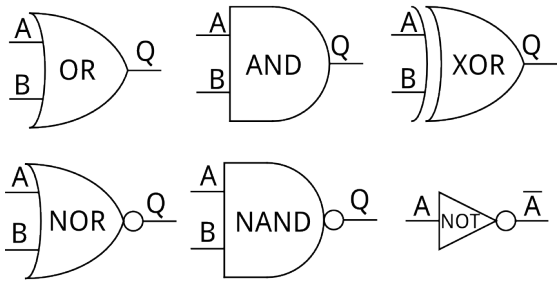
Space Invaders on an FPGA

CSE 185E Final Paper using IEEE

Gary Mejia

A lab report for CSE 125
Final Project on a Space Invaders
implementation on iCEBreaker FPGA board
For the CSE 125 Teaching Staff and
those curious about FPGA project design flow



Computer Science and Engineering
University of California, Santa Cruz
Rachel Carson College

Space Invaders Lab Report

CSE 185 Final Project
Gary Mejia

Abstract—Report the results and design process of a Space Invaders implemenation on an FPGA. Project is located at the following Git repo: https://github.com/colbarron/spaceinvaders

Index Terms—FPGA, Space Invaders, field programmable gate array, game design, Verilog, testing

1 Introduction

As CSE 125, Logic Design with Verilog, presented dropping the final exam in exchange for a final project, I immediately accepted. Field programmable gate arrays or FPGAs have sparked my interest after completing the first couple of CSE 100, Intro to Logic Design, labs. After completing CSE 100 lab 6, a jumping game, I wanted to implement another, slightly more difficult, game. Space Invaders is a famous retro game that incorporates simple input and output to create an entertaining game. This project had the option of it being a duo project, to which this one is. Edwin Rojas helped with creating all the display elements and the enemies.

2 What is Space Invaders?

A burning question some readers may have is: ”What is Space Invaders?”. Released in 1978, Space Invaders was created by Tomohiro Nishikado using custom hardware. At the time, the hardware was the considered the best. Nishikado had Japanese corporation Taito published the game.

Space Invaders is a two dimensional ”shoot-em up” where the player controls the bottom ship. The aim of the game is to defeat all the aliens. Aliens randomly shoot at the player. If the player gets hit, they lose a life. Once all lives are lost or the aliens land on the ground, the game is over.

2.1 Why Space Invaders?

The question I pose is here is for me. Why choose Space Invaders for this project? Since the final project for CSE 125 is open ended, making games have been an easy out in a sense. Game design always has a clear end goal. A research article written by two teachers, Anne Sullivan and Gillian Smith, titled ”Lessons in teaching game design” describes the use of game design in teaching introduction to programming. Students are more open to begin programming a game simply due to the lack of background knowledge needed in class. This concept applies to me as well. Space Invaders is more known rather than an audio processor.

3 Hardware

Hardware projects differ from software projects due to the fact that physical hardware is required for hardware projects. Who would have guessed? This section will go into detail of what hardware is used and its purpose in the project.

3.1 FPGA Board

CSE 125 had the number one goal of using an open source tool chain with open source hardware. According to Professor Dustin Richmond, he wanted to not lock students to Xlinix Vivado, therefore locking all Apple Silicon users from the class in lab. For the development board, the 1bitsqured iceBreaker FPGA development

board is used. It contains a Lattice iCE40UP5k FPGA chip. For peripherals, it has three peripheral module ports or PMODs; where one contains a breakaway input and output board with three buttons and five light emitting diodes. On board, one button and two LEDs are present for basic I/O.

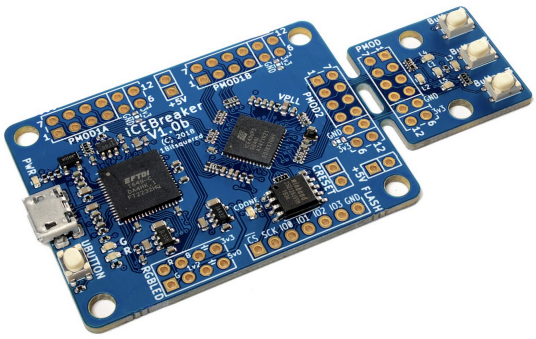


Fig. 1. iCEBreaker FPGA development board

3.2 HDMI Display Module

Space Invaders requires external display to a monitor. The 1bitsquare team has created a HDMI display module for the board. It supports up to 1080p display with an HDMI connection using the VGA protocol. Due to my current logic design experience, I opted to display at 480p. This allows leeway in logic design as logic now does not have to be highly optimized. Physically, the display PMOD takes up all the remaining PMOD ports, thus limited me to either using the included break away PMOD or use something else. Since the controls of Space Invaders only involve moving left, right and shooting, I opted to keep the breakaway PMOD.

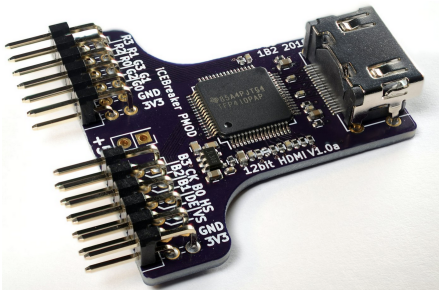


Fig. 2. HDMI Display Module

4 Software

Despite being a hardware project, software is still used to aid in design. This section will cover the choice of language and tools used in the project.

4.1 SystemVerilog

FPGAs require hardware description languages to describe their intended function. Programming languages are widely known such as C/C++. Where hardware description langauages and programming languages differ is in their output. Generally programming languages compile to machine code, like x86 assembly. Hardware description ”translates” to logic gates, where I use ”translates” as a term to simplify an entire complex process of creating hardware. For CSE 125, the hardware description language taught is SystemVerilog and thus the final project, if on an FPGA, had to be in SystemVerilog. SystemVerilog is to Verilog like what C++ is to C. That is to say, SystemVerilog takes a lot of Verilog syntax and extends it with additional features. Many of these features allow us to organize code based of their use in the project, such as differing combinational logic and sequential logic.

4.2 Synthesis, Route, and Placement

How does hardware description language become a circuit? This is an excellent question that is explored heavily in graduate school. Generally hardware description code goes throught the process of synthesis, routing and placement, which is the ”translation” to produce a circuit that I alluded to in the previous section. The tool used for this job is Yosys and and its subproject nextpnr. Sythesis is done by Yosys and the rest is done by nextpnr. These are open source tools that support the iCEBreaker board, fulfilling the goal of making the entire tool chain of CSE 125 open source.

4.3 Circuit Simulation

To test circuits before programming them onto the FPGA, simulation can be done to catch some bugs and view intended behavior. The

programs used in CSE 125 to accomplish this tasks are Verilator and iVerilog. Both take test benches and create waveform files. These wave forms are viewed in a program called GTK-Wave. All three programs are open source. Two simulators are used since simulators themselves are not perfect. For example, Verilator can not catch X (unknown) or Z (disconnected) signals. With two, there is less chance of a bug not being caught but the chance is never zero.

5 Starting the Project

The project began with designing the state machine diagrams for the player and taking some of the modules done and provided for CSE 125 labs. Our counter module is taken from lab 2 of CSE 125 and slightly modified to take in different reset values and increments. Below is the list containing the rough draft of the player state machine diagram.

- Idle -
  The state where the player is not moving and alive.
  If not hit and pressing the left button only, the state transitions to moving left. If not hit and pressing the right button only, the state goes to moving right. If hit and the player life counter is not zero, go to hit but alive. If hit and the player counter is zero, go to hit and dead.
- Moving Left -
  The state where the player is moving left.
  They remain in this state if the left button is held and the player has not hit the boundary. If both left and right buttons are held or the left button is held at the left boundary, go to the idle state. If the left button is let go, the player goes to idle unless the right button is also immediately held, then it goes to the moving right state. If the player is hit and the lives counter is not zero, go to hit but alive else go to hit and dead.
- Moving Right -
  The state where the player is moving right.
  They remain in this state if the right button is held and the player has not hit the boundary. If both left and right buttons are held or the right button is held at the right boundary, go to the idle state. If the right button is let go, the player goes to idle unless the left button is also immediately held, then it goes to the moving left state. If the player is hit and the lives counter is not zero, go to hit but alive else go to hit and dead.
- Hit but Alive -
  The state where the player was hit but still had lives to spare. To leave this state, the player had to press the shoot button and be sent to idle.
- Hit and Dead -
  The state where the player was hit and had no lives left. This state is an ending state and the only way to leave it was to restart the board.

Additionally, my partner took the initiative on working on the display controller logic. Since video projects are common on FPGAs, we found several display controllers online to which we took one from the site called Project F by Will Green. Compared to the video controller made in CSE 100, this one is far more concise and to the point since actual Verilog syntax is used. Additionally, the same site had several basic display test modules to test our display PMOD and see if we had somehow destroyed the board. With these items, we began implementing the player.

6 Debugging the Player

To begin debugging, I made a simple test bench which is taken from the test bench used to test our shift registers for lab 2 of CSE 125. The entire purpose of the test bench is to test that the player has not lost a state nor gone to multiple states and to produce a waveform. This is due to us deciding to use one-hot state encoding, where each state is encoded to its own flip flop.

The test bench proved invaluable to the debugging process; however there was the major issue that Verilator did not catch X's. Whenever you have sequential logic, no matter what it is

initialized too, at the very first clock cycles, flip flops will be unknown and thus X's. Verilator never passed the test bench, no matter how many fixes I tried, thus we relied a lot on the iVerilog waveform.

An interesting problem that persisted with us to the end was caught in the player waveform: reset. All reset is in this project is the onboard button. That button has the job of resetting all flip flops in use. However on the test bench I had the problem of needing to decide what to input into the reset port of the player. One of the provided modules for CSE 125 is a resetting module used for creating a reset signal in test benches. Additionally, I had a test input of reset as well, which many of the provided test benches have as well. Whenever I would input **reset_l** | **reset_i**, where reset_l is my own reset test input and reset_i being the output of the reset generator, the player waveform would output useless data. Once I went against all previous test benches and changed the or operator to the and operator, the waveform would produce useful data.

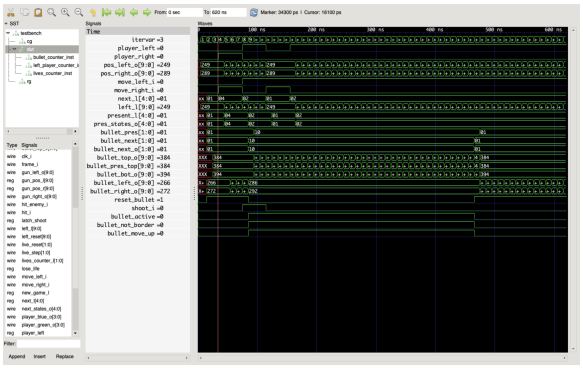


Fig. 3. Player Waveform from GTKWave using iVerilog

Getting the player to display did not take long. However in getting basic player display we ran into the first reset problem. In all our CSE 125 labs, the on board button is used for resetting is unsafe and asynchronous to the clock. To fix this, the button is ran through two flip flops to synchronize it to the clock and through an inverter since its negative polarity. We first used this synchronized signal as our reset but nothing ever showed up, however using the button directly did work. More reset issues would rise at different modules, this was just the first. Thanks to the test bench, the player did not require much debugging after initial display. All changes done here are minor, such as where the player is placed and adding on detail. What did take a while to debug was the bullet firing.

6.1 Debugging the Bullet

Test benches are helpful tools however tools only help in solutions. They are not all mighty solvers onto themselves. We added the bullet state machine into the test bench to track its state and movement on the waveform. For the entirety of the debugging process we saw the bullet moving correctly though its states and line of movement. It correctly stored its horizontal position and kept it for the entirety of its flight. However we saw bizarre behavior on the monitor. One example would be the bullet only flew during a button press; to fly the shoot button must be held else it would return to its stationary position. Another example would be where the bullet never stopped at the designated top boundary and would roll over in the monitor. Fixing these issues were not too bad onto themselves. Most of the time, the reset logic for the bullet movement counter would be off by a term causing the bullet rollover. We did use a finite state machine with two states: idle and flying. The bullet would go to the flying state if the shoot button was pressed in the idle state. To return to idle, the bullet either had to hit an enemy or hit the top boundary. During debugging we debated in removing the state machine since a single flip flop would be needed for the states, however we decided in keeping it. Once the bullet worked as intended, we moved on to the hardest part of this project: enemy movement.

7 Enemies

Enemy movement is actually the main reason I decided to pick Space Invaders over other games. Pacman, for example, only four enemies but with rather complex artificial intelligence. In Space Invaders, all enemies do is move left

to right and randomly shoot, which to mean at the beginning of the project did not sound hard to implement. Those are famous last words. To begin the enemies, we decided to just have one enemy module and combine fifty-five of them to create the entire enemy array seen in the actual game of Space Invaders. The state machine for a single enemy at the beginning was simple and is as follows:

- Move Right -
  The state where an enemy is moving to the right.
  If it hits the right boundary, it descends and transitions to move left. If the enemy is shot, it goes to the dead state and disappears from the screen.
- Move Left -
  The state where an enemy is moving to the left.
  If it hits the left boundary, it descends and transitions to move right. If the enemy is shot, it goes to the dead state and disappears from the screen.
- Dead -
  The state where the enemy is dead.
  It will be gone from the screen and can not shot nor be shoot.

The reason for the lack of idle state is due to the fact that enemies will start moving to the right immediately. Upon finishing the state diagram for the single enemy is where I started to have doubts on the scope of the project. Sure the lack of an idle state for the enemy makes sense due to them never staying stationary outside of the dead state but how would we control the entire game without a game state machine? Implementing a game state machine would be rather simple but the FPGA only has so many resources to throw around. My partner and I decided not to worry about resources until the tool chain told us to, thus we went on to draft how to link up fifty-five enemies together.

7.1 Enemy Linking Ideas

Linking enemies became a clear goal after playing two rounds of Space Invaders. In most implementations each row of enemies move at different times than others but some the whole block of enemies move at once. Since movement is shared, linking enemies would be an effective way to simplify input and output. The first major idea was to create a column of enemies and then instantiate eleven linked columns. In creating the column, already we hit a roadblock. How do we link things? A simple data structure to link objects is a linked list; in our case we decided a doubly linked list. At the beginning we thought it would be relatively simple. Since we are not creating an entire doubly linked list data structure, we do not need to worry about the operations for it. Yet in that we found another issue: how do we point to nothing? The head and tail of a doubly linked list point to a pointer to nothing yet a pointer to nothing is already an abstraction. Fundamentally, we had been using relatively high abstractions to solve our problem which was a common theme in trying to link the enemies. In our enemy module parameters we added a bus which just acted as an identification number for the ship but we have no way to use the number effectively. Both me and my partner have ample C/C++ experience so at the time we thought the use of the id number could lead to use affecting a single enemy however that is a big abstraction onto itself as well. Another bigger issue came from me trying to solve the problem of differentiating the deletion of an inner column of enemies to that of an edge column of enemies. Below are two examples of how defeating different columns affect the enemy movement.

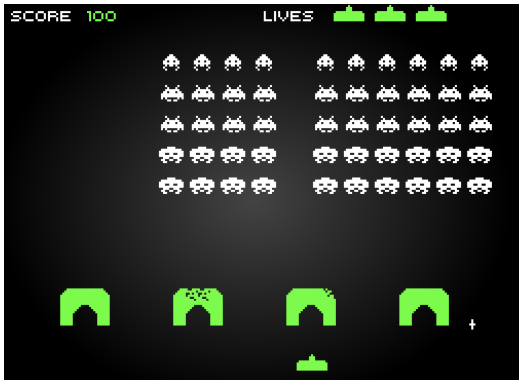


Fig. 4. Inner column of enemies is defeated thus boundary is unaffected

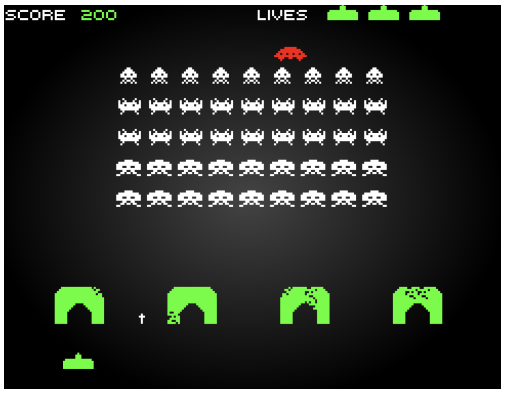


Fig. 5. Left most column of enemies is dead thus enemies have more area to cover before hitting a boundary

We thought use of pointers could remedy this problem. Two counters would point to each left most and right most column and move depending on a signal sent out from the column. Again we thought it was a simple enough implementation yet the case of knowing when to jump a dead column became too hard to solve in two weeks time.

A total of three complete days were spent on just drafting solutions for enemy movement without any programming being done. We ran our idea with Professor Richmond for some guidance to each we received another good approach, use memory. Although we have not started collision logic we had the idea of giving each enemy the bullet coordinates outputted by the player, however we failed to account for the high fan-out this would require. By using memory, we remove the fact that each enemy is a hardware, thus freeing up resources, and take advantage of large amount of time we have in each frame cycle relative to our operations. The operations done in-between frame cycles are quick and simple, thus accessing each enemy in memory would not pose a timing issue. We attempted this solution by writing basic programs to write initial memory files but did not get far due to timing constraints. With not much time left, we decide on just focusing on getting one enemy to move left and right. At best maybe to shoot.

8 Results

How did it turn out? In the end we came up with a comprise: the player can not die however it must defeat the enemy to win; the enemy will move faster each time it hits the border. With one enemy, collision became simple. Of course reset became an issue here as well, a huge problem of not resetting the enemy's position when shot or at its end point. We never got the reset button to work with the enemy itself. One idea I had was to look on how global reset worked in CSE 100 labs. In those labs we were given each a clock module is a button input for reset. On further investigation reset is handled by a module called "STARTUPE2". Googling this module lead me to a Xilinx page which meant we could not use it directly as our FPGA is from Lattice. Thus I suggested to use the CRESET port on the board itself. From my previous experience, onboard reset pins needed to be shorted to reset so with a leap of faith I got a jumper cable and shorted the pins. Thankfully it reset the enemy correctly. With that we decided to call the project finished and add any finishing touches. My partner just added in a win screen and lose screen depending on if the enemy landed or got hit.

9 How to use this Project

To run this project you will need a couple of things. For hardware you will need the 1bit-squared iCEBreaker FPGA board with an Lattice iCE40 FPGA and its HDMI display PMOD. This project is done on MacOS thus the installation of all software requires Homebrew and the following commands:

```
brew install icarus-verilog
verilator gtkwave

brew tap
ktemkin/oss-fpga

brew install
--HEAD icestorm yosys
nextpnr-ice40
```

Cloning the project with the following command:

```
git clone
https://github.com/colbarron
/spaceinvaders
```

Move to the **top_module** directory and run the command: **make prog** with your board plugged into your computer and plugged into a display monitor. With that you should now be able to play the game.

10 Conclusion and Reflection

Upon completion of this project, I greatly underestimated the complexity of Space Invaders. Of course with open ended projects, one enters them with a lot of hope and excitement which leads to heavy optimism. My project proposal included multiple levels with the win condition of overflowing the score counter. Funny enough, the win condition of overflowing the score counter was only added because I thought the project was easily enough that the game itself would be too hard to lose. The look Professor Richmond gave us upon the proposal goals should have told us that we were too ambitious.

Talking with other CSE 125 students lead me to realize each project has different challenges. Some students had the challenge of communication between their boards and their PMODs, others had the challenge of resource management. Our challenge was resource management and lack of abstractions. Every couple of hours we kept telling ourselves: "We would be done by now if we were using C or C++". This project helped me realize how high in abstraction programming languages are, including C/C++. In my introduction to C class I remember absolutely hating pointers and wished they never existed. However in this project every day we wished we could use C pointers to solve all our problems. Programming languages are a marvel that I feel many students take for granted. However the main goal of a programming language is to talk with hardware. At the low level of hardware there are many aspects to appreciate. For this project, if we had taken into account the use of memory modules, the speed of our operations are unreal when I think about it. Despite the operations being fast, the challenge would be to effectively pipeline the system to work properly.

How do I feel at the end of this project? Truthfully a bit disappointed with myself. Sure the project is complex. Space Invaders is a complex system. I had only two weeks to create a complex system that took years to develop originally. However it is not all disappointment. I do feel more confident in my ability to plan out a project, thus if given a quarter to do this project I feel I would have the results I wanted. Additionally I learned several things, the major one being start simple. Simplicity is easier to debug and understand.

I will admit, I did cause a lot of the over complexity in this project. Many of my ideas did cause complexity such as adding pointers to each enemy. However most computer systems are already complex. The data sheet just for the Lattice FPGA is 52 pages long. At least, this project services as experience in working with a complex system and I hope to work with more of them in the near future.

Acknowledgements

I would like to thank Dustin Richmond and the CSE 125 teaching staff for giving me and others the opportunity to do a final project. As a poor test taker, I would rather attempt to reach a goal over the course of two weeks instead of taking a three hour exam. I have encountered bizarre problems and came up with the most shoddy yet I will self admit, funny, solutions for the first time in class. All readings that I have done, have served the purpose of guidance and inspiration for solutions. Most importantly of all, I am grateful for the prize of keeping the board even if the project did not produce the wanted results. Hopefully in the future, the project will come to a fruitful end.

Additionally I would love to give a thanks to my project partner, Edwin Torres. He took charge of creating the enemy and all display logic. Torres helped narrow the focus of the project and assisted in helping cut out content to make the project doable in the small time frame. The idea that "you shoud not reinvent the wheel" is something he took to heart in this project which is an amazing quality to have in engineering. Often solutions are out there and

you have to look for them, which is what he did and did well.

REFERENCES

[1] "The Centre for Computing History." Centre For Computing History. Centre For Computing History. Accessed March 10, 2023. http://www.computinghistory.org.uk/det/47162/40-Years-of-Space-Invaders/.

[2] Sullivan, Anne, and Gillian Smith. "Lessons in Teaching Game Design." Proceedings of the 6th International Conference on Foundations of Digital Games, June 29, 2011, 307–9. https://doi.org/10.1145/2159365.2159421.

[3] "Icebreaker FPGA." Crowd Supply. Crowd Supply. Accessed March 19, 2023. https://www.crowdsupply.com/1bitsquared/icebreaker-fpga.

[4] Green, Will. "Beginning FPGA Graphics." Project F. Project F, March 2, 2023. https://projectf.io/posts/fpga-graphics/.

[5] "STARTUPE2." AMD Adaptive Computing Documentation Portal. Xilinx. Accessed March 22, 2023. https://docs.xilinx.com/r/2021.2-English/ug953-vivado-7series-libraries/STARTUPE2.