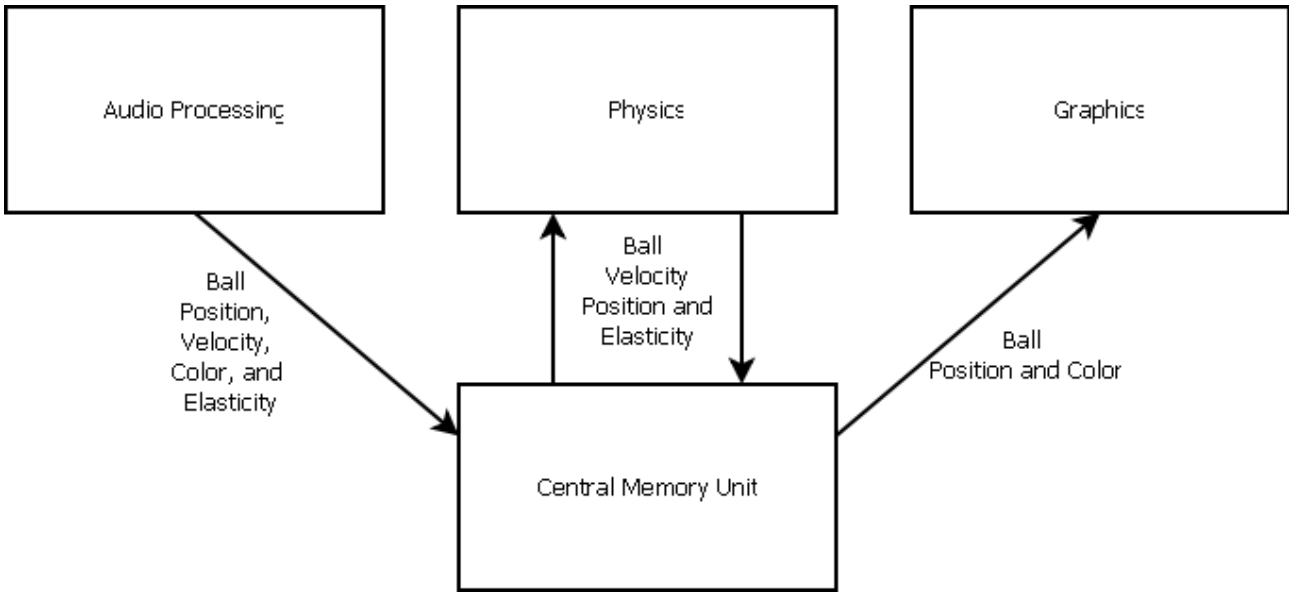
6.111 November 9, 2009
Bellagio Fountain Simulation

1 Introduction
We propose to create a simulation of the Las Vegas Bellagio fountains using an FPGA. Our project is subdivided into four categories. The first module implements real-time audio analysis algorithms to control the behavior of the fountains. The second provides an accurate 3D physics engine to moderate the behavior of the water balls that exit the fountains. The third module implements realistic raster graphics with customizable vantage points. Finally, the fourth module is a memory that keeps track of all ball characteristics.

Each module has been implemented in python to validate important concepts in our design. Final specifications will be determined through testing in the FPGA environment.



2 Audio Signal Processor
This module is responsible for handling incoming audio data and converting it to signals that are useful for controlling the behavior of the fountains. From the audio signal, we will extract amplitude, frequency, beat measurements, and tempo. Each fountain will correspond to a different band in the frequency spectrum. The direction and speed of the balls exiting the fountain will depend on the amplitude the audio signal within the fountain's frequency band. Additionally, the color of the balls will change depending on the tempo of the music. The module nature of this approach allows us to easily add or remove behavior from our design.

2.1 Fast Fourier Transform (FFT)
Input: from_ac97, ready
Output: magnitude

The FFT Module converts the incoming audio signal into the frequency domain. The frequency spectrum will be subdivided into 16 separate bands, each corresponding to a different fountain in our simulation. The amplitude of each frequency band is used to control the velocity of the balls coming out of their respective fountains.

2.2 Beat Detector
Input: from_ac97, ready
Output: beat

The Beat Detector module implements a beat detection algorithm to locate musical beats in the audio signal. The beat detection algorithm compares the instantaneous audio signal energy to the average energy of the audio signal within 1 second of the current sample. A beat is confirmed if the instantaneous energy is above the average energy multiplied by a constant. To accommodate different types of audio, the constant is dependant on the variance of the sample energies.

2.3 Tempo

Input: beat

Output: tempo

The tempo module records the times that a beat is recorded by the Beat Detector and translates the occurrences into an accurate tempo. A running average will be used to estimate the tempo to help reduce error caused by noise within the beat detection signal.

2.4 Ball Clock

Input: clock, time_parameter

Output: ball_enable

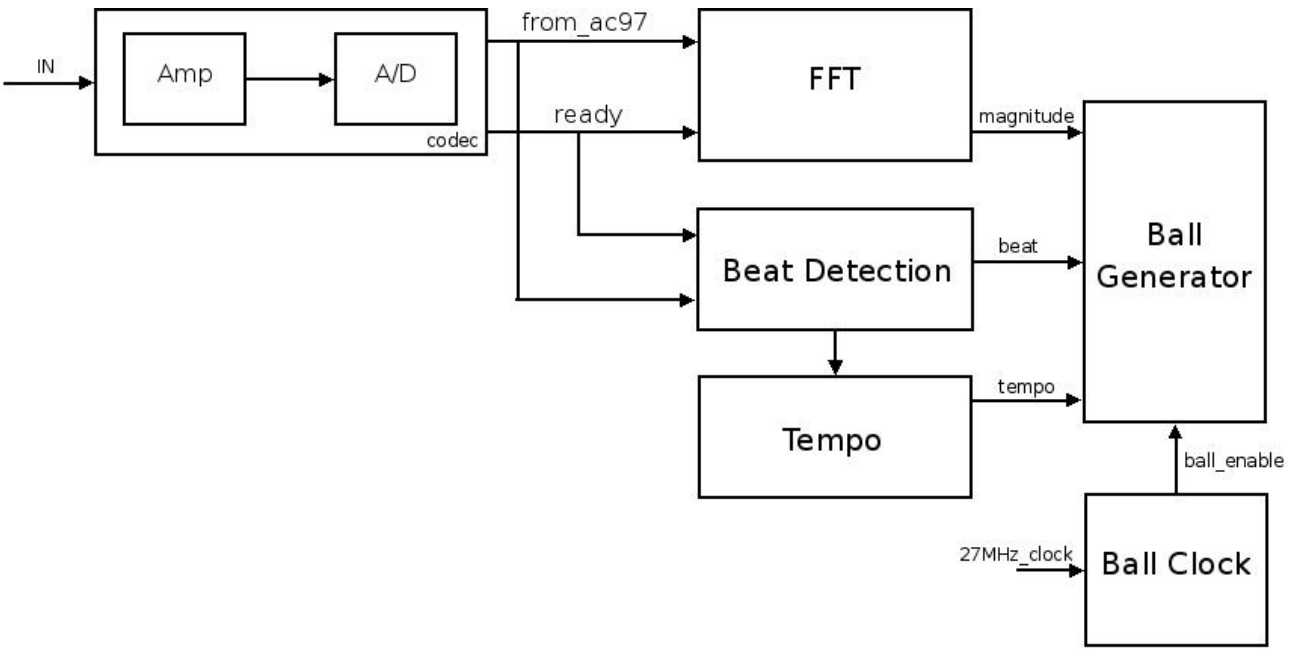
The Ball Clock module converts the default clock signal into an enable signal that instructs the ball generator to generate a new set of balls to be launched from the fountains. The default frequency for the ball_enable signal will be 8Hz. The user will be able to control this parameter through switches on the labkit that correspond to different ball_enable frequencies.

2.4 Ball Generator

Input: spectrum_amplitude, beat, tempo, ball_enable

Output: Writes to ball memory

The Ball Generator takes in the signals computed by the FFT, Beat Detector, and Tempo modules and translates them into fountain behavior and ball characteristics. Each time the ball_enable signal is high, the Ball Generator writes 64 new balls to the top of the ball_memory. The spectrum frequency determines which fountain the ball is launched from. The magnitude of the frequency spectrum samples control launch velocity. The color of the ball is determined by the tempo. Finally, the beat signal is used to mark possible transitions in the fountains. Additional behavior suited to the beat signal will be added if time permits.



3 Graphics

This module takes the data calculated by the first two modules and creates a graphical representation of the scene. We use a modified version of raster graphics to accomplish our goal. Since a ball just looks like a circle from every angle with proper shading we can create the illusion of a 3D scene without actually taking the processing power to render 3D.

3.1 3D Ball Projection

Input: ball locations, camera position and angle, viewer position
Output: ball location on the 2D screen and distance from the viewer to the ball

This is the heart of the graphics engine. It takes in the location of the ball in 3D space and projects it onto the screen. This is done by drawing a vector from the location of the viewer to each ball. If the vector passes through the screen then the ball is in view. This location on the screen is the center of the ball. While making this calculation the distance from the viewer to the ball is also computed.

3.2 Drawing and Shading

Input: ball location on the 2D screen, distance from the viewer to the ball, ball color
Output: pixel data

Once the location of the ball on the screen has been determined it must be drawn into a buffer. The radius of the ball is a function of its distance from the viewer. For each point near the location of the ball on screen it is determined whether $x^2 + y^2 < r^2$. If it is then that pixel contains the ball, but there is still the problem of multiple balls taking up the same pixel spaces. This problem is solved by assigning each pixel a depth. The first time a pixel is written in a frame the distance from the viewer to the ball is saved as the pixel depth. Then the next time that a ball is rendered on the same pixel. The new pixel depth is compared to the old pixel depth. Whichever one is smaller is the pixel that is kept.

This module also provides shading to the balls. We are planning to use Phong shading to add specular highlights to the balls. This should greatly increase the illusion of depth to the scene. The technique is done by drawing a vector from a light source which then reflects off of the ball. The dot product is then taken between this reflected vector and the viewer's vector to determine shading.

3.3 Floor Rendering

Input: coordinates of the corners of the floor
Output: pixel data

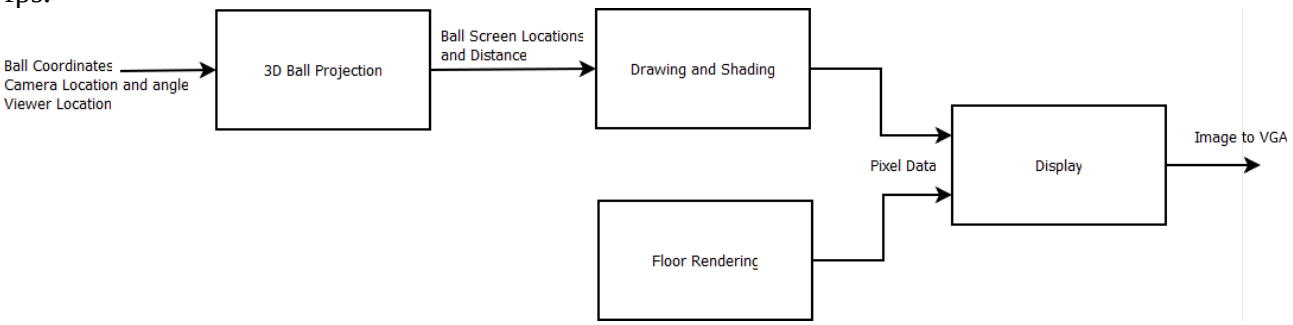
Since the floor is rendered slightly differently than the balls is gets its own module. Each corner is projected onto the screen just like the center of each ball is projected onto the screen. Then to determine which other points on the screen are within the floor area, a vector is drawn straight up at each point. If the vector passes through exactly one side of the base, that pixel is within the base and should be shaded accordingly. If the vector passes through 0 or 2 sides of the base, then the pixel must be outside the base and is not shaded.

3.4 Display

Input: pixel data
Output: image to VGA

The Display module consists of two buffers. The first contains the image that we are

currently displaying on the monitor. The second is the upcoming image that we are currently calculating. Every 1/30th of a second the buffers are swapped giving us a constant frame rate of 30 fps.



4. Physics

This module updates the ball positions at each iteration step based on calculated velocities. Then collisions are detected and further Newtonian calculations will be carried out to determine the post-collision velocities of each ball involved.

4.1 Collision Detection

Input: Ball Data from Memory Unit, Disable Signal from Audio Signal Processor
Output: Ball Data to Memory Unit

All inputs into the memory module will be sorted by space into separated 'bins', this is discussed in the Memory Unit section. When the Disable Signal is not on, Each pair of balls in each of the space bins is passed through this module, to determine whether it's involved in a collision. The result is updated in the corresponding Memory Unit parameter.

4.2 Physics Calculation

Input: Ball Data from Memory Unit
Output: Ball Data to Memory Unit

After all of the entries in the memory unit is updated for by the collision detection module, each ball/pair (depending on whether they're involved in a collision based on their updated collision-parameters), is passed to the corresponding module for physics calculation when the Disable Signal is not on.

4.2.1 Non Collision Physics Module

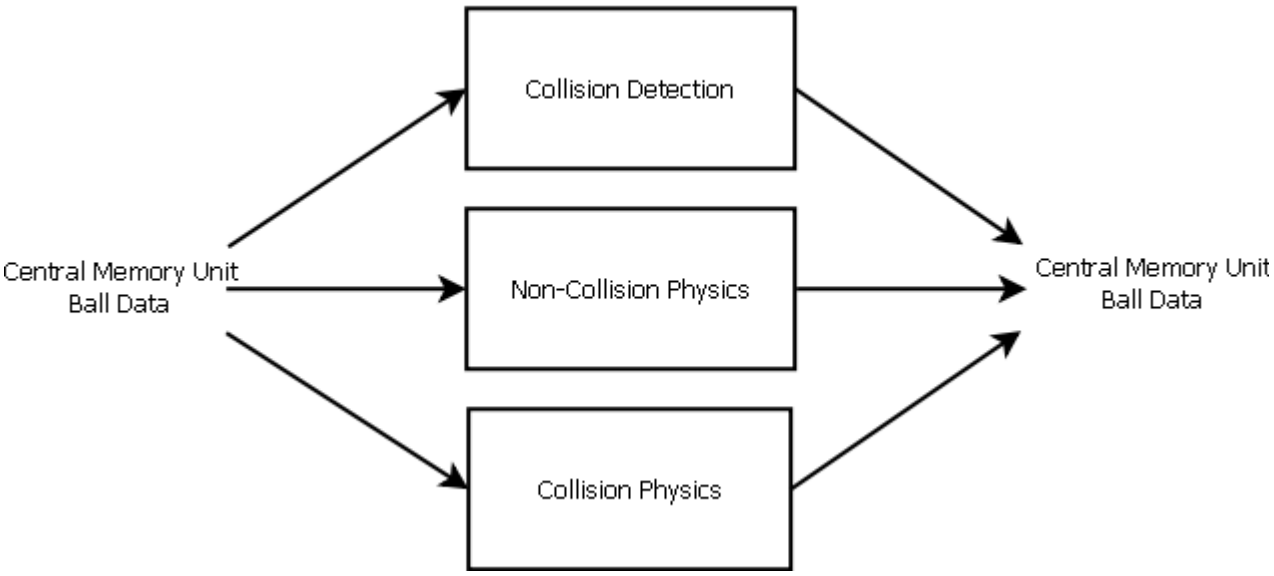
Balls that are not involved in any collision is passed to this module. The calculation involves simple addition and subtraction from each ball's given velocities and gravity. Multiple of this unit will be present for parallelization.

The resulting positions will be passed to the Memory Unit.

4.2.2 Collision Physics Module

Pairs that are involved in collision detections are passed through the Collision Physics Modules. These modules encapsulate multiple multipliers and adders which complete an entire series of collision calculations. Multiple of this unit will be present for parallelization.

The resulting positions will be passed to the Memory Unit.



5 Central Memory Unit (CMU)
Input: Ball Data from Audio Signal Processor and Physics Module
Output: Ball Data to Physics Module and Graphics Module
The CMU contains all of the existing ball data. These data is updated by the Audio Signal Processor and the Physics Module. Further, all of the ball data is sorted by space regions to make collision detection possible a reasonable amount of time

5.1 Sorting Unit

This Unit sorts all of the ball data according to which space region it belongs and puts the ball data into the corresponding 'bin'.

5.2 Memory

The bulk of the CMU is the many registers which hold the ball data. Each register contains the ball's position coordinates, velocity coordinates, elasticity, color, timing flag and collision flag. There is a total of 1,000 (estimated) of these registers corresponding to a maximum of 1,000 balls to be simulated.

The content of these registers are read by the Physics module as well as the graphics module. Both the Audio Signal Processor and Physics Module write to the registers as well. However, since the Audio Signal Processor and the Physics Module should not access the registers at the same time, the Audio Signal Processor has a disable signal that when on will prohibit the Physics Module from reading and writing to the memory.

5.3 Oldest Ball Unit

Since only a finite number of balls can be simulated, new balls generated will replace the oldest balls in simulation. This is done by the Oldest Ball Unit. Each ball has a timing parameter that ranges from 0 to 1,000/8Hz (this is the number of iterations it takes to reach the maximum number of balls allowed). On the 1,000/8Hz + 1 iteration, the balls with '0' in the timing parameter would be replaced. On the next iteration, the balls that are to be replaced will be those with '0' and '1' (depending on the specific numbers), and so on.