Asteroids – Project Proposal

Description

The aim of the project is to create on the labkit a version of the popular computer game ‘Asteroids’. Asteroids is a game set in 2 dimensions featuring a spaceship in a field of moving asteroids. The asteroids move randomly and spin. The player controls the spaceship, moving it around the playing field and shooting at the asteroids to destroy them. When hit by the spaceship’s weapon the asteroids break down into smaller asteroids and eventually are removed from the field. The game is completed when all asteroids have been destroyed. The game is lost if the spaceship collides with an asteroid. The game will allow for a maximum of 30 large asteroids on screen at the beginning of the game, which will each split into 2 smaller asteroids upon collision with a bullet. The ship will be able to fire multiple bullets at the asteroids. The game will be implemented in a 1024 by 768 pixel screen in 256 colours and be controlled by 1 player using a joystick.

System Overview

The system will be divided into several smaller modules each of which contributes some specific aspect of functionality. The main system modules will be: gameplay, audio and video, their interconnectivity is shown in Figure 1 (some of the video control lines are omitted for clarity).

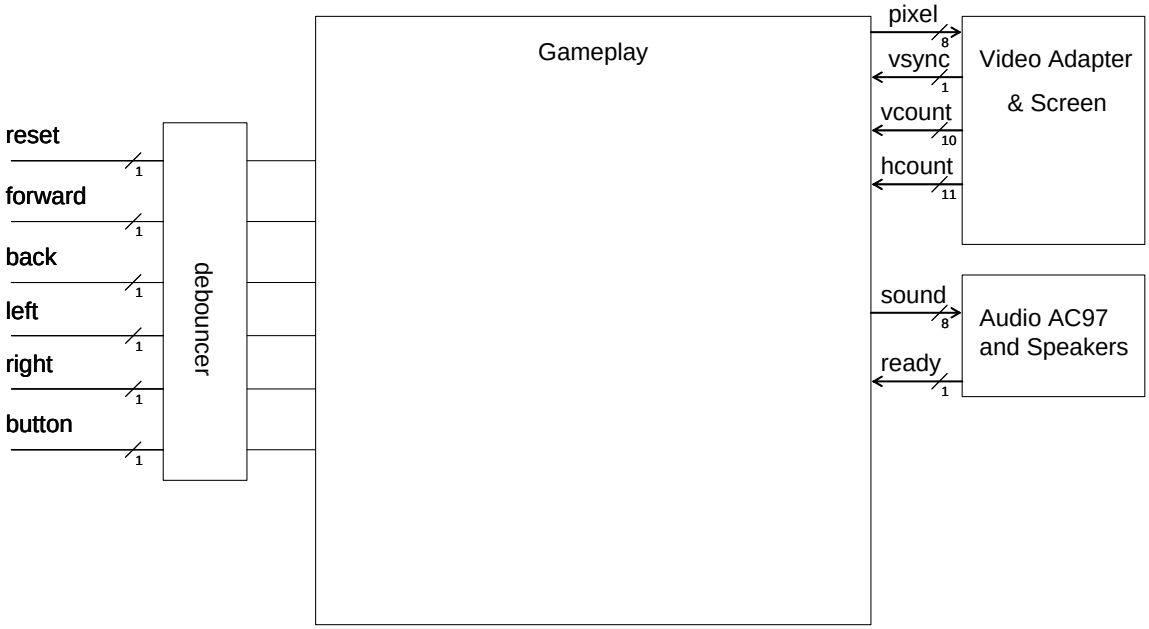

Figure 1 - System Overview Diagram

The inputs to the system are from the joystick (`Forward, Back, Left, Right, Button`). These inputs will be debounced and synchronised before going into the gameplay module. The only video control line of interest to us in designing the gameplay module is the `vsync` line and this will be used to cause an update of screen positions once per frame.

Gameplay Module

The gameplay module does all the hard work, it contains several smaller modules. Each of these module contains the position and speed of an object (e.g the ship, an asteroid, etc). For each frame, each of these modules calculates the new position of the object and when called upon by video hardware, outputs what pixel exists for each screen pixel. The gameplay module itself will provide collision detection and appropriate creation and destruction of the objects. We consider below the function of each of these modules.

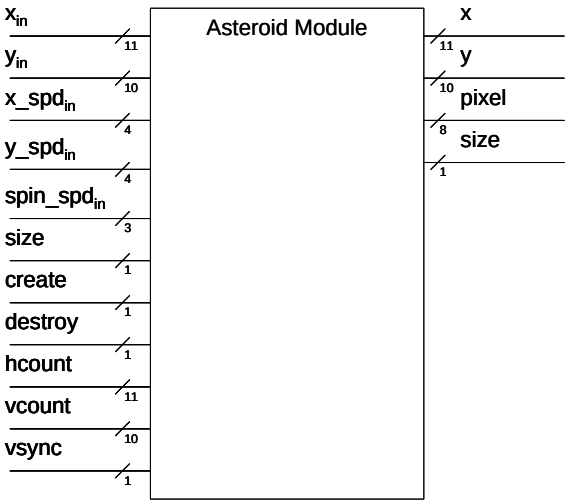

Figure 2 - Asteroid Module Diagram

Asteroid Module

The asteroid module (shown in Figure 2) is initialised using the x_{in}, y_{in}, x_spd_{in}, y_spd_{in}, $spin\_spd_{in}$ and `size` inputs. At the start of the game, and when smaller asteroids are created later in the game, each asteroid module is called in turn and initialised with random values (or, if being created from the destruction of a larger asteroid, values consistent with Newtonian mechanics). When the `create` input is taken high for a clock cycle the asteroid is initialised. The random values will be generated within the gameplay module from a random number table. The asteroid module outputs appropriate pixels for use in the video based upon its current position and the values of `hcount` and `vcount`. When an asteroid is destroyed, `destroy` will be taken high by gameplay and the module will stop outputting pixel values. By detecting changes in `vsync` the asteroid

module will at the end of every frame, calculate its new position value based upon its speed.

Asteroid Module

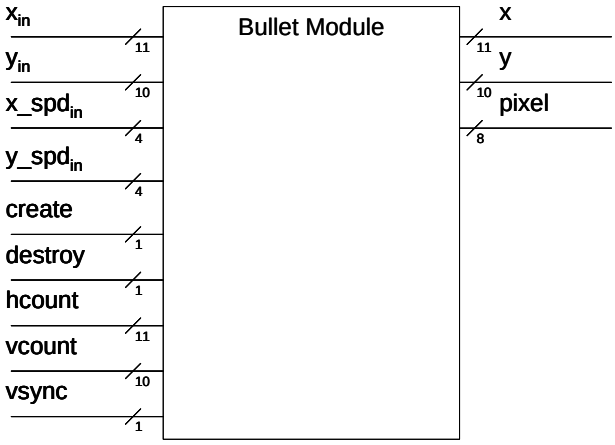

Figure 3 - Bullet Module

The bullet module (shown in Figure 2) is initialised using the x_{in}, y_{in}, x_spd_{in} and y_spd_{in}, inputs.

When a bullet is fired by the played, one of the bullet modules will be initialised. Initial values for the speed and position will be calculated based upon the angle, speed and position of the ship and will show the bullet as having been fired directly forwards from the ship at a particular relative speed.

The bullet module outputs appropriate pixels for use in the video based upon its current position and the values of `hcount` and `vcount`. When a bullet is destroyed, `destroy` will be taken high by gameplay and the module will stop outputting pixel values. By detecting changes in `vsync` the bullet module will at the end of every frame, calculate its new position value based upon its speed.

Ship Module

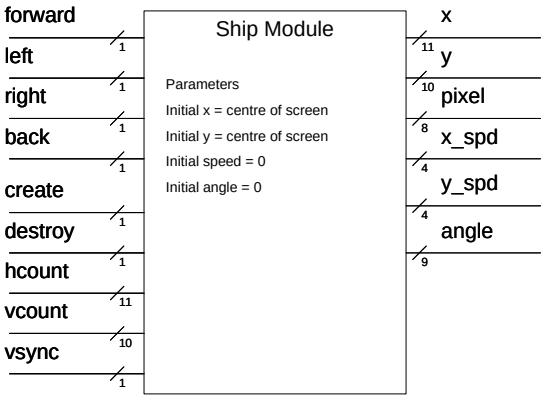

Figure 4 - Ship Module

The ship module depicted in Figure 4 is always initialised to place the ship in the centre of the screen. The ship is created by taking the `create` line high. The bullet module outputs appropriate pixels for use in the video based upon its current position and the values of `hcount` and `vcount`. When the ship is destroyed,

`destroy` will be taken high by gameplay and the module will stop outputting pixel values.

By detecting changes in `vsync` the ship module will at the end of every frame, calculate its new position, speed and angle values based upon its current position, speed and angle and the inputs from the joystick.

The control system will function as follows:

- Forwards and Back will affect the speed in the forward/backwards direction. Holding the joystick forwards or backwards will cause the ship to accelerate/decelerate linearly upto some maximum speed.
- Left and Right will affect the ships angle. Holding Left or Right will cause the ship to rotate with constant angular velocity

Visual Module

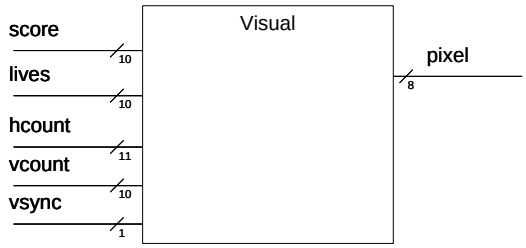

Figure 5 - Visual Module

The visual module shown in Figure 5 creates a border around the screen and a score bar at the bottom of the screen. The video module outputs appropriate pixels for use in the video based upon the values of `hcount` and `vcount`.

Sound fx Module

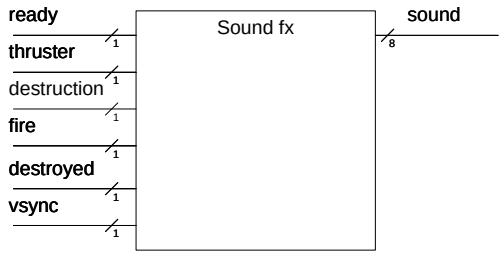

Figure 6 - Sound fx Module

The sound fx module generates 4 possible sound effects. A thruster sound for when the thrusters is being used, a destruction sound for when an asteroid is hit, a fire sound for when a bullet is fired and a destroyed sound for when the ship collides with an asteroid and is itself destroyed. These are activated using the appropriate control line and the output is sent to the audio module.

Video and Audio Modules

The audio and video modules interface with the hardware systems to provide the actual inputs and outputs. These will function using similar methods to the final labs. The audio module will take in sound data and play it out using the

ac_97 hardware. The video module will display pixels in 256 colours based upon its input and provide appropriate control signals.

Integrated System

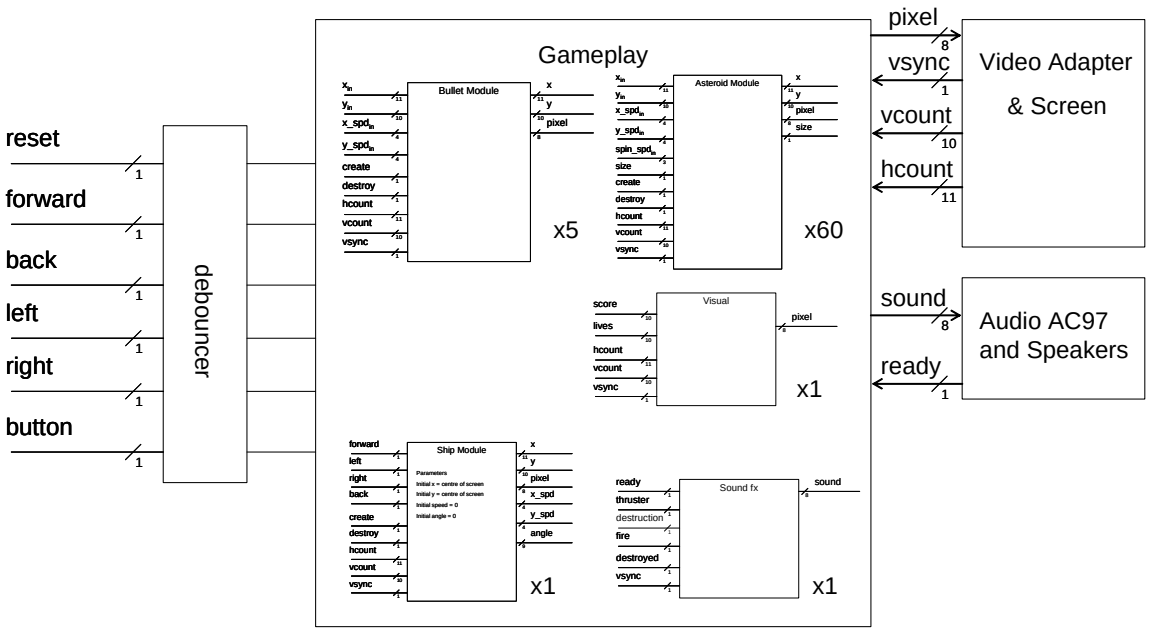

Figure 7 - Integrated System Diagram

Implementing each of these modules the integrated system diagram is shown in Figure 7. An appropriate number of each sub module is implemented in the gameplay module to take into account the overall game specification, e.g. 60 asteroid modules, 5 bullet modules (based upon speed of bullets, rate of firing and size of screen), 1 ship module.

Levels of Implementation

The basic implementation will feature the basic function as described above excluding the sound effects module. Collision detection will initially be basic and implemented by simplifying asteroids, bullets and ships to be rectangles. If we complete these tasks in sufficient time we play to also implement the following:

1. The sound effects system
2. Alien robots
3. Power-ups
4. Improved Collision Detection
5. Choice of spaceship

Division of Workload

From Figure 7 the modules will each be created by one team member.

Shield will create the:

- Gameplay
- Asteroids
- Visual

James will create the:

- Ship
- Bullets
- Video Adapter