Produced by

Luke Hsiao & Jeff Ravert

Revision 1.1  
16 October 2014

SPACE INVADERS GUIDE

Luke Hsiao & Jeff Ravert

ECEn 427 | Lab 3

2 October 2014

Table of Contents

[Chapter 1: Space Invaders Overview 2](#_Toc401042046)

[Section 1.1: Space Invaders History 2](#_Toc401042047)

[Section 1.2: Game Play 2](#_Toc401042048)

[Section 1.3: Game Details and Specifications 4](#_Toc401042049)

[Chapter 2: Game Console and Engine 7](#_Toc401042050)

[Section 2.1: Game Console 7](#_Toc401042051)

[Subsection 2.1.a: Digilent ATLYS Board 7](#_Toc401042052)

[Subsection 2.1.b: Xilinx Spartan-6 and Microblaze 7](#_Toc401042053)

[Subsection 2.1.c: System Organization 7](#_Toc401042054)

[Section 2.2: Game Engine 7](#_Toc401042055)

[Subsection 2.2.a: Game Engine (Main Game Loop) 7](#_Toc401042056)

[Subsection 2.2.b: Meeting the Game Specifications 7](#_Toc401042057)

[Section 2.3: Application Programming Interfaces 7](#_Toc401042058)

[Appendix A: Bug Report 8](#_Toc401042059)

[References 9](#_Toc401042060)

# Chapter 1: Space Invaders Overview

## Section 1.1: Space Invaders History

Space Invaders, synonymous with classic arcade game culture, was first released in 1978. Its pixelated aliens and chromatic sound loops have become pop-culture icons referencing a simpler age of gaming. Its gameplay have been cloned, mimicked, and remade hundreds of times. It was one of the earliest shooting games and a forerunner of modern video gaming. Its impact helped push video gaming into a global industry.

Space Invaders was originally launched in Japan by Taito, a Japanese publisher of video game software and arcade hardware now owned by Square Enix[[1]](#endnote-1). Tomohiro Nishikado, the game’s designer, drew ideas from popular science fiction media from the time like *The War of the Worlds* and *Star Wars* and from popular level-based games like *Breakout*. Unlike the creation of modern games, Nishikado created Space Invaders singlehandedly over the course of a year. He not only designed and programmed the game, but also created artwork, sounds, and arcade hardware to support it.

At its inception, Space Invaders was run on a custom-made arcade board that featured an Intel 8080 CPU, raster graphics on a CRT monitor, and monaural sound created by analog circuitry and a Texas Instruments SN76477 sound chip. Nishikado found that despite his best efforts, the CPU lacked the power to display more colors or move enemies faster. He also discovered that the CPU was able to render graphics faster the fewer aliens were on the screen. Today, the challenge of progressively faster enemies is a gameplay mechanic used prolifically in the industry.

By the end of 1978, Taito had installed over 100,000 machines and grossed over $600 million in Japan[[2]](#endnote-2). In two more years, Taito had sold over 300,000 arcade machines in Japan with an additional 60,000 in the United States[[3]](#endnote-3). By 1982, Space Invaders grossed $2 billion in quarters[[4]](#endnote-4). Overall, Space Invaders has been a significantly influential piece of hardware and software, and this manual will explain yet another rendition on a modern FPGA.

## Section 1.2: Game Play

Space Invaders is a two-dimensional shooting game. The player can maneuver a tank, positioned at the bottom of the screen, to the left or to the right, and fire lasers at the descending block of aliens. The objective is to eliminate the five rows of eleven aliens before they reach the bottom of the screen. The aliens simply move horizontally left and right on the screen, and drop vertically each time they hit the edge of the screen. As more aliens are killed, the entire block moves more quickly. If the entire block is destroyed, a new block will appear. Players earn points each time an alien or spaceship is killed.

Aliens attack the player by firing bullets downwards while they approach the bottom of the screen. The player can either dodge the bullets, or seek cover behind one of the four stationary, destructible bunkers. If the aliens reach the bottom, the game ends. Bonus points can be acquired by destroying the spaceship which occasionally appears at the top of the screen. Score is kept track at the top left of the screen. The number of lives the player has is tracked at the top right of the screen.

For a summary of each of the game’s units and components:

|  |  |
| --- | --- |
| **Unit Icon** | **Unit Description** |
|  | [40 points]  There are only 11 of these aliens in the back row of the alien block. These are the alien elite which have earned the privilege of being away from the front lines. |
|  | [20 points]  There are 22 of these aliens. These are the middle class and form the middle two rows of the alien block. |
|  | [10 points]  These are the alien grunts. 22 of them form the front two lines. This alien, along with the other two are animated in the game by toggling between two appearances. |
|  | [50-350 points]  The spaceship. Occasionally, the space invaders’ spaceship itself will move through the top of the screen. It is worth significant bonus points if the tank can get a precisely aimed shot past the aliens and take it out! |
|  | There are four stationary, destructible bunkers that the tank can use as a shield from alien bullets. However, any time the tank’s laser or the alien bullets hit the bunker it will begin to be destroyed, so watch out! |
|  | The tank. The player is a tank is equipped with a top-of-the-line laser cannon that has unlimited ammo. This unit is earth’s last line of defense from the space invaders and has to make sure to wipe all those aliens out! The game starts with three additional lives, but it only takes one hit by an alien bullet to destroy the tank. |

Table 1: Space Invaders Unit Summary

## Section 1.3: Game Details and Specifications

Below are the details of the game that must be implemented for the game to play correctly. This guide discusses the tank, bunkers, aliens, and bullets individually. Note that this game was created to be run on a 640x480 resolution monitor, and all of the pixel calculations for all rendering and movement are based on this size.

**Tank**

The tank is rendered from a 32x16 bit bitmap, stored as an array of unsigned integers. For convenience, the software simply tracks and draws the tank based on an (x,y) coordinate storing the location of the tank’s top-left corner. The y-value was made constant so that the tank cannot move vertically. Rather, when the user presses one of the tank’s movement keys, the x-coordinate is updated and the tank is re-rendered. This means that the range of the tank’s x-coordinate values ranges from 0-608.

In regards to unrendering, the software simply blacks out the entire 32x16 box that contains the tank before rendering the tank in its new location.

In order to fire the tank’s laser, the tank bullet’s position is calculated based on the tank’s current location and the corresponding global variables are set. Bullet mechanics are discussed more below.

When a tank is hit, a death animation is shown by toggling between two bitmaps of an exploding tank. Then, a life is subtracted. If the tank is hit and has no extra lives remaining, the game ends.

**Bunkers**

The four stationary, destructible bunkers are made up of 12 sub-blocks (as shown in Figure 1), arranged in 3 rows of 4 blocks. The bunkers are numbered 0-11 going from left-to-right, top-to-bottom. Bunkers are made up of bitmaps for the 4 corner cases (at the top left and right as well as the middle left and right) along with a generic solid block for the rest. Each block can take 4 bullet hits before disappearing.

Similarly, there are bitmaps that contain the 4 erosion patters for the blocks. All the blocks will erode in the same pattern. These erosion bitmaps are ANDed with the bitmap that is currently being drawn. This effectively turns off the pixels that need to be turned off, without altering the pixels that were already off. Each bunker’s location is tracked using global variables of (x,y) coordinates of their top-left corners.

Figure 1: Bunker Blocks

**0 1 2 3**

**4 5 6 7**

**8 9 10 11**

Each block’s erosion state is managed using a single unsigned integer. Because each block has five states—(1) full health, (2) hit once, (3) hit twice, (4) hit thrice, (5) destroyed—it only requires 3 bits to track each block’s status. The organization of each block’s status is shown in Figure 2 below. Bits 31 and 30 are unused.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  | Block 11 | | | Block 8 | | | Block 7 | | | Block 6 | | | Block 5 | | | Block 4 | | | Block 3 | | | Block 2 | | | Block 1 | | | Block 0 | | |

Figure 2: Bunker Block State Layout

Note that because blocks 9 and 10 are always “off” and will only display black, space didn’t need to be allocated to track their status. This design choice allows the data to be compact as possible.

**Aliens**

There are 55 aliens (5 rows of 11 aliens) in the alien block. Each alien has two *guises* or forms so that the game can simulate movement by toggling between the two guises. All of the alien guises are shown in Figure 3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Alien 1** | | **Alien 2** | | **Alien 3** | | **Death** |
|  |  |  |  |  |  |  |

Figure 3: Alien Guises

In order to track, render, and unrender the aliens, the software keeps several global variables. First, it maintains the (x,y) coordinate of the top-left position of the entire alien block. Next, it saves a 16x5 bit array to track whether each alien is dead or alive. If an alien is killed, the death bitmap is shown and then the space is blanked.

The same numbering convention as the bunker blocks is followed, the top-left alien is 0, and incrementing going left-to-right, top-to-bottom, ending on alien 54 at the bottom-right. These two global variables are all that is needed to draw each alien at the correct location.

In order to allow the aliens to continue to bounce off the edges of the screen when entire outer columns are destroyed, there are two variables that serve as imaginary “walls”. The alien block will automatically move horizontally until one of these abstract walls is hit, and then it will move down, and reverse horizontal direction. The wall variables are updated whenever an entire outer column of aliens is killed. This allows for the (x,y) coordinate of the top-left corner to actually wrap around past zero when moving to the left, but the wall variables compensate for the wrap around.

Similarly, the position of the lowest level of live aliens is tracked so that when the aliens get far enough down on the screen, the game will end.

Calculations about whether an alien was hit or not are made using the tank bullet’s (x,y) coordinate, the array tracking alien’s life/death status, and the (x,y) coordinate of the alien block itself.

**Spaceship**

Occasionally throughout the game a red spaceship will appear at the top of the screen. The purpose of this spaceship is for the player to get bonus points. Like the other objects, the spaceship is tracked using the (x,y) coordinate of its top-left corner. The points that this spaceship is worth is a random multiple of ten between 50 and 350. Unlike the aliens that bounce on the edges of the screen, the spaceship will fly off of the edge, giving the player only a small window of opportunity to score the extra points.

When killed, the alien spaceship score will be displayed where the spaceship was when it was killed.

**Bullets**

The aliens can fire two types of bullets. In order to show them as animated, each bullet also has guises as shown in Figure 4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Alien Bullet 1** | | | **Alien Bullet 2** | | | |
|  |  |  |  |  |  |  |

Figure 4: Alien Bullet Guises

Like the aliens, each bullet rotates through each of its guises as it moves so that it appears animated. All of the alien bullets update and move at the same rate, while the tank laser moves at a much quicker rate. In order for the bullets to be drawn correctly, they are drawn last—this way they appear on top of everything else. Aliens are also drawn on top of everything. However, because a bullet and a tank laser will never overlap, rendering a tank laser on top of an alien is not an issue.

The aliens can have up to four bullets in flight at a given time. Only the lowest alien on a column can fire a bullet. So, a random column is generated, and the lowest alien in the column is selected to fire the bullet.

In order to coordinate all of these variables, the software stores an alienBullet struct. This struct contains the bullet’s (x,y) coordinate, its current guise, and its type. In order to make sure that only four bullets can by fired at one time, once the bullet hits something or flies off of the screen, its y-position is set to a number larger than 480, indicating that it is now available to be shot. If a bullet’s y-location is less than 480, it indicates that it is still being drawn on the screen, and thus, still in flight.

The tank laser is rendered in a similar way. However, since the laser is just a simple line, no bitmap is used for rendering. The only variables that need to be associated with the tank laser is its (x,y) location.

**Score**



Figure 5: Score Display

Score is displayed on the top left of the screen as shown in Figure 5. The number is updated each time an alien or spaceship is killed. Aliens and the spaceship are worth the amount of points shown in Table 1.

**Lives**



Figure 6: Lives Display

Lives are shown on the top right hand side of the screen by using the same tank bitmap as shown in Figure 6 above. When a tank is hit, its death animation will play, and one of the life icons will disappear. When there are no remaining extra lives, the game is over.

# Chapter 2: Game Console and Engine

## Section 2.1: Game Console

### Subsection 2.1.a: Digilent ATLYS Board

This version of Space Invaders was designed and developed for use on the ATLYS™ Spartan-6 FPGA Development Board. This board was developed by Digilent and houses a Xilinx Spartan-6 FPGA. Its features are shown in Table TODO below:

|  |  |
| --- | --- |
| **IC:** | Xilinx Spartan-6 LX45 FPGA in a 324-pin BGA package |
| **Connectors:** | One Vmod™ (high-speed VHDC) connector  One 12-pin Pmod™ connector  One RJ-45 connector for 10/100/1000 Ethernet PHY and RS-232 serial  Two HDMI video input ports & two HDMI output ports  Two on-board USB2 ports for programming & data transfer  AC-97 audio with line-in, line-out, mic, & headphone |
| **Programming:** | JTAG programming interfaces compatible with Xilinx's iMPACT™ and Digilent Adept™ |
| **Memory:** | 128Mbyte DDR2 16-bit wide data  16Mbyte x4 SPI Flash for configuration & data storage |
| **CMOS Oscillator:** | 100 MHz |
| **Ethernet:** | 10/100/1000 Ethernet PHY |
| **Power:** | Real time power monitors on all power rails  Ships with a 20W power supply and USB cable |
| **I/O:** | 48 I/O’s routed to expansion connectors  GPIO includes 8 LEDs, 6 buttons, & 8 slide switches |

Table 2: ATLYS Spartan-6 Development Board Features

Currently, this board cost $230.00 for academic institutions to purchase.

### Subsection 2.1.b: Xilinx Spartan-6 and MicroBlaze

The Xilinx Spartan-6 LX45 FPGA included in our development board contains integrated hard memory. It is built using a 45nm, low-power copper process. The LX series are optimized for applications that require the lowest costs. They support up to 147K logic cell density, 4.8Mb memory, integrated memory controllers, DSP slices, and high-performance Hard IP. [TODO more?]

In order to run our game, we’ve opted to use a soft-core processor. A soft-core processor is a processor that is implemented in FPGA logic, rather than cast directly in silicon. Specifically, we use Xilinx’s proprietary MicroBlaze IP. The MicroBlaze contains over 70 user-configurable options. It can operate in a 3-stage pipeline to optimize size, or a 5-stage pipeline to optimize speed. [TODO we are using…]. It also has advanced architecture options like AXI or PLB interface, Memory Management Unit, instruction and data-side cache, Floating-Point Unit, and more. It is a 32-bit RISC Harvard architecture processor.

### Subsection 2.1.c: System Organization

[TODO better picture?]

Figure 7: Block Diagram of System

## Section 2.2: Game Engine

### Subsection 2.2.a: Game Engine (Main Game Loop)

### Subsection 2.2.b: Meeting the Game Specifications

## Section 2.3: Application Programming Interfaces

# Appendix A: Bug Report

This appendix contains a list of bugs and solutions encountered throughout the lab.

|  |  |  |
| --- | --- | --- |
| **Bug Identifier** | **Symptoms** | **Solution** |
| Move to DDR | After moving to DDR, the screen display is incorrect. | The solution to moving to the DDR correctly was making sure that the FRAME\_BUFFER\_0\_ADDR was changed correctly everywhere in the program. It had not worked the first time because a second macro definition was not properly updated. |
| Slow Alien Render | The Alien block appears “flashy” as it draws. | Initially, the software had functions that calculated the color for each pixel of each alien and returned it to be drawn. To significantly reduce the number of function calls and calculations made, the software was altered simply returned a pointer to the bitmap, and used array indexing to draw the entire alien. |
| Missing Collisions | Both the spaceship and the aliens would not register collisions if their top-left position was off of the left-edge of the screen. | This was caused because we use unsigned ints to track (x,y) coordinates. This meant that when an x value was less than zero, it would overflow and prematurely jump out of our logic. To fix this, we added constants that would compensate for the overflow on the bounds. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# References

1. Square Enix History (2003-). (n.d.). Retrieved October 1, 2014. [↑](#endnote-ref-1)
2. Can Asteroids Conquer Space Invaders? (1981, Winter). Electronic Games, 30-33. [↑](#endnote-ref-2)
3. Jiji Gaho Sha, inc. (2003), Asia Pacific perspectives, Japan 1, University of Virginia, p. 57, retrieved October 1, 2014. [↑](#endnote-ref-3)
4. "Space Invaders vs. Star Wars", Executive (Southam Business Publications) 24, 1982: 9, retrieved October 1, 2014. [↑](#endnote-ref-4)