High Striker
6.111 Final Project Proposal

Jennifer Chan and Mike Stunes

November 8, 2010

1 Overview

We propose to create a system that recreates a simple carnival game known as "High Striker." In High Striker, the player swings a hammer, striking a target, which sends a weight flying up a pole. There is a bell at the top of the pole; if the weight hits the bell at the top of the pole, then the player wins. We will recreate this by attaching an accelerometer to a hammer, which is used to strike a target. There will be an LED and photodetector near the target, which will be used to determine when the hammer strikes the target. The information from the accelerometer will be converted to an initial velocity, which will be used by a physics engine to calculate the motion of the weight flying up the pole. The motion of the weight will be sent to a graphics engine, which will draw the weight, pole, and bell to the screen. A sound module will also play various sounds, including the bell ringing, providing a fun interactive experience.

See Figure 1 for a system block diagram.

2 Accelerometer to Velocity (Mike)

2.1 Accelerometer

The motion of the hammer is measured by a Dimension Engineering DE-ACCM3D 3-axis accelerometer. Either one or two axes of the accelerometer will actually be used; determining which axes are used will be done by experimentation. The accelerometer will most likely be mounted inside of the hammer; its output will be sent directly to the ADC.

2.2 ADC

The ADC is an Analog Devices AD7824 four-channel ADC. It takes analog output from the accelerometer and outputs the raw digital equivalent to the FPGA, sampled at the maximum rate sustainable by the ADC.

2.3 ADC Interface Logic

This is a module inside of the FPGA that is responsible for interfacing the ADC to other parts of the circuit. It will generate control signals necessary to drive the ADC, read digital data from the ADC, and generate ready signals and pass along the data from the ADC to other modules in the system.

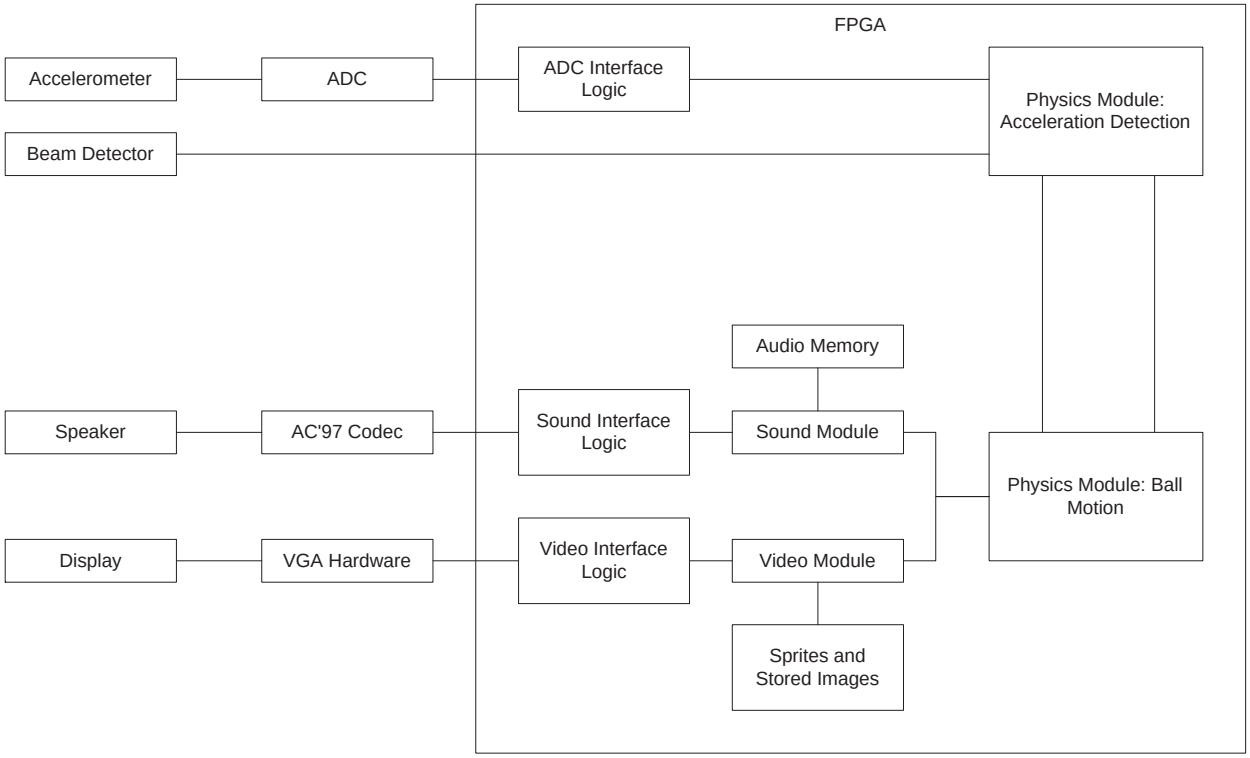

Figure 1: Block diagram of the High-Striker game.

2.4 Force Computation Module

The force computation module takes data from the ADC interface logic and computes the amount of force applied to the weight. The output of this module is the initial velocity of the weight, which is computed from the force. The beginning of the computation is triggered by the LED beam module (below).

This module may also incorporate an adjustable sensitivity, implemented as a scaling factor on the initial velocity.

2.5 Beam Detector

This module detects when the hammer strikes the target, using an LED and a photodetector. Its output is a single logic pulse when the beam is broken, and is used to trigger the start of computation in the force computation module.

3 Computing and Displaying Motion (Jenny)

3.1 Timestep Module

This module will input the master system clock and output a timestep signal, which will be used by the physics module to calculate the motion of the weight as a function of time.

3.2 Physics Module

The physics module will take, as input, the initial velocity and a timestep signal. It will output the position of the ball based on the amount of time elapsed. We can calculate the position of the ball with the equation

$$x = v_0 t + \frac{1}{2} a t^2$$

where v_0 is the initial velocity and a is acceleration due to gravity. t is the timestep, as given by the timestep module. x is the relative position of the weight, assuming that its initial position is $x = 0$ (without loss of generality). The final task of this module is to calculate the absolutely position of the weight in screen coordinate space from the relative position of the weight on the pole.

3.3 Collision Testing Module

If enough force is applied, the ball could collide with the bell at some nonzero velocity. This module will take as input the absolute position of the bell and the current position of the weight. If the current positions of the weight and bell are the same, this indicates that the weight and bell have collided. This module outputs the modified position of the weight, based on whether the bell and weight have collided, as well as a signal that is asserted if the bell and weight collide.

3.4 Graphics Module

The graphics module features several components that will be drawn to the screen. See Figure 2 for an overview.

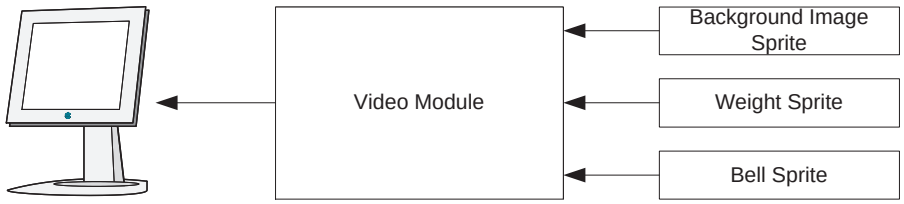

Figure 2: Block diagram of the graphics subsystem.

3.4.1 Background Image

The first component will be the background image. A MATLAB script will convert a bitmap image to a form suitable for storage on the FPGA. Because of possible memory limitations, we will constrain the image to 800x600 pixels, using 4-bit color. If 4-bit color does not accurately represent our image, we will consider using 8-bit color.

3.4.2 Weight

For initial testing purposes, we will use a circular or rectangular sprite as a weight, in a way similar to the `blob` module from Lab 5. Once initial testing is complete and we have verified the correctness of the physics system, we will use a MATLAB script to convert a bitmap image of a weight to an FPGA-suitable format, and implement it as a sprite. The x, y coordinates of the weight will be taken from the collision module.

3.4.3 Bell

The bell will also be drawn using a sprite. As a possible extension to the project, we could cause the bell to appear to vibrate when struck by the weight. This could be done by simply moving the bell a few pixels in several directions.

4 Audio Modules (Mike)

See Figure 3 for a diagram of the audio subsystem.

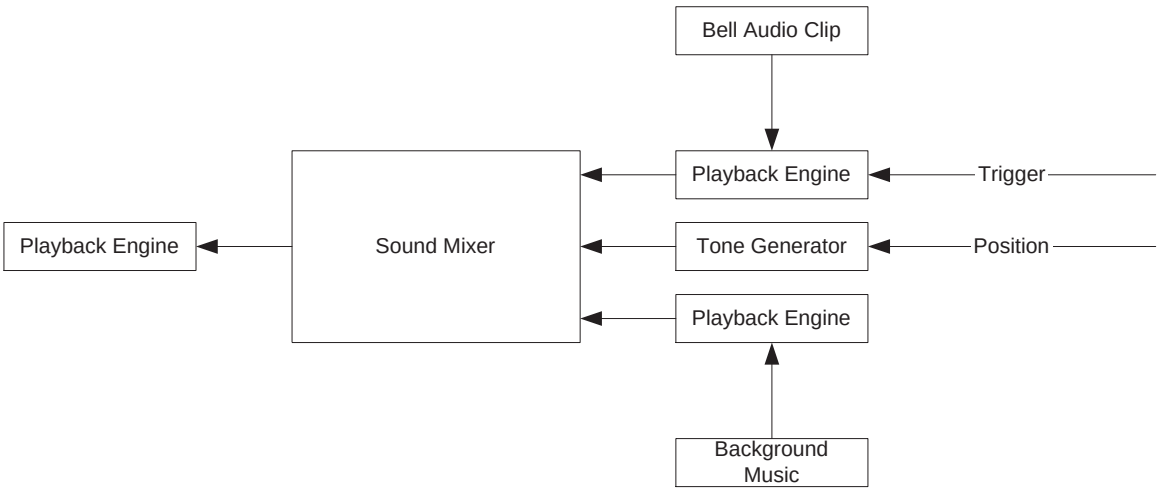

Figure 3: Block diagram of the audio subsystem.

4.1 Bell Ringing

This module will receive the signal asserted when the weight collides with the bell. On a rising edge on this signal, this module will begin playing a stored audio clip of the bell ringing, using the AC'97 codec.

4.2 Possible Extension: Weight Travel Sound

A sound could be played as the weight travels up the pole This could simply be created by generating a tone with a certain frequency; as the weight moves up the pole, the frequency of the tone would increase. This could be implemented using the position of the weight relative to the top and bottom of the pole, and a provided minimum and maximum frequency.

4.3 Possible Extension: Background Music

A short audio track could be played in the background of the game. Because of memory limitations, the sound could be generated by merely creating tones at a series of stored frequencies and lengths. The track would simply repeat continuously.

5 More Possible Extensions

5.1 Keeping Score

One possible extension is to allow for a high-score list, showing a list of players and their scores on the screen. By mapping characters to bitmap images, we could create an additional sprite for the high-score list. When a player finishes the game, he/she could enter his/her initials using the controls on the labkit.