Star Wars

Design Report

CSEE 4840 Embedded System
Spring 2014
Prof. Stephen A. Edwards

Fang Fang (ff2317)
Jiaxuan Shang (JS4361)
Xiao Xiao (xx2180)
Zhenyu Zhu (zz2281)

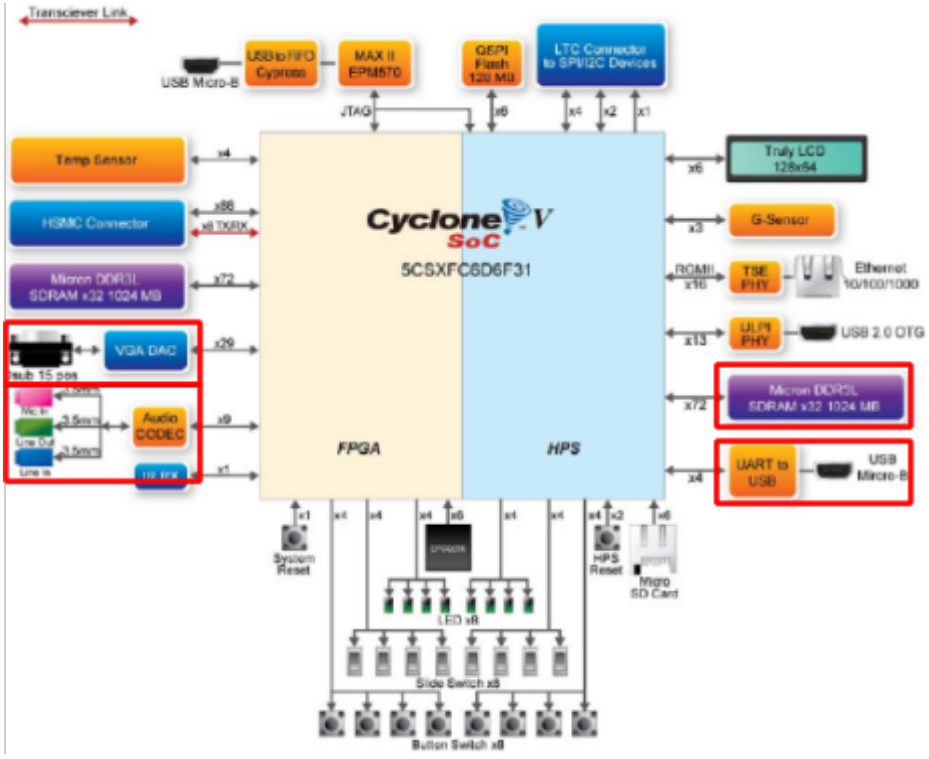
Project Introduction

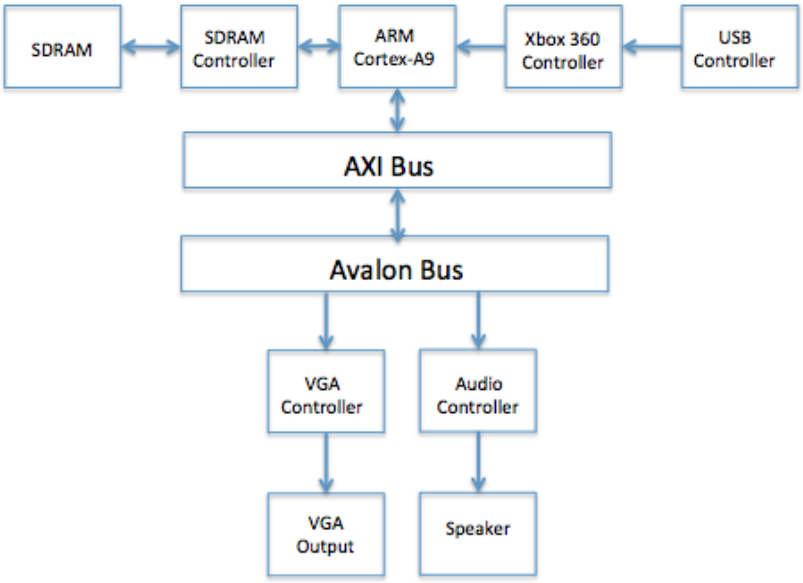
The project is inspired by the classic Xbox 360 game *Geometry Wars*. The object of the Star Wars is to survive as long as possible and score as high as possible by destroying an ever increasing swarm of enemies. The player is able to control a spaceship with an Xbox360 Controller.

Some of the asteroids in the original *Geometry Wars* game are turned in to invading spaceships and they are also able to shoot bullets. The bullets coming from the enemies have two kinds of colors, black and white. The spaceship that the player controls will be able to change its color to match the color of the incoming bullets to avoid damage and increase energy. If the energy gauge is full, the spaceship is able to use up all the energy to throw out a bomb to clear all the asteroids and invading spaceships on the screen. If the energy gauge of the spaceship is more than half full, the player could use half of the energy to turn the spaceship into burst mode, in which the spaceship will be able to shoot 3 bullets at a time, rather than 1 bullet at a time in normal mode. There will be bullet-packages appearing on the screen having different functionalities. The player will be able to add extra energy, points, or lives to the spaceship.

High-level Block Diagram

There are many components on the SoCKit, including the LCD, flash, Audio DACs, and IR. The block diagram of the SoCKit is shown below: The components we plan to utilize are also marked:





The Avalon Bus communicates between the ARM processor and different peripherals. The Xbox 360 Controller will be connected to the ARM processor through HPS. We will be using the SDRAM on the HPS for memory to store sprite figures and the sound waveforms. VGA and audio output are connected to the FPGA portion of the board through the Avalon bus. The HPS could talk to FPGA through an AXI bus.

Hardware Design

Game Logic Module:

The game logic module is the major portion of the design. It consists several sub-modules such as a random number generator module, a collision module and an entity movement module.

The random number generator module will be generating random numbers for the starting positions of the spaceship and the enemies.

The collision module detects if there are collisions happening. Three of the possible collisions are *bullet-enemy collision*, *bullet-spaceship collision*, and *spaceship-enemy collision*.

The movement module analyzes the data containing the position of the spaceship on the Avalon bus and returns with the updated positions of the enemies chasing after the spaceship.

VGA Display Module:

This module produces a sprite graphics signal that goes directly out to the VGA interface. The level structure of sprite from bottom to top in order is listed as below:

Background space picture, Gaming score and energy bar information, Spaceships and enemies, Bullets and attacking effect.

There are several kinds of elements to be shown: enemies, avatar, bullet/bomb, background and the score/life/power record. All those elements can be stored as fixed figures in the SRAM on the

board as it save the resource on the FPGA. To implement the attacking/defending action display dynamically, we plan to display all the elements by refreshing the sprite frequently. The sprite is generated by graphics module illustrated above.

We will assign 8 registers from 0x0 to 0x8 for the VGA peripheral, with 16 bits in each register. X, Y coordinates will be stored in the first four registers for position of the avatar and the direction of the bullets. There is one register for saving data of the color of the spaceship and one for sending out the bomb. And also one register for Pause/Resume and one for the game to get started.

Input (controller) module:
An Xbox 360 controller will be connected to HPS on the board through a USB cable. The output of input module includes: spaceship coordinate (x_pos, y_pos), bullet coordinate (x_blt,y_blt), note the coordinate is transmitted as a 16-bit value for each axis, plus 16-bit process control (pause/continue/), 16-bit for start-over, 16-bit bomb_button to fire the bomb, and 16-bit color_change.

Audio Controller:
The SoCKit board provides 24-bit audio via the Analog Devices SSM2603 audio CODEC. The SSM2603 is controlled via a serial I2C bus interface. Every clock cycle, audio controller will read the audio data stored in the SDRAM and play it using the audio output device. In this project, we need to produce the general background music and sound effects such as firing, collision, and ending-up of the game.
Audio effects will be added to the game at the end of the project.

Software Design

The module of game control could be done in software. The control includes the position of the spaceship, issuing the bullet, firing the bomb, color changing of the spaceship, the game process like start-over, pause, and continue. The module takes the outputs of input module and decides the next position, whether or not to change the color of the spaceship and other functions according to the control of the joystick. Then it sends the generated coordinate (x, y) of the ship and the bullet to the game logic module to generate the coordinates of enemies.

SW-to-HW Communication Protocol

There are 8 16-bit registers used for the software-to-hardware connections. 0x0 and 0x1 store the coordinate (x, y) of spaceship. 0x2 and 0x3 store the coordinate (x_b, y_b) of bullets. In order to control the game process, 0x4 and 0x5 are responsible for recording START and PAUSE signals. 0x6 stores if the color of the spaceship changes. 0x7 stores if a bomb is used to destroy all the enemies on the screen.

Memory

Our design needs memory to record the positions and states of all entities as well as the sprite figures. The SoCKit board has 1GB (2*256MB*16) DDR3 SDRAM on HPS. We could have the processor use several block RAMs to store each entity. The entity information is designed to store in a standardized way: each has public and private information.

Public information is shared with the game module to get updates. This information is stored in the form of 32 bits and consists of “id”, “x”, “y” and “angle_tag”. The “id” and “angle_tag” together determine the sprite pattern. These two parts are used by the game module to determine the object’s behavior and by the graphics module to determine its graphical sprite representation. X and y are integers used to represent the center of the object.

Private information is also 32 bits used to record each entity’s private information including the following parts: X, Y and state. X and Y are fixed point extensions of the integers x and y and are used to guarantee smooth movement. The state bits’ function often varies by entity. For instance, some entities use these bits as a timer, while others use them to store its velocity.

The sprite figures that represent the spaceship and different kinds of enemies are stored in the other part of SRAM as a lookup table. The table is indexed by the mentioned “id” and “angle_tag” in public information.

Reference:

1. SoCKit User Guide
   http://www.rocketboards.org/pub/Documentation/ArrowSoCKitEvaluationBoard/SoCKit_User_manual.pdf
2. SoCKit Getting Started Guide
   http://www.rocketboards.org/pub/Documentation/ArrowSoCKitEvaluationBoard/SoCKit_QSG.pdf
3. Vertex. Final design report. Don Goldin Mark Sullivan
   http://web.mit.edu/6.111/www/f2008/projects/aureus_Project_Final_Report.pdf