6.111 Project Proposal

Matt Fishburn, fishburn@mit.edu
Hongyi Hu, hongyihu@mit.edu

November 4, 2005

1 Project Overview

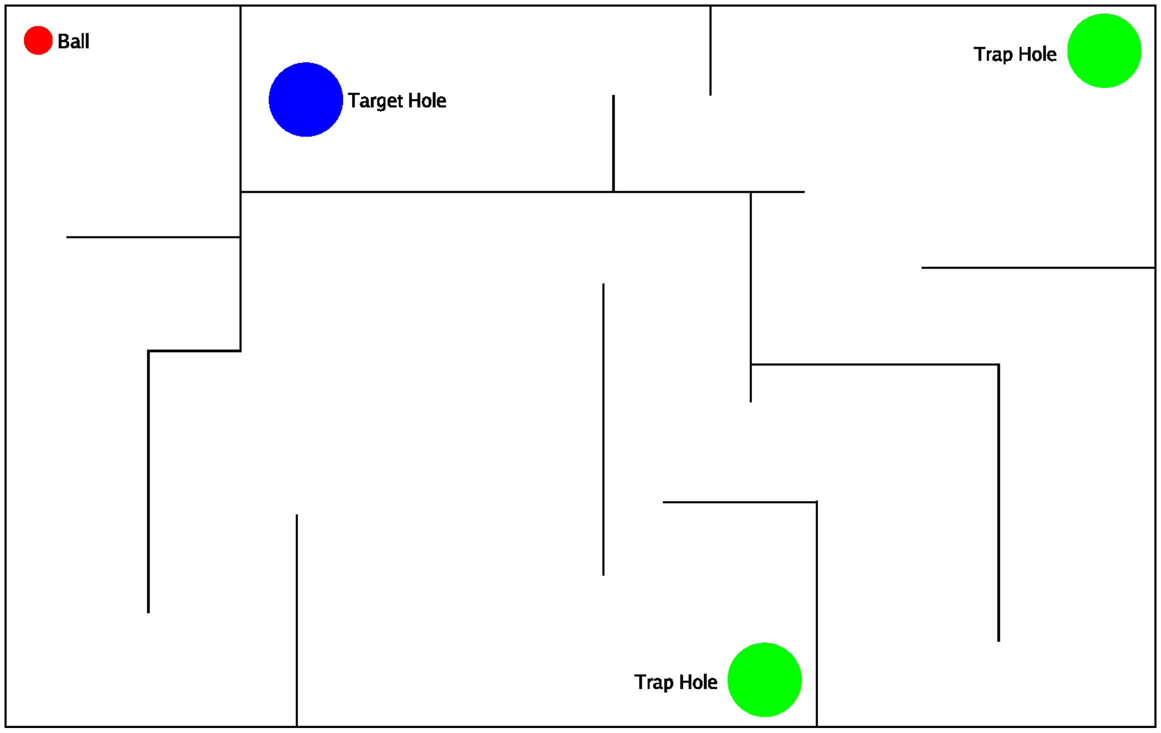

Figure 1: **Sample Screenshot -** A sample rendering of the expected video output for the final product.

This document contains a design proposal for an electronic tilting maze game. The project will be implemented using a 6.111 Virtex II labkit and a prototype accelerometer-driven user interface device.

The game will consist of a virtual ball on a virtual tilting maze board. The board will have a configuration of obstacles such as walls and trap holes, along with a target hole. The user will control the ball's movement by controlling the tilt of the maze board. The objective of the game is to navigate the ball to the target hole without falling through any trap holes.

The labkit will render the maze board, obstacles and ball. The labkit will track the physical state of the game, which consists of the board's tilt, the ball's velocity, the ball's position and game state information such as the current level. The labkit will update the game's state based on input from a three dimensional physics engine and a user interface device.

The user interface device will control how the user tilts the board. The device will consist of two two-axes accelerometers attached to an LCD. Based on how the user moves the LCD, the maze board shown on the screen should mimic the tilt of the LCD.

Potential issues include handling measurement error from the accelerometers in the UI device, interfacing the UI device to the labkit, the complicated rendering and shading of 3D objects on a 2D screen, and memory issues caused by multiple units requiring access to the same information. Possible extensions to this game include adding multiple levels to the maze board, introducing complicated obstacles such as autonomously moving enemies, and requiring multiple balls to be guided to the target hole. We hope to eventually implement this design on a smaller FPGA, creating a unit that can simply be attached to an LCD.

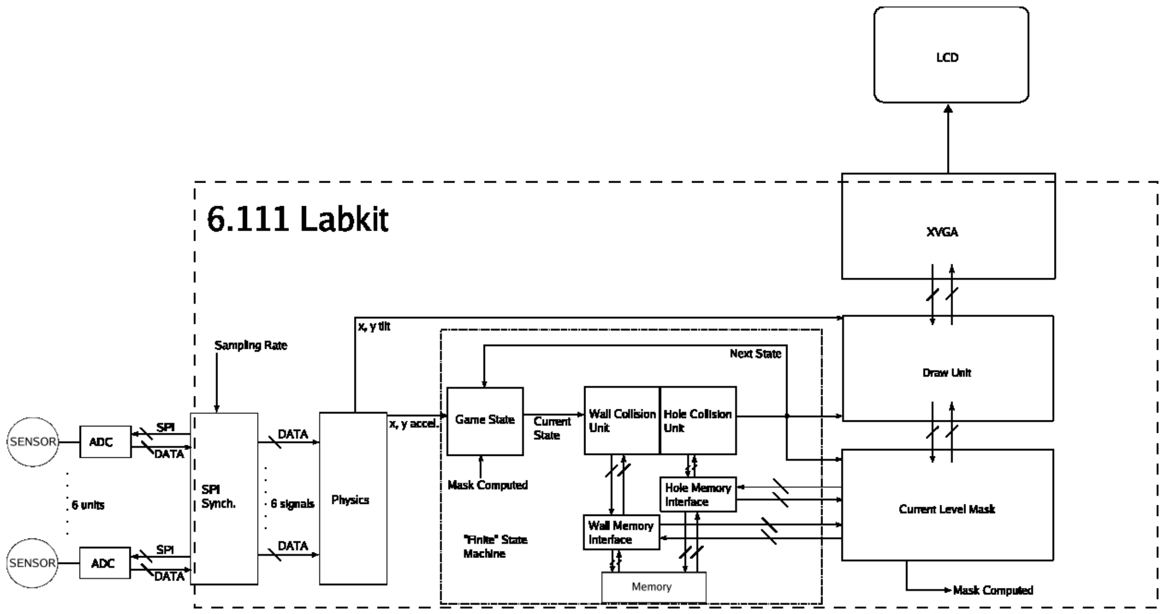

Figure 2: **Block Diagram** - The proposed design for the tilting maze game. Components inside the larger dashed box are inside the 6.111 labkit. The smaller dashed box represents the finite state machine.

2 Module Specifications

Our system design is composed of seven major modules: the user interface device, the SPI synchronizer, the physics engine, the finite state machine, the XVGA unit, the draw unit, and the current level mask.

2.1 User Interface Device

The user interface device uses accelerometer and gyroscope sensors that detect the motions of the player and transforms those motions into control signals for the game. The device feeds its output signals into a set of ADCs. These ADCs convert the analog signals into digital signals and send the signals to the labkit. The ADCs and labkit will communicate using the SPI protocol. The ADCs will output a 16-bit signal per sensor corresponding to the current orientation of the interface device.

2.2 SPI Synchronizer

The SPI synchronizer module acts as an interface for the external user interface modules such as the ADCs and the sensors. The module samples control signals passed in from the ADCs and synchronizes the signals with the labkit's internal clock. The SPI synchronizer module takes these control signals and a sampling rate signal as inputs, communicates with the ADCs via the SPI protocol, and the module outputs the synchronized and sampled version of these signals to the physics engine module. The outputs are six 16-bit synchronized and sampled versions of each control signal provided by the ADCs.

2.3 Physics Engine

The physics engine is responsible for computing the force of gravity on the ball caused by the incline of the user interface device. The engine takes the player's control signals that have passed through the SPI synchronizer. It makes the appropriate calculations and outputs x and y axis acceleration information for the ball to the FSM and tilt information for the virtual board to the draw unit. The x and y acceleration

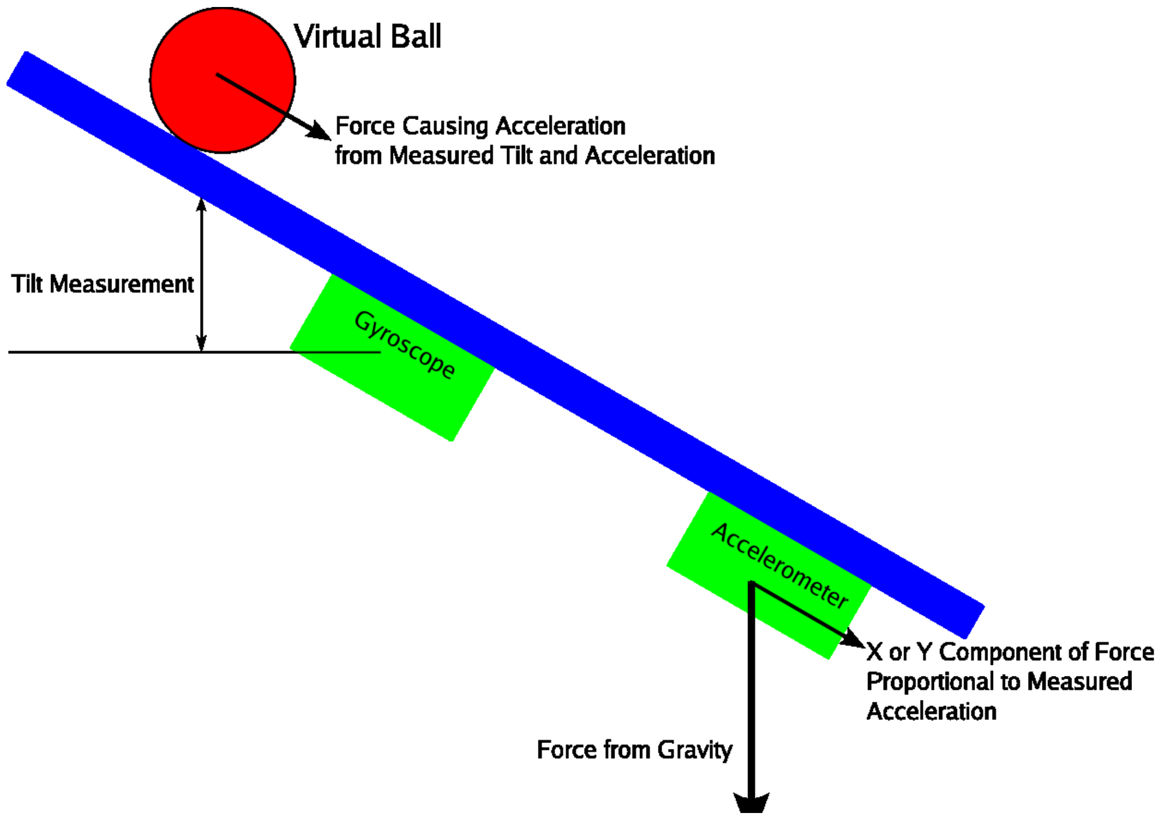
Virtual Ball
Force Causing Acceleration from Measured Tilt and Acceleration
Tilt Measurement
Gyroscope
Accelerometer
X or Y Component of Force Proportional to Measured Acceleration
Force from Gravity

Figure 3: **One Axis View of User Interface Sensors** - This picture documents the information recorded by the sensors in the user interface device. Force vectors describing the acceleration of the virtual ball are shown. The physics unit computes these force vectors from the sensor data.

are 32-bit signed fractional numbers corresponding to acceleration in Earth g's. The x and y tilt signals are 32-bit signed fractional numbers corresponding to the current x and y axis incline angles of the user interface device.

2.4 Finite State Machine

The finite state machine contains logic to drive the game. The FSM takes 32-bit x and y acceleration signals from the physics engine as well as level layout information from the memory. Actions in the game occur in discrete time frames, where each time frame represents a fraction of a second in real time. Each loop through the FSM updates the state of the game for the next time frame. Every state in the FSM contains information about the ball's current velocity, the ball's current position, the current level number, whether the ball has fallen into a trap hole, and whether the ball has reached the target.

To determine the state of the game for the next time frame, the FSM logic performs collision detection for the ball and any walls or holes in the game. Collision detection is accomplished by checking the path of the ball for the current time frame with the topology of walls and holes in the game. Information about the layout of these obstacles is accessed via the wall and hole memory interfaces.

The finite state machine is composed of six submodules listed below with their descriptions.

2.4.1 Game State

The game state unit stores the current state information of the game. It takes the 32-bit x and y acceleration signals from the physics unit and the 1-bit mask computed signal from the level mask. If the mask has not been computed for the current level, gameplay is suspended until mask computation is complete. Otherwise if the mask has been computed, the ball's path for the current time frame is computed using the x and y acceleration signals along the current velocity and position of the ball. This path information is a 48 bit signal that represents the start and end coordinates of the ball. Each coordinate value is stored in 12-bits of information. The 48-bit path signal and the 32-bit velocity signals for the ball are passed to the wall collision unit.

2.4.2 Wall Collision Unit

The wall collision unit takes a 48-bit path signal and two 32-bit velocity signals from the game state unit for the ball. The module determines if the ball's path for the current time frame will cause the ball to collide with a wall. Collisions are detected by looping over each wall and checking the wall's position with the ball's path. The location of each wall is accessed via the wall memory interface. If a collision is detected, the wall collision unit calculates a new 48-bit path signal and two 32-bit velocity signals for the ball and passes them to the hole collision unit.

2.4.3 Hole Collision Unit

The hole collision unit takes ball path and velocity signals from the wall collision unit and determines if the ball's path will cause it to fall through a hole. If the ball's path goes over a trap hole, the level is reset. If the ball will reach the target hole, then the game advances to a new level, and the state information passed is updated. Collisions are detected by looping over each hole and checking it with the ball's path just as with walls. The location of each hole is accessed via the hole memory interface. The hole collision unit passes the potentially modified game state information to draw unit, current level mask and back to the game state unit.

2.4.4 Wall Memory Interface

The wall memory interface reads locations of walls in the current level upon requests made by the wall collision unit and the level mask. The level mask's accesses have priority over the wall collision unit's accesses since gameplay is suspended if the level mask has not been completely computed. Wall locations are stored in memory using 24-bit start and end coordinates.

2.4.5 Hole Memory Interface

The hole memory interface functions similarly to the wall memory interface. It reads locations of holes in the current level upon requests made by the hole collision unit and the level mask. The level mask's accesses have priority over the hole collision unit's accesses since gameplay is suspended if the level mask has not been completely computed. Hole locations are stored in memory using a 24-bit position coordinate value and a 6-bit radius value.

2.4.6 Memory

The memory unit stores the locations of obstacles such as walls and holes for all levels in the game. It is accessed by other modules via the wall and hole memory interfaces. The memory is 80-bits wide and contains 8k locations. Each value in memory contains a 20-bit header field describing the level and type of object the value corresponds to and a 60-bit data field that describes the location of the object. The memory is read-only and has 13 address lines and 80 data lines.

2.5 XVGA Unit

The XVGA unit is responsible for communication between the ADV7125 and the draw unit. The XVGA unit will be similar to the XVGA unit from lab four, but the unit will rely on the draw unit instead of the pong_game unit.

2.6 Draw Unit

The draw unit is responsible for creating the pixel output to be displayed on the screen. It takes a 48-bit ball position signal from the FSM, 48-bit obstacle position signals from the level mask, two 32-bit board tilt signals from the physics engine and video signals from the XVGA unit as inputs. The tilt signals are used to determine appropriate shading to give the correct perspective of a tilting board in two dimensions. The draw unit outputs a three-bit color display signal back to the XVGA unit.

2.7 Current Level Mask

The current level mask contains information about the topological layout of the current level. The draw unit queries the level mask to correctly display the current level layout without directly accessing the memory. The level mask acts as a buffer for the memory to avoid draw unit latency and potential access conflicts if the draw unit and the FSM logic directly query the memory simultaneously. Until the level mask is computed, the mask computed signal is set low so that gameplay is suspended. The level mask takes a hcount and vcount signal from the draw unit and outputs back a three-bit color value for the corresponding pixel.

3 Testing

All individual modules will be rigorously simulated in isolation to check that they meet their specifications. The design will be divided into three separate systems consisting of the external sensor modules, the fsm logic and memory modules, and the display logic. Each system will have an integrated series of simulation tests to check that the system as a whole functions correctly. Finally, the entire design will tested using the labkit and a logic analyzer.

4 Division of Work

Matt will implement all external components and their interfaces along with the SPI synchronizer, the physics unit, the draw unit, the level mask unit and all modules related to video display. Hongyi will implement the finite state machine, collision detection units, and memory modules and interfaces.