Phillip Mercer
Richard Agbeyibor

6.111 Project Proposal

3 November 2011

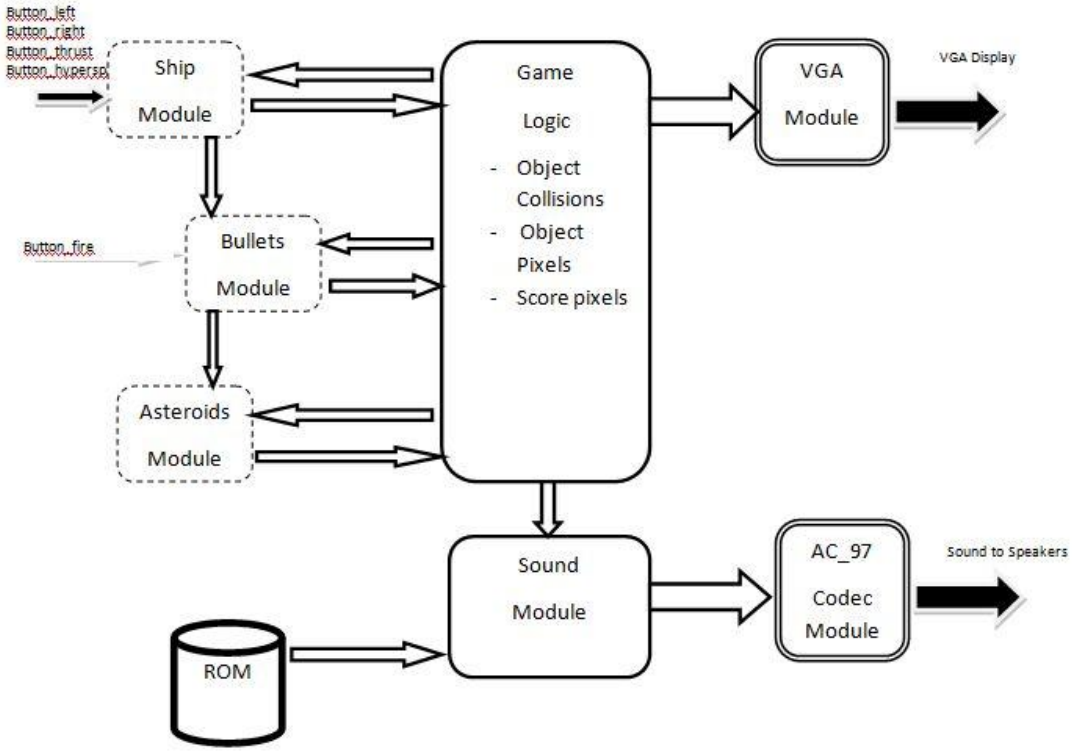
ASTEROIDS

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, that allow it to rotate and accelerate, and shoot incoming asteroids.

Our 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. Our game has a sound module which provides a musical score to the events happening on-screen.

We begin with a discussion of the individual object modules and the sound module. The game logic module regulates the interaction of all the objects on screen, and determines what pixels should be drawn to screen at each frame, and send this data to the VGA module which outputs it to the screen.. We conclude the project proposal with a testing plan and implementation timeline.



Asteroids - Block Diagram w/ External Signals

Ship Module

This module will control the position and heading of the battle ship in our game. The battleship is of triangular shape and rotates in space. The module takes as inputs the status of buttons for right and left rotate. In the original Asteroid game, the ship is able to rotate continuously and shoot in any direction. We wish to replicate this feature, and by having finely quantized angular rotation.

In addition to left and right rotate buttons, the ship takes as input a thrust button. 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 come 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 easy or fast as we deem reasonable.

Our headings will be quantized angle values with a precision of 10 degrees. We will have a trigonometry look up table in our code, to mimic the calculation of cosine and sines. When the user hits 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 optimizations, the calculations of these formulas will take several clock cycles. To prevent any data timing mishaps, we plan to insert enough delays and buffers between stages to allow the completion of these calculations.

A feature of the original Asteroids game is hyperspace tele-transportation. The ship would disappear from its current location and travel at high speed to another plane in space. On screen the ship would disappear briefly and reappear in another spot, with a new configuration of asteroids in its neighborhood. We will leave the implementation of this feature as a bonus time-permitting project.

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.

Bullets module

This module will control the bullets shot by the battle ship. In the original game Asteroids, the ship could fire a continuous stream of bullets if it was not actively rotating or accelerating. 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.

This module takes in as input, the button fire, position of the ship's leading vertex or gun turret an the ship's heading.

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.

Asteroid Module

This module will control the size, speed, and direction of an asteroid. In the original game of asteroids there are three different sizes of asteroids, each type with its own speed of motion. When shot, the asteroids split into smaller pieces, unless they are the smallest size in which case they are destroyed. We will also implement the three different sizes of asteroids and expect the splintering to be a challenge. We intend to solve this problem by coupling together multiple smaller asteroids to create a larger one which, on contact from a bullet, will lose coupling and split off into smaller pieces. Another issue that we will be facing with the asteroids in this game is limiting the number of asteroids on the screen. Our solution, thus far, is to limit the number of Asteroid modules that are created within the labkit code.

This module only requires one input: a signal from the Game Logic module to determine if the asteroid has been hit. Internally the module will keep track of horizontal and vertical directions. The speed of the asteroid is predetermined based on the size of the asteroid, though the velocity will be determined from the proportions of the horizontal and vertical directions. One way to implement this is to utilize a lookup table with sine and cosine functions to determine the appropriate horizontal and vertical velocities. In order to create randomness in the asteroids movements the module will also maintain a number from which we will select certain bits for each direction before squaring it in preparation for the next asteroid. The randomness in determining original position and asteroid size will be handled similarly.

The output from this module will be the pixel coordinates of the asteroid, if it is not destroyed. This output will be sent to the Game Logic module to determine object interactions before being displayed.

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. In order to minimize multipliers we can utilize the one number and utilize different bit ranges for the location, directions, and asteroid size, thus minimizing hardware requirements. The directions will be updated by adding a coordinate velocity to the current coordinate position. However, as stated earlier we will need to utilize a look up table in order to simplify finding the exact coordinate velocities.

Sound Module

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 have remaining 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. The sound waves to be played will be synthesized with the aid of MATLAB and then placed into ROM storage for access.

Once the module receives a play input and selects the correct sound data, it will send it to the onboard AC97 for playback.

Game Logic Module

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 then 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.

Due to all of these interactions, there are a lot of inputs into this module. The module requires the pixels of the Ship, all Bullets, as well as all Asteroids. The determination of contact should be an AND function so this step should not be complicated. This module will also keep track of the in game score which will be updated upon Asteroid/Bullet contact.

There will also be a large amount of outputs from this module. There will be an output signal to each of the Asteroid, Bullet, and Ship modules in order to inform the module as to the whether or not that object has been destroyed. The other output will be the final pixel information, containing the locations of the Asteroids, Bullets, and Ship as well as the player score, will be sent out to be displayed.

VGA Module

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

Many of our object modules depend on each other, which makes independent testing somewhat challenging. The first module to be implemented is the VGA display module, which is just 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 objects are complete, the Game Logic module will be written. There are sub-components to the Game Logic module, such as the superposition of the different sprites, as determined by rules of collisions. In addition to drawing the modules, the Game Logic module keeps track of the score and 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