien collision, I kept the 15 enable signal. This approach knocked my total signals from 45 down to 23, something a little more manageable. This also had some nice benefits later on in the design when I was

detailing the logic for the FSM to control the bullets. I didn't need as many states or logic variables to keep track of collisions, so I am very happy that this was the approach I took. During this time is when I developed the FSM shown in Figure 9. Figure 15 shows the test run results. The aliens are moving (and hitting boundaries), the bullet is firing, and bullet collision with the top boundary is working correctly.

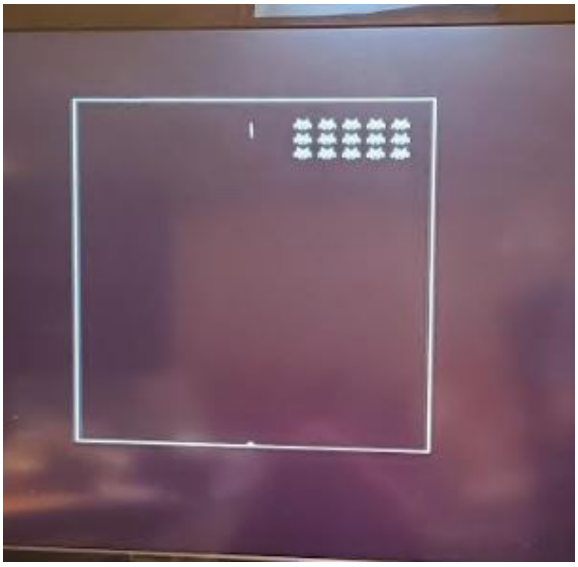


Figure 15: Milestone II Test Run Results

3.3 Final Implementation

The final implementation can be seen in Figure 16 below. The ship has been changed to something much more impressive than a little triangle ship. A score counter was implemented as well as a "You Win!" state when all the aliens are hit, this is shown in Figure 17.

Additionally, the link to the final video project working can be found below:

https://drive.google.com/file/d/1B-FDbZy10yznWIRKXLwLYr0Urf3WHh24/view?usp=sharing

Figure 16: Final Implementation Test Run Results

Figure 17: Final Implementation Test Run Results

4 References:

(1) - User: SonicWave. “Setting Up Slow Clocks” StackOverflow Web Forum, March 6, 2013. https://stackoverflow.com/questions/15244992/vhdl-creating-a-very-slow-clock-pulse-based-on-a-very-fast-clock

(2) - Falkinburg, Jefferey, ”CSCE 436 Lecture 11”. Advanced Embedded Systems, Feb 7, 2020, University of Nebraska-Lincoln. Microsoft PowerPoint Presentation.

(3) - Lab 1 - Video Synchronization
Jeff Falkinburg - https://cse.unl.edu/ jfalkinburg/cse_courses/2020/436/lab/lab1/lab1.html

(4) Gabrielbianconi/arduino-nunchuk
GabrielBianconi - https://github.com/GabrielBianconi/arduino-nunchuk

5 Appendix A: Running the Project

To set this project up, one will need the arduino source code and the VHDL source code in my bitbucket repository.

Additionally, you will need to hook up some hardware. The first step is to connect the the wii nunchuk up to the arduino, the arduino to a resistor divider and the resistor divider to the JB PMOD connector. The pinout of the wii nunchuk is shown in Figure 18 and the schematic for the circuit is shown in Figure 19.

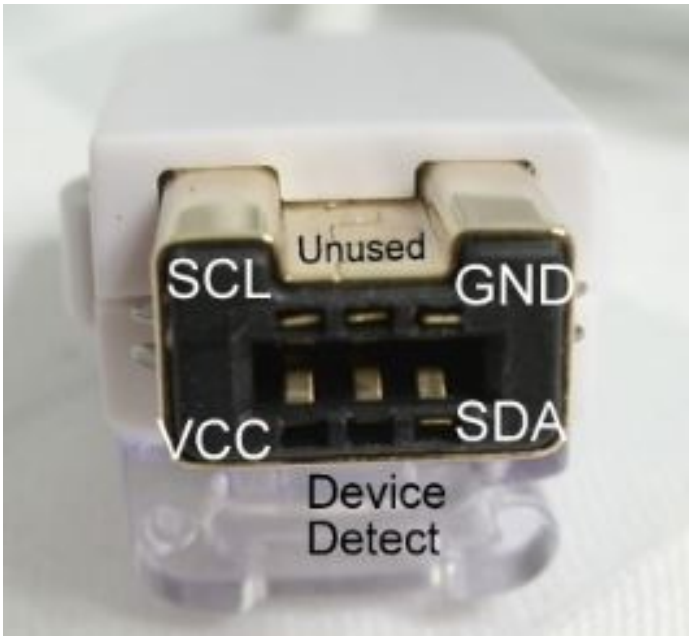


Figure 18: Wii Nunchuk Pinout

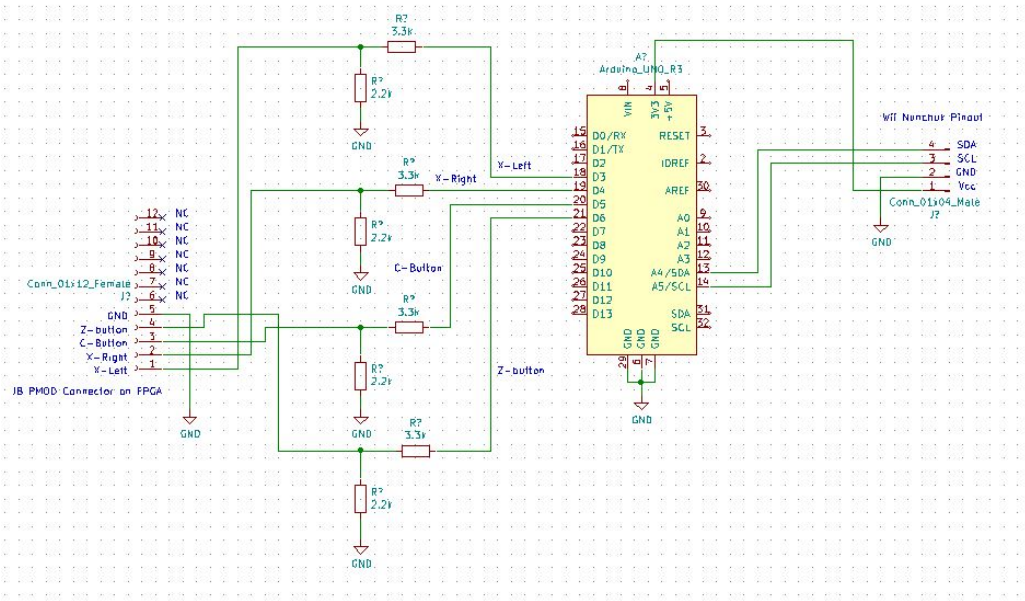


Figure 19: Project Setup

6 Appendix B: Git Repository

I understand that I must provide the faculty member with a BitBucket repository containing four directories. These four directories will be included in my current CSCE 436 repository in a folder called FINAL PROJECT. SOURCE will contain all the project files (including all intermediate files generated by the compiler), PRESENTATION will contain your final power point presentation, DEMO will contain a mpeg, mov, avi, etc. of my project in action along with any documented tests, and REPORT will contain your written report.