Lab 4 - Taylor Cowley and Andrew Okazaki

Chapter 2: Game Console and Engine

Section 2.1: Game Console

2.1.a: Diligent ATLYS Board

Xilinx’s *ATLYS* board “is a complete, ready-to-use digital circuit development platform based on a Xilinx® Spartan®-6 LX45 FPGA.” [store.digilentinc.com] it has a large number of peripherals built-in, including switches, buttons, LEDs, several HDMI ports, Ethernet, sound, USB, and a VHDC connector for GPIO. It is a “legacy product”; replaced by the *Nexys Video* product line.

2.1.b: Xilinx Spartan-6 and Microblaze

Xilinx’s *Spartan-6* FPGAs are marketed as a low-cost FPGA, “Spartan®-6 devices are the most cost-optimized FPGAs, offering industry leading connectivity features such as high logic-to-pin ratios, small form-factor packaging, and a diverse number of supported I/O protocols.” [www.xilinx.com] They are therefore good for applications with low-power necessities and high-volume. The *Spartan-6* product line is currently being replaced by Xilinx’s *Artix-7* line.

Xilinx developed a soft microprocessor core called the *MicroBlaze* for their FPGAs. Being a soft microprocessor, it is implemented entirely in the general logic of the FPGA. Hence it can be very customizable for the specific application. It is a RISC-based architecture, and with few exceptions can issue a new instruction every cycle, maintaining single-cycle throughput most of the time.

2.1.c: System Organization

For our game space invaders on the Digilent ATLYS Board we were running our program from the DDR. This enabled us to use more memory because we ran out of space in the BRAMs. The resolution of our game was VGA resolution (640 x 480) pixels and was outputted via a HDMI cable to a screen. To input and communicate with the user we used the UART. With the UART we were able to input from the keyboard commands to either move the tank, fire bullets or any on screen action. By the end of the lab we were able to use an interrupt and buttons to be able to control all user I/O. The left button on the board was able to move the tank left and right button moved the tank right. The tank fired when the center button was pushed.

Section 2.2: Game Engine

2.2.a: Game Engine (Main Game Loop)

In the game Space invaders there are a many objects that we split into separate files. Those files included:

Tank

In the tank file we had full control over the movement which was comprised of panning left and right. The tank would as well shoot a bullet. The bullets position was stored in the tank file and could be shared with the aliens file so aliens could detect a hit. The bullet would as well store a flag to tell if it was alive or dead because there is only one bullet aloud on screen. The tank could be shot as well when the tank was shot there is an animation stored in the file. This was the implementation of the tank file

Aliens

In the alien file the aliens would move. The aliens could move left and right as well when they hit the right or left side of the screen move down. Aliens could fire four bullets and these bullet locations were saved in the aliens file. When the aliens would die an animation would show and the score would be updated. This was the implementation in the alien file.

Bunkers

Bunkers would initialize which would place four bunkers to the screen.

Bunkers would not move but would degrade if they were hit. As well they would be destroyed if the bunkers were hit by the alien. These were the implementations of the bunker file.

Mother ship

In the game a mother ship would randomly show up and move across the top of the top of the playing screen. If the mother ship was hit it would show a random point value and add it to the games total points. These were the implementations of the mother ship file.

Interface

The interface file was tasked with drawing the games interface such as the score and the tank life indication. It would as well draw the game over screen and the win screen. These were the implementations of the Interface file.

Main

In the main file the program was able to call the many functions from these files to build a running game. In main we also implemented an interrupt and the boards buttons.

2.2.b: Meeting the Game Specifications

To pass and meet the game specification we had to pay close attention to our game. Some requirements that we were looking at was that the game started and ran smoothly with no hieroglyphs, flickering and was not slow. We also paid close attention where we were drawing objects and were we were deleting objects. The specs that a little easier to see was that our game performed like the game Space Invaders with a score and tank firing and aliens dyeing.

Section 2.3: Application Programming Interfaces

Space invaders has several files of code talking to each other. This is evident in looking at the .h files. When the aliens move and when the aliens and tank update their bullets, they call detect_collision in bunkers, aliens, and tank to see if they hit them.

Also, every file of code draws pixels to the screen, so there is a utils.h file that has a “draw_pixel” function that performs the proper logic for increasing the resolution to the big screen.

The game engine uses built-in microblaze functions to initialize, activate, and read interrupts from the timer and buttons. When the game is over, either when the player fails or wins, the program calls exit().

Timing and Memory Report

To determine cpu usage, a counter was set to run in our while(1) loop for 40 seconds while no game logic was executing. This gives a base estimate of how often the cpu ticks during that time period. Then the same counter was left running, but the interrupts and all the game logic were also running. The counter with the game logic only went up to 75.03% of the game-logic-free counter, showing 75% idle cpu. So we determine that Space Invaders has 25% CPU usage.

For memory usage, when Space Invaders builds, it outputs this report:

Invoking: MicroBlaze Print Size

mb-size spaceInvadersLab4.elf |tee “spaceInvadersLab4.elf.size”

text	data	bss	dec	hex	filename
50838	1496	6570	58904	e618	spaceInvadersLab4.elf

Bug Report

Lab four brought similar bugs that we faced in lab 3. Some of those similar errors were assigning the correct screen position to our objects. Such as the bunkers, when they were hit it erodes that square of the bunker. Finding the location of that square gave us problems. However we were able to see that we were shifting squares causing bugs within the bunkers.

An error that took us a longer time to figure out was the process of drawing an image for a short period of time then taking it away. This process would happen whenever an alien was shot; an explosion would be drawn to the screen then shortly deleted. In the first place we would always draw black to the screen on the next alien move. This was the wrong way however, because when we would reach the bunkers there would be a black box drawn where dead aliens intersected with the bunkers. To solve this we built a flag with in the alien struct to tell if the alien was exploding. This enabled us to draw black only if the alien was exploding.