Phillip Mercer
Richard Agbeyibor

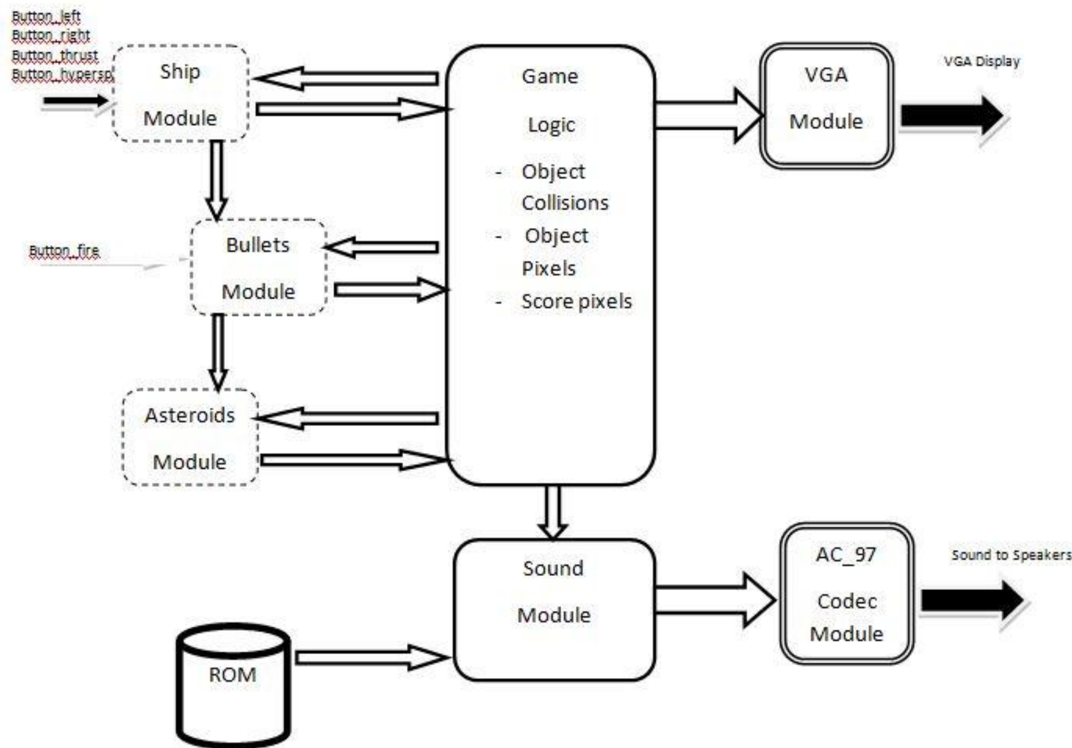
6.111 Project Proposal (Revision)

Implementation of the Atari game ASTEROIDS on a Xilinx II FPGA

Introduction

Our project is to recreate the classic video game Asteroids, which was originally released to the Atari 2600, on the Xilinx II FPGA. In the game Asteroids, the player navigates a spaceship in outer space and shoots at incoming asteroids and other space debris in order to protect the ship. The ship has thrusters and a gun, which allow it to rotate, accelerate, and shoot incoming asteroids.

The game design is divided into the physical design of the objects: the ship, the bullets, the asteroids, and their physical behavior on screen. There are multiple instances of these objects on screen and they all interact with each other. A sound module provides a musical score to the game events.



Asteroids - Block Diagram w/ External Signals

Ship Module

(Agbeyibor)

The ship module controls the position, velocity and heading of the battle ship in the game. The battleship is of triangular shape and rotates in space. In the original Asteroid game, the ship is able to rotate continuously and shoot in any direction.

The ship module takes as input the status of buttons for right and left rotate, as well as a thrust button, to control the triangular ship on screen.

The Ship module will output the ship's x and y coordinates, the ship's heading, and the position of the ship's leading vertex. The leading vertex is the edge of the triangle where the gun is mounted and from which bullets emanate.

Asteroid takes place in space, and the ship can fire both engines to accelerate in the direction of its current heading. We plan to model the ship as a point in space and use simple 8.01 equations to calculate its acceleration, velocity and position in an x, y coordinate system. There is a coefficient of drag which causes the ship to come to a stop after some time. Our unit of time will be delta_t, an experimentally determined period at which we will poll the status of the buttons.

The basic formulas of motion of the space ship are as follows:

```
acc <=  (thrust)?  (thrust_coeff - drag_coeff) : (vx>0 or vy>0) ? (- drag_coeff) : 0
vx <= vx + acc*arccos(heading)*delta_t
vy <= vy + acc*arcsin(heading)*delta_t
x <= x + vx* delta_t
y <= y + vy* delta_t
```

If the user turns on the thrusters, the ship will accelerate with a magnitude that we determine by subtracting the drag coefficient from a thrust coefficient. If there is no user input to thrust, we want the ship to decrease its velocity and stop after some time. Therefore the acceleration will be negative and equivalent in magnitude to the drag coefficient. Otherwise, there is no acceleration.

The thrust, drag coefficient values, and delta_t will be determined experimentally to make gameplay as slow or fast as we deem reasonable.

Calculating trigonometric operations is challenging because of the continuous nature of the values involved. To avoid the difficulties of continuous math, headings of the ship will be quantized angle values with a precision of 10 degrees. We will store a trigonometry look up table in memory, to automate the calculation of cosine and sine values. When a user presses the left and right rotate buttons, we will add to or subtract from the heading using modular arithmetic, so as to wrap around at 360 degrees.

Despite this trigonometric optimization, the calculations of the movement equations will take several clock cycles. To prevent any data timing mishaps, we plan to insert delays and buffers between stages to allow the completion of these calculations.

A feature of the original Asteroids game is hyperspace tele-transportation. In Hyperspace mode, the ship disappears from its current location and travels at high speed to another plane in

space. On screen the ship would disappear briefly and reappear in another location, with a new configuration of asteroids in its neighborhood. This feature is entertaining and adds to the quality of gameplay, however it is difficult to implement and poses a risk to the completion of our primary milestones. We will leave the implementation of the hyperspace feature as a bonus time-permitting project.

Bullets module

(Agbeyibor)

This module controls the bullets shot by the battle ship. In the original game Asteroids, the ship can fire a continuous stream of bullets if it is not actively rotating or accelerating.

This module takes in as input: the button fire which is user input, and the positions of the ship's leading vertex or gun turret and the ship's heading which are outputs of the ship module.

The bullets module will output bullets_positions and bullets_visibility which determine for each bullet where it is and whether it is visible/active or not.

The challenge we face with our implementation in Verilog is to appear to create and destroy objects on the fly and store variables for each of those objects.

Our solution is to keep create a determined number of available bullets ahead of time. There is a maximum of 10 bullets in the gun turret, and every time we poll the user inputs and get the order to fire, we will shoot off the next available one. If all bullets are currently on screen, no new bullet can be shot. Bullets are returned to the gun turret if they impact an asteroid, reach the boundaries of the screen, or are a certain distance away from where they started.

Each bullet will keep track of the position of the ship's leading vertex and heading, at the time it was fired, and figure out what direction to travel on for a pre-determined value bullet_range.

Asteroid Module

(Mercer)

This module will control the size, speed, and direction of an individual asteroid. We intend on implementing three different sizes of asteroids, each with its own unique range of speeds. Each asteroid, when shot, will split into smaller asteroids, unless it is the smallest size in which case it will be destroyed.

The module will require an input signal from the Game Logic module to determine if the asteroid has been hit as well as the horizontal and vertical pixel counts of the VGA. The output will be the pixel coloring of the asteroid if it has not been destroyed and if the pixel counts correspond to a pixel on the asteroid. This output will be used by the Game Logic module to determine object interactions before being displayed on screen.

As a method of creating randomness, the module will maintain an internal variable that is squared every time that the asteroid is recreated. Certain ranges of bits will be selected from this variable to determine the different parameters of the asteroid, such as the initial x and y positions

as well as the x and y velocities. The updates to the positions will be a simple addition of the velocity to the corresponding position after the completed drawing of a frame.

This module doesn't have many mathematical complexities. The main complexity is that it will require a multiplier in order to square the randomness variable. Coupling multiple asteroids together to form the different sizes of the asteroid will prove to be difficult so we will focus on creating a functional module with only one size of asteroid and then implement the asteroid coupling.

Sound Module

(Mercer)

The sound module will be in charge of playing sounds at the appropriate times during gameplay. While this module will drastically increase the entertainment value of our game, it is lower on the priority list behind the fundamental game logic. The number of sounds that we will be able to incorporate will be dependent on the amount of time we will have at this point of the project.

This module will require two inputs: a signal from the game logic to play sound and a signal selecting which sound to play, should we incorporate multiple sounds. Once the module receives a play input and selects the correct sound data, it will send the sound to the onboard AC97 for playback.

The sounds to be played will be calculated using MATLAB and then placed into ROM storage for access. The size of the ROM will be proportional to the number of sounds that we are able to implement.

Game Logic Module

(Mercer)

The game logic module will keep track of any and all object interactions before sending out information to be displayed. The interactions of importance are Asteroid/Bullet and Asteroid/Ship. Should an Asteroid hit a Bullet the Asteroid will splinter or be destroyed and the Bullet will be destroyed. If an Asteroid hits the Ship then the Ship is destroyed and the game is over. If there are no interactions at the moment then the game just proceeds.

The module requires the pixels of the Ship, all Bullets, as well as all Asteroids. This module will output a signal to each of the Asteroid, Bullet, and Ship modules to inform the module if it has been destroyed. The other output will be the pixel information to be displayed on the screen.

The determination of contact will be hardware intensive as you have to consider every Asteroid/Bullet pairing as well as every Asteroid/Ship pairing. In order to determine if a pair has collided we can use an AND gate with one bit of pixel information from the two objects to inform you if they both occupy the same pixel space. In order to minimize the hardware that is being used simultaneously we can first determine which pairings need to be considered.

VGA Module

(Agbeyibor)

The VGA module receives pixel information for each frame from the Game Logic Module. The VGA module implements the FPGA to VGA link, according to appropriate timing and configuration parameters for display to a 1024x728 resolution screen.

Test Plan

(Agbeyibor)

Many of our object modules depend on each other, which makes independent testing challenging.

The first module to be implemented is the VGA display module, which is an adaptation of code written by course staff for Lab 3. This module will be used by the author of each game object to test the proper rendering and movement of their object.

Once the individual object modules are complete, the Game Logic module will be written. The Game Logic module has many functions. It superposes all the different sprites of the game onto the visible frame, draws these frames to screen pixel by pixel , and keeps track of the score and other game statistics.

Project Timeline

Dec 05: Demo
Nov 28: Multi-player features
Nov 21: Sound Module and Hyperspace Module
Nov 14: Game Logic
Nov 7: Object Modules: Ship, Bullets, Asteroids