Peter Lin (998317548)
David McTavish (998449585)
Peng Xue (998477811)

Billiards

I. Introduction

Video gaming has evolved from simple analog stick controls to modern motion capturing devices like the PlayStation Move and the XBOX Kinect, yet games are still displayed on a vertically oriented display. Conventional table-top arcade games such as air hockey and billiards are still prevalent group activities. But these set-ups are often too financially cumbersome and space intensive to be a common household object.

With this project, we would like to demonstrate the ability for electronic devices to create affordable, space efficient realistic gaming experience that will cater to more than just video-gamer enthusiasts.

Similar billiards simulation games have been made for consoles and PC platforms that try to emulate the real life billiards experience. Like our design, these games have a realistic physics engine for the ball cue and ball. One advantage of our design over previous virtual pool simulations implementations is our horizontal display which is a better representation of the real game. On the other hand, pool games on consoles and PCs are allowed better physics engine and graphical interface.

II. Goal

We want to create a realistic billiards experience that will not be dampened by the need to learn a new controller system. With this pool-cue interface, we hope to create a life-like billiards experience that will feel like the traditional game, but without the financial and spatial commitment.

Peter Lin (998317548)
David McTavish (998449585)
Peng Xue (998477811)

III. Specifications

a) Functions and features required

- software-based physics engine (to calculate collisions)
- pool cue (user interface)
- horizontal display/monitor (table)
- motion capturing camera (for pole movements)
- game logic library (scoring, illegal moves)

b) Peripheral Requirements:

- motion tracking camera (to track the cue's position)
- pool cue with:
    - a button (toggle tracking on/off)
    - an led light (to be detected by the camera)
    - (optional) Gyroscope (to determine the cue's orientation when it strikes the cue ball to implement backspin/forespin)
    - computer screen (laid on it's back as the table)
    - External DDR memory (to store input/output video data)
    - (Optional) Speakers (to output sounds in addition to the video)

c)Constraints and limitations of your design

- the table area and screen size/resolution will likely be fixed for simplicity.
- the processing power available will likely limit the number of balls in play and the rendering rate
- User interface only available with the cue ball
- physics limited to 2-D due to processing power (no bouncing of pool balls)
- Limit to two players, only implementing one game mode (8-ball)
- solid coloured balls, cannot animate rolling

Peter Lin (998317548)
David McTavish (998449585)
Peng Xue (998477811)

IV. System overview

Pool Cue

The user will be holding the cue over the computer monitor (serving as the table). The user will press a button on the cue when he/she wants to take a shot and will move the cue to "hit" the cue ball. An led light will be attached to the end so that the cue's position can be tracked (via the camera).

Video Camera

A video camera will be positioned above the "table", facing downward to track the movements of the pool cue. It will send the video feed to a block of external memory, where it will be stored for further processing.

Position Locator Module

A custom IP block. This will read the video frame from the memory and compute the Cartesian coordinates of the led light. The status of the other controls on the cue (button, gyroscope, accelerometer) will be stored here as well. The cue's position and other parameters will be read from this module to the microblaze processor.

Microblaze Processor

The software running on the processor will keep track of the positions of each ball and any game logic. When it is a player's turn to take a shot, it will poll the position locator module to get the whether the cue is in use, if it is "colliding" with the cue ball, and the cue's speed and orientation at that time. This will give the cue ball its initial velocity, and then the processor will simulate the collisions of the balls and walls of the table. Based on the position of the balls, the processor will also create the a video frame of the table. This frame will be saved into another external memory block.

Computer Monitor

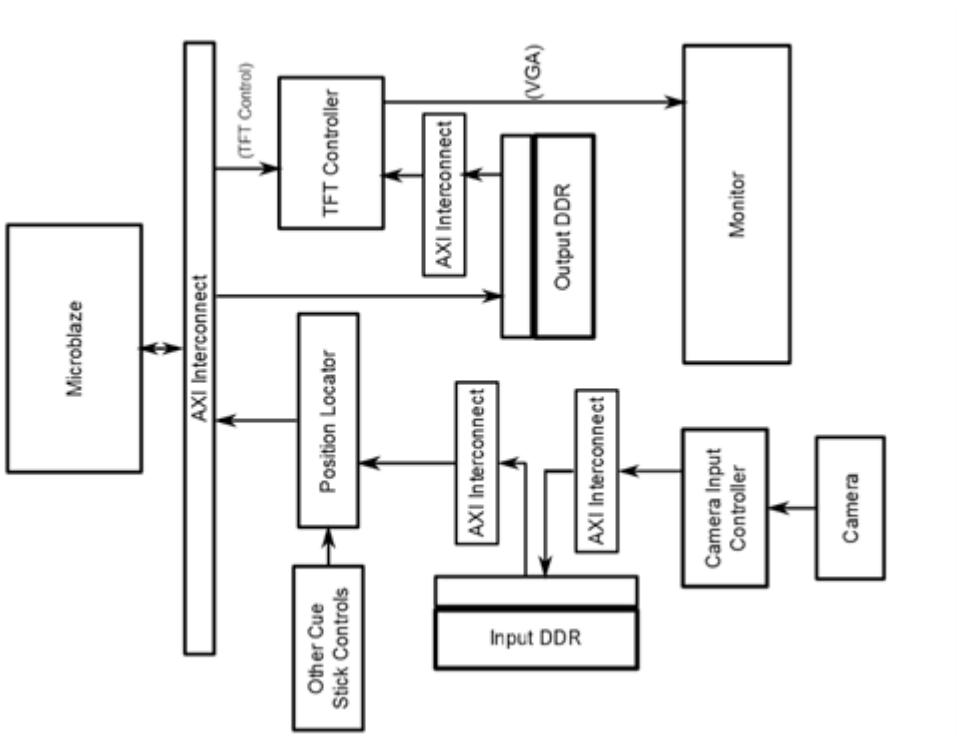
Will be laid face up, serving the role of the table. As new frames are written to the output memory, it will display the game state.

Peter Lin (998317548)
David McTavish (998449585)
Peng Xue (998477811)

External Memory (2 blocks)

The DDR memory on the board will be used to store the input video from the camera and the output video frames going to the monitor. Both memory blocks will be double buffered so that the new data can be written while the previous frame is being used. (While they will be treated as two separate memory blocks, they will be on the same DDR chip, the controller accessed through different memory buses.)

V. Block Diagram



VI. Testing

Pool Cue Tracking

This module will be tested separately in two parts. First, in simulation with virtual frames. Here, the tracking module will be supplied with a manufactured image, with a tracking point (to simulate the pool cue's LED), and we will verify that the tracking module can accurately

Peter Lin (998317548)
David McTavish (998449585)
Peng Xue (998477811)

determine the position of the point. Secondly, the camera will then record real video data. The position tracker module will read these video frames in real time and output the LED's position. The resulting position could also be drawn on the screen directly once basic rendering is done.

Physics Engine (Pool Cue and Cue Ball Interactions)
Initially, this will be tested in software. Sample positions of the pool cue will be generated, along with the initial cue ball position. The pool cue positions will be chosen to create a collision with the cue ball. The resulting collision of the pool cue and cue ball will yield a speed, and this velocity vector can be verified.

Physics Engine (Collisions)
This can be tested exclusively in software. The initial positions of the balls and the initial velocity of the cue ball can be set to create the following use cases:

- Cue ball colliding with one game ball
- A secondary collision
- Cue ball colliding with a cluster of game balls within the same time frame
- Any ball colliding with a wall(s)
- Any ball colliding with a pocket
- Any two or more of the above occurring in the same sampling period

The collision between the game and cue balls will be simulated, and their resulting velocities will be calculated and printed to console. The results can then be verified.

Rendering To Screen
A simple test of the drawing API, testing the ability to draw walls (thick lines) and circles (for the balls) on a background. This would start as static pictures, with no physics simulation. The positions of each ball will be predetermined for each frame.

VII. Risk

One possible problem is that the load on the microprocessor could be too high. Both the collision calculations and creating the output video may take up too much time, reducing the

Peter Lin (998317548)
David McTavish (998449585)
Peng Xue (998477811)

output video frame rate. There is little that can be done about the physics calculation, but if the video frame generation creates too much traffic to the processor, then the processor can just output the balls positions and a custom hardware block could use that to create the image in parallel.

Another potential issue is the tracking of the LED on the cue. Due to changing environmental conditions (eg. lighting, the image on the screen changing underneath), it may not be reliable. Alternative solutions could be using a different tracking sensor, potentially infrared or sonar.

One of the strength of our team would be our experience in programming and debugging FPGA designs. In addition, we are well versed in interacting with simulation tools such as waveforms and timing analysis to find errors in our design.

One weakness of our team is our lack of experience in making a realistic physics engine. Another possible difficulty for our team is creating a user friendly interface for the game.

VIII. Milestones

- **Milestone 1 (Feb 10)**
  - Write drawing API for basic shapes, output static image to screen using TFT controller
  - Implement a tracking algorithm in C (for LED position)
  - C structure for the storage of the position and velocity of each ball
  - Create physics for ball-pool cue collisions (sourced velocity vector)
- **Milestone 2 (Feb 24)**
  - Create physics for single ball-ball collisions
  - Create physics for ball-pool cue collisions (derive velocity vector)
  - Create test frame images (for tracking)
  - Design tracking algorithm to work on FPGA
  - Read video data from camera into DDR. Read video into position tracker.

Peter Lin (998317548)
David McTavish (998449585)
Peng Xue (998477811)

- **Milestone 3 (Mar 3)**
    - Create physics for ball-wall collisions
    - Create physics for ball collisions with pocket
    - **Integrate:** Generate video output from physics collisions
    - Implement double buffering in creating output video
    - Implement tracking algorithm on FPGA. Test using the same frames as the C version.
- **Milestone 4 (Mar 10)**
    - **Integrate:** Position tracker using the live video frames
    - Enable/Disable cue tracking when button depressed/released
    - **Integrate:** Create physics for deriving velocity vector from position tracker.
- **Milestone 5 (Mar 17)**
    - Implement double buffering in reading input camera video
    - Create physics for simultaneous collisions
    - Create game rule: Remove balls when they fall into the pocket
    - Create Game Rule: switching between player's turns
- **Milestone 6 (Mar 23)**
    - Create game rule: Striking opposing ball with cue ball
    - Create game rule: Losing when sinking the 8-ball inappropriately
    - Read gyroscope orientation. Implement backspin when striking cue ball.