Keith Thompson

Mike Forsyth

Chapter 2: Game Console and Engine

Section 2.1: Game Console

**Subsection 2.1.a: Digilent ATLYS Board**

The Atlys circuit board is a complete, ready-to-use digital circuit development platform based on a Xilinx Spartan 6 LX45 FPGA. The on-board collection of high-end peripherals, including Gbit Ethernet, HDMI Video, 128Mbyte DDR2 memory array, audio and USB ports make the Atlys board an ideal host for complete digital systems built around embedded processors like Xilinx’s MicroBlaze. Atlys is fully compatible with all Xilinx CAD tools, including ChipScope, EDK, and the free WebPack, so designs can be completed with no extra costs.

**Subsection 2.1.b: Xilinx Spartan-6 and Microblaze**

Spartan®-6 FPGA delivers an optimal balance of low risk, low cost, and low power for cost-sensitive applications, now with 42% less power consumption and 12% increased performance over previous generation devices. Part of Xilinx’s All Programmable low-end portfolio, Spartan-6 FPGAs offer advanced power management technology, up to 150K logic cells, integrated PCI Express® blocks, advanced memory support, 250MHz DSP slices, and 3.2Gbps low-power transceivers.

The MicroBlaze™ embedded processor soft core is a reduced instruction set computer (RISC)

optimized for implementation in Xilinx® Field Programmable Gate Arrays (FPGAs). Figure 1-1

shows a functional block diagram of the MicroBlaze core.

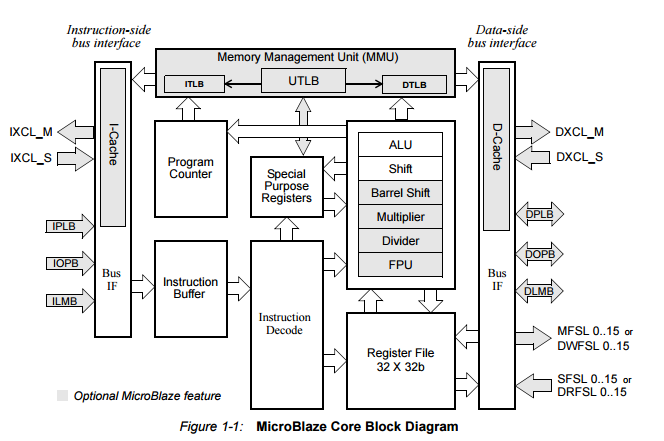
The fixed feature set of the processor includes:

• Thirty-two 32-bit general purpose registers

• 32-bit instruction word with three operands and two addressing modes

• 32-bit address bus

• Single issue pipeline



**Subsection 2.1.c: System Organization**

The Space Invaders hardware setup was given by Dr. Hutchings, but uses the fixed interval timer, push-button interrupts, and the frame pointer to print the objects to the 640x480 monitor. The game is able to be played using only 3 buttons. The left button moves the tank left, center button shoots, and the right button moves the tank right.

Section 2.2: Game Engine

**Subsection 2.2.a: Game Engine (main game loop)**

The main loop primarily consists of initializations including the interrupts, the frame pointer, and a few global variables. After initializations have finished, the main function loops indefinitely allowing the FIT and push-button interrupts to take control of the game. Every loop iteration causes the frame pointer to update the screen.

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

In order to avoid any flickering we reduced overhead by drawing as little as possible. This was accomplished by leading and trailing most moving objects with black space that will automatically clear the areas around them as they moved.

Most timing requirements (as in the speed that bullets, aliens, tank, etc. move) were met by timing the flash game with a watch and then using the FIT. Knowing that every 100 ticks equals a second made it easy to get our game close to the original.

We never completely redrew large objects such as the entire display or the alien block. Leftover artifacts were tricky to get rid of. If an alien that was above another alien is killed, the lower alien no longer gets cleared by the moving black-space. We solved this by **HERE** .

**Section 2.3: Application Programming Interfaces**

We did not use any programming interfaces. All of our code was accomplished using if, switch, for, and while statements. In hindsight we see that we should have used a state machine for the FIT. It would have made the code more readable and efficient.

**Timing & Memory Report**

**Bug Report**

We had several difficult bugs. At one point we accessed outside of the bounds of an array causing a very peculiar and difficult to find bug. Only by using the debugger and stepping through slowly were we able to find it.

Initially we did not think through how we would allow the alien block to move further to the left after the left most column was killed. This nearly doubled our lab time as we had to essentially rework the entire lab to get it to work properly. Nearly everything was using the original alien block location as a reference point and got messed up when we realized we would have to change that reference point to look at the next column.

Our bullets kept overwriting the bunkers during erosion. The major problem was that our bullets moved 6 pixels (quite far) with trailing black space so our bunker sections needed 6 consecutive vertical pixels to prevent the black space from overwriting the bunker. Had we had more time we would have made the bullets move less far and then it would not have been an issue.