Phoenix Game on DE1-SOC Board CSEE E4840 – Embedded Systems – Spring 2020

Brianna Williams, Ignacio Ramirez, Vaishnavi Murthy
bjw2135@columbia.edu, ir2331@columbia.edu, vm2591@columbia.edu

Introduction

Phoenix is a space-themed "slide and shoot" game originally developed by Taito and Amstar Electronics in the early 1980s. It is known for having multiple levels and enemy variants at different points in the gameplay in addition to being one of the first games to have a video game boss at the final stage. Another unique feature of the game is also that it was one of the first to have continuously playing music during the initial levels, when in-game music was not common.

Fig. 1 Phoenix game level 1 graphics display

In our project, we will implement one level of this game, which consists of the player destroying a formation of alien birds (as shown in Fig. 1) that move in a patterned, but somewhat unpredictable manner. Some of the birds in the formation will also shoot downward "kamikaze style" in an attempt to crash into the player's spaceship.

Architecture

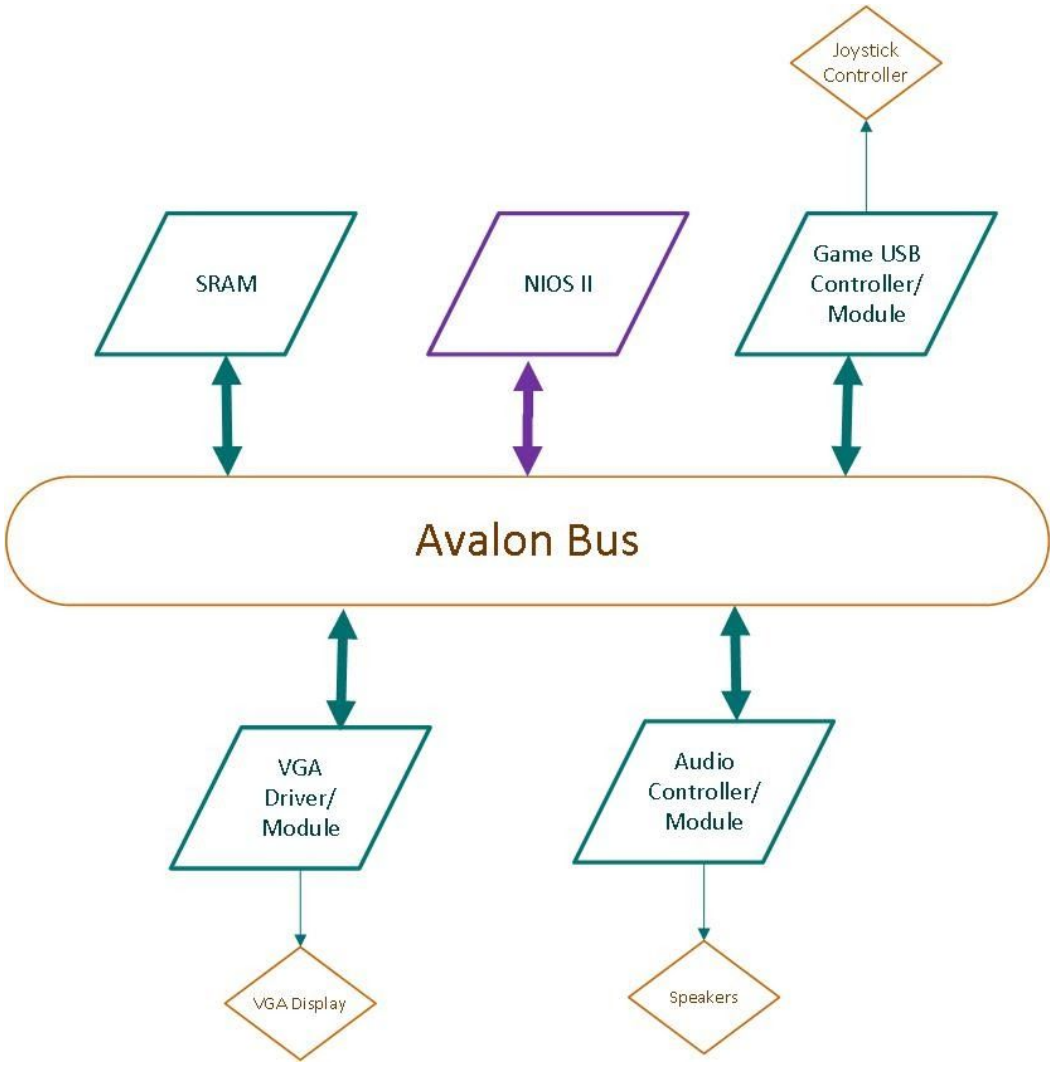

Fig. 1 Architecture Block Diagram

In the above diagram, the Avalon bus functions to enable communication between it and the slave peripherals such as the SRAM, VGA controller, audio controller, and the USB joystick controller. The code for these various peripherals will be written and compiled in VHDL with the SOPC builder in quartus. The Nios II is the processor that acts as the CPU and connects to the FPGA board peripherals. Software modules will allow for generation of the appropriate graphics and audio components that are essential for proper gameplay.

Hardware Implementation

The hardware devices to be used in this project include the DE1-SoC FPGA, VGA Display, joystick input, and the speakers. Fig. 2 shows how the peripheral hardware devices should communicate with the FPGA through the Avalon bus. Because we need to communicate with multiple devices using one bus, we can implement the I2C protocol. The I2C interface consists of two lines- serial clock (SCL) and serial data (SDA). Additionally,

each device has a unique address that can be used by the master to specify individual slave devices to talk to at certain times.

The VGA driver receives the x and y position for a sprite and adds it to a queue for that particular frame. It continues to receive the position information until an end bit is placed in the queue. From here, the frame is created by assigning each pixel from the sprite queue, which tells us if any of the sprites are positioned at that pixel. If there is one, the controller gets the sprite image needed and displays that appropriate pixel. Once the entire frame is completed, the sprite queue is cleared.

Software Implementation

A. Spaceship Control

The user spaceship will be controlled by the joystick controller. The joystick moves left and right, which will correspond to the spaceship's horizontal movement. Software will be used to keep track of the lives of the spaceship and the position of the spaceship.

B. Shooting functionality

The user spaceship fires a laser to hurt the enemies. This laser is controlled by the press of the joystick button. Software will be used to translate the press of the button to the laser shot that moves in a straight vertical line. If the laser intercepts something in its path, the laser disappears and whatever that laser hit is "hurt" [Requires a sound]. Once the laser is shot, its path does not change.

C. Enemies

Enemies float around the board trying to hurt the user spaceship. Software will be used to assign new locations of the enemies as the enemies move. The enemies have the shooting functionality and fly down to "crash" [touch] the spaceship so that the enemies hurt the spaceship. Each enemy also has a tracker for how many shots are needed to "kill" that enemy, when to shoot its laser, when to fly down to attack the spaceship, and whether it should be displayed on the board or not.

D. Scoring

Software will keep the score of the user based on how many enemies defeated and the type of enemies defeated. The score will be displayed and labeled in the upper left corner of the screen. The highest score recorded will be kept and labeled in the upper middle of the screen.

Memory

Audio

The DE1-SoC is connected via I2C interface to the Wolfson WM8731 audio CODEC (Encoder/Decoder).
Estimated memory needed: 30sec * 44kHz (samples/second) * 2bytes/sampling = 2.64MB
1GB SRAM on board is enough to hold the approximately 30 seconds of background music and sound effects needed for this videogame:

20 seconds of background music - repeat until the game is over
1 second of enemy destroyed

0.5 second of shot fired
0.5 second of enemy hit
0.5 second of player hit
3 seconds music for the end of level

VGA display

Memory needed for this game (Each pixel is 3 bytes - 8 bits R, 8 bits G, 8 bits B)

Sprite/Tile	Amount	Approx. Pixel size	Approx. Total ROM (bytes)
Player	1	32*32	3072
Enemies	~8	32*32	24576
Bullets	~10	8*8	1920
Background	1	640*480	921600
Score	3	32*64	6144

Total ROM = 957312 = ~957KB

The game display will be divided into several layers. The sprites of higher priority will be displayed above those of lower priorities. The Sprite Controller Module will calculate the final RBG information by merging the different sprites, and send that information to the VGA module.

Milestones

Milestone 1: Finish writing preliminary version of software/ hardware modules. Recognize peripheral devices. Hardware Setup.

Milestone 2: Static display of background, spaceship and enemies. Have a start screen. Audio output not dependent on game action.

Milestone 3: Moving bullets, players, and enemies. Final implementation of game logic:

- Shooting destroys and corresponding graphics for enemy/player dies and disappears. Shielding logic.
- Improved graphics throughout the game.
- Keeping track of score and highest score between rounds.
- Audio output varies with game action.

References:

[1] https://en.wikipedia.org/wiki/Phoenix_(video_game)
[2] https://www.youtube.com/watch?v=vT4-0eJfsto