All Invaders Are Belong To Us

CSEE 4840 Spring 2011

Project Design

Sanat Apte*, Mashooq Muhaimen†, Rahul Parikh* , Yibin Sun*
*Department of Electrical Engineering
†Department of Computer Science
School of Engineering and Applied Science, Columbia University in the City of New York
{ sa2832, mm3858, rtp2119, ys2522}@columbia.edu

Introduction

The objective of this project is to build an embedded system that mimics the classic arcade game *Space Invaders*. This project will utilize both hardware and software capabilities of the Altera DE2 board. The implementation will involve a combination of C and VHDL. We plan to use a PS2 keyboard as our game controller. We will be using 3 primary keys (left/right for movement and space for fire).

Game Description

Space Invaders is a 2D game in which a player attempts to shoot rows of space invaders as they approach the home planet. The player shoots at the invaders using a spaceship that can only move horizontally. The player spaceship can hide behind bunkers to avoid enemy fire. If the player survives one wave, he is faced with another wave. The waves get progressively more difficult, where difficulty is defined by speed of movement. In our rendition, we plan to limit the number of waves to a small number in the range of 5 to 10. We also plan to implement a point system where the player is awarded points for each invader s/he kills.

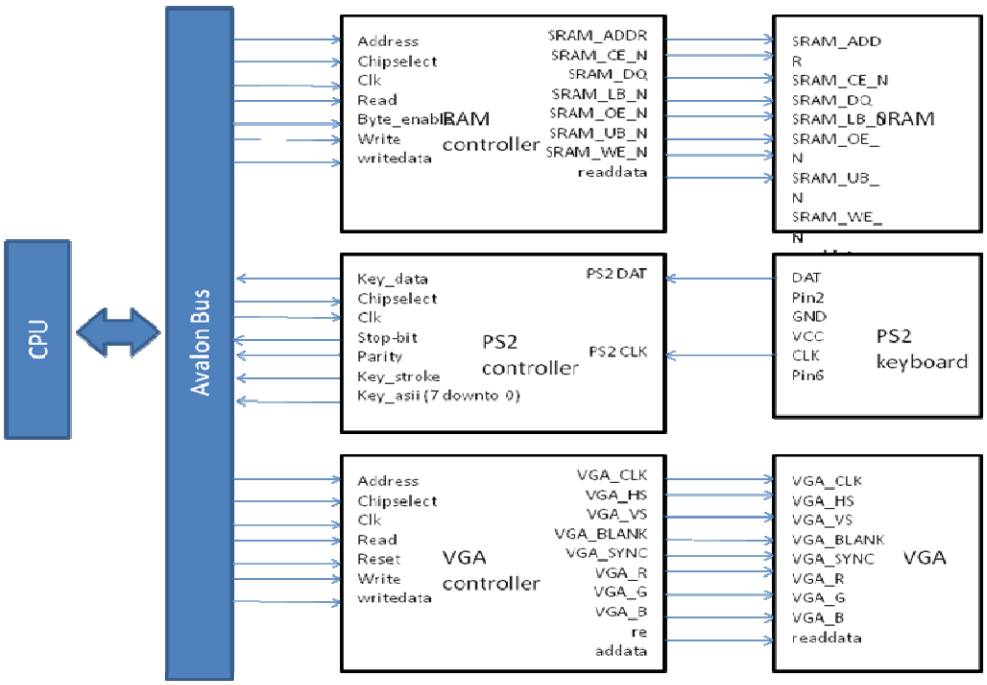

Figure 1. High Level Design

What the software will do:

Software will keep a state of the game. At each time tick (time tick for the SW will be implemented by use of a counter delay), software will update the state and send a state vector to the hardware. An interrupt handler will be setup for the keyboard inputs. When a keyboard interrupt is received, software will make the appropriate update to the game state. However, SW will delay sending the state vector to the hardware until the next time tick. Note that the SW will make continuous update to the state even if no keyboard input is received. This is because the state of the game changes with time.

There will be 5 rows of alien ships, and each row will have 5 ships.

Description	Current Game Score	X Coord of homeship	X Coord of top left alien ship	Y Coord of top left alien ship	Bitmap of dead/alive status for 25 alien ships	Number of active alien missiles	X Coordinate of enemy missile in each column	Y Coordinate of enemy missile in each column
# of bits	32	16	16	16	32	16	80 (5x16)	80(5 x 16)

Table 1. 288 bit state vector that is sent from the hardware to the hardware.

Table 1 shows the 288 bit state vector that the software sends to the hardware. Our design tries to minimize the information software has to send to the hardware. The state vector

consists of the current score, horizontal position of the home spaceship, a starting coordinate of the top-left alien spaceship, and a bitmap for the wave of spaceships indicating alive/killed status of each of the ships. It is sufficient to pass only the starting coordinates of the top-left alien ship as all other ships in the invading wave will have a position that can be calculated from the top-left position (the spacing of the ships relative to each other is always the same). There will be an additional array passed by the software that will indicate the position of each of the enemy missiles. Note that this array will have the same size as the number of spaceships in an invading row as the missiles do not move horizontally.
We decided to make the barricades indestructible, so no information about the barricades has to pass from the SW to the HW. The SW will take the stationary barricades into account in its collision logic.

What the hardware will do:

The character maps will be hardcoded in the hardware code. The character maps stored in the hardware will be: the home ship, an alien ship (we use the same image to draw all the alien ships), an image of a missile and the digits 0-9 for displaying score. Since each of the characters will be at most 32x32, all the images together should occupy less than 13K. Given that the on-chip RAM has 60K, we believe all the characters will fit into the on-chip RAM so we don't need to use the SRAM (other than for CPU usage). As the VGA raster scans the screen, based on the state information sent by the software, it will decide whether it is supposed to draw a pixel from a sprite image at location (x,y) or leave it blank.
Upon reception of the SW state vectors, hardware will update its own version of the state information. We will implement a state machine to ensure hardware receives all the information before the VGA raster starts using the updates.
The hardware will also contain a PS2 keyboard controller that passes commands from the user to the software.

Hardware/software interface:

The SW will communicate to the VGA module through the Avalon bus. The PS2 controller passes information to the software by use of an interrupt set up at initialization by the software.

Milestones:

Milestone 1: March 29
- Get the PS2 controller to work i.e. make sure the software receives the user key inputs properly.
- Get the software to talk to the hardware and have the hardware update its own version of state information. Test with simple display.

Milestone 2: April 12
- Display home ship, barricades and score using state information passed by the software.

Milestone 3: April 28
- Get hardware to display the alien wave and missiles.
- Start developing game logic in SW.

Final Deadline: May 13
- Finish game logic implementation.
- Finish the project. Prepare for summer.