ECE 385

Fall 2019

Final Project Proposal

SPACE INVADERS PRO
スペースインベーダ

Pouya Akbarzadeh and Patrick Stach

Section: ABC

Lab TA: Vikram Anjur, Yuhong Li

Idea & Overview

We propose to re-design and implement the beloved game of Space Invaders using the FPGA as our System-on-Chip.

The original space invaders game was a Japanese shooting game made in the late 1970s. It was heavily inspired by Star Wars. The idea of the game was simple, there are aliens coming and you have to defend against them. The game is considered to be a fixed shooter in which the player, you, controls a laser by moving it horizontally across the screen. The aliens or invaders will also move horizontally but will slowly work their way up toward you. You have to shoot the aliens before the aliens shoot your base. As you survive longer the game should get more difficult with the aliens attacking you faster and more rapid shooting.

Our design will also include a NIOS II CPU for the purposes of interfacing with the USB keyboard as seen in lab 8.

The design will actually be reusing a lot from lab 8- the USB interface with NIOS, the VGA Driver for syncing with the vga monitor, the color mapper which draws colors to the screen, and the collision checking which is essentially a more advanced “ball.sv” where not only are we checking collisions at edges so objects don’t go on screen, but also collision checking for every instance of an enemy, every instance of a bullet, and the instance of you (also the title screen).

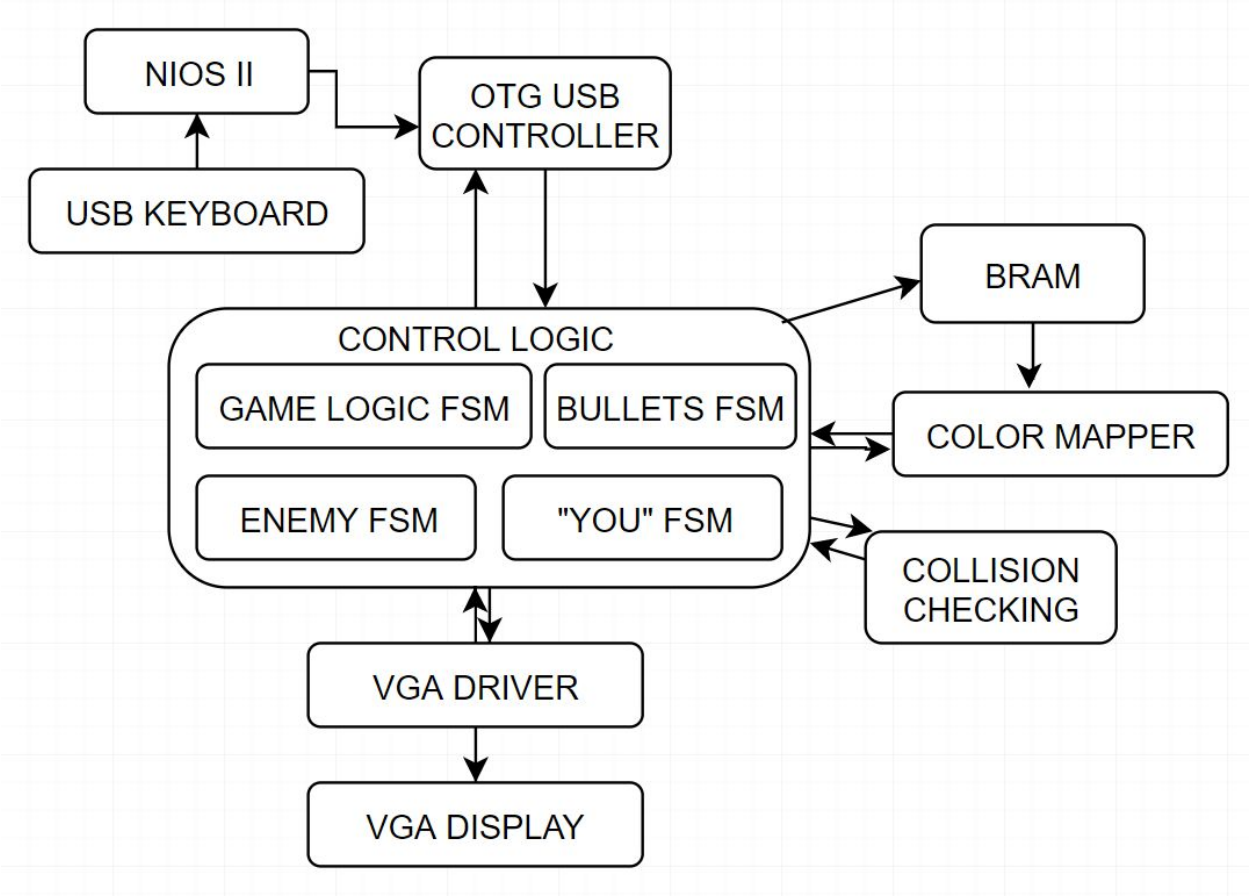
For the title screen, we are planning on having a cursor that is controlled with arrow keys and will constantly be collision checking, it will “select” when you press enter. If there is a collision with a specific choice it will execute it else nothing will happen.

Some things that we will be adding onto are animations for the color mapper (from sprites stored in BRAM), and four separate finite state machines. In previous labs we have only implemented a single state machine and only one instance of it, now we will have four different state machines

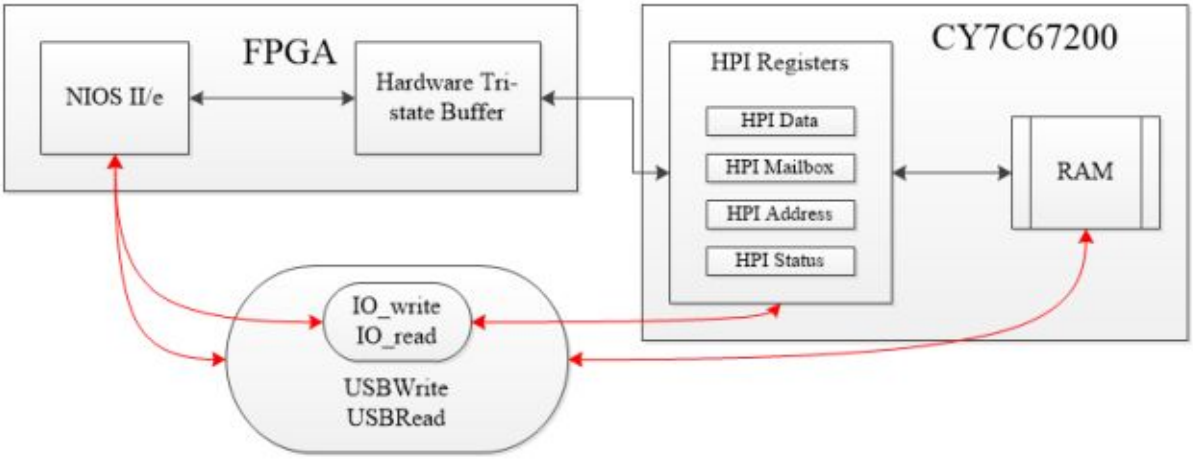
interacting with each other, in addition to instantiation multiple instances of them (i.e. 7 enemies at once following the same state machine logic)

More details of the specific modules will be discussion in the block diagram

Block Diagram



The only thing that we are planning to use the NIOS II for is for interfacing with the USB keyboard. From the block diagram given in lab 8



The NIOS II will interface with the OTG USB controller CY7C67200, writing and reading from the chips RAM, and reading and writing from the HPI registers. The FPGA, or control logic in our block diagram, will read and write to and from the HPI Registers on the OTG USB controller.

For the modules we need in platform designer:

Clk_0: The clock module, 50 Mhz from the FPGA

Nios2_gen2_0: The Nios II processor used for communicating with and processing usb communication

Sdram: we store the software program on here, specifically the files for interfacing with USB protocol (have the IO_READ, IO_WRITE, USB_READ, USB_WRITE functions in addition to many more)

Sdram controller: Need this to interface with the bus because the way the CY7C67200 uses row/column addressing, and we need to constantly refresh data

Onchip_memory2_0: Smaller than SRAM but faster so we can store smaller bits of data here.

Sdram_PLL: generates clock that goes into SDRAM which is delayed 3ns to allow outputs to stabilize

Jtag_uart_0: USB protocol communication through JTAG

In addition to various PIO blocks corresponding to the various signals we need for operation with CY767200:

keycode
Otg_hpi_data
Otg_hpi_address
Otg_hpi_data
Otg_hpi_r
Otg_hpi_w
Otg_hpi_cs
Otg_hpi_reset

Our control logic is divided up into four finite state machines:

Game logic:

Controls things such as lives, keeping track of time, score, difficulty, level number, and transitioning from main menu to actual game, end conditions

"You"

This is the ship that you control and moves left and right, and we need to check collisions with this enemies and their shots, and need to generate bullets to shoot

Enemies

Enemies will have alive/dead state, logic for moving down/ shooting, and be able to generate bullets being shot

Bullets

Theses bullets can either be generated by “You” or Enemies, and we will need to collision check this with them. When there is a collision, the bullet will disappear and the recipient of the bullet will take damage or die

The VGA driver interfaces with the VGA monitor with the VGA protocol and takes care of things such as vertical/ horizontal sync and screen refresh

The color-mapper module selects what colors to put at a specific spot on the monitor, which we will use to draw the ships, enemies, etc. The color-mapper will also be connected to the sprites that are stored in BRAM (we store them in BRAM for sake of speed) which we choose based on control logic

The collisions checking module will monitor collisions between enemies, you, and the bullets and communicate with the control logic when there is a collision detected

List of Features

Base-Line:

The game will be in color.
The invaders will be have an animation
The invaders will shoot back
The invaders will slowly attack and increase in speed

Additional Features:

The player will choose difficulty level (Beginner, Intermdiate, and Hardcore). The harder the difficulty the faster and more shooting the invaders would do. We will consider implementing changing difficulty on the fly.

There will be lights behind the monitor that would react differently based on game play, we will need to learn about using the IO pins and learn about controlling LED strips.

You can use a potentiometer to set the difficulty for the game. If we do that, we will have to research documentation about the having an A to D converter on the board.
Last additional feature would be to add sound (if time allows).

Expected Difficulty

The baseline alone should be a 5/10 for difficulty because the game itself has a lot going on, and the invaders also attack back thus there is some AI that we have to implement. With the additional features not including sound the difficulty should be 7/10 and with sound 8/10. The reason we believe the difficulty should be set at 7 for additional features because of the implementation of analog to digital conversion. And 8 because of implementation of sound is not very well documented and there are a lot of buggy drivers out there.

Proposed timeline

Week 1: Get background, Character Models and Movement (Monitor and Keyboard)
Week 2: Player shooting, Health Bars, Life Bar, Starting Menu, Difficulty Choices
Week 3: Implement lights and Analog to digital conversion (Potentiometer)
Week 4: Attempt to add sound