WiiMaze

Design Document

March 13, 2008

Yezhen Lu (yl2194)
Brian Ramos (bjr2102)
Shaun Salzberg (sds2110)

Introduction
We are going to implement a simple maze game using the Altera DE2 Board, a Wiimote, a monitor, and some hard work. We will be using the USB Controller, the CPU, the RAM, the RAM controller, and the video controller in our design. The basic idea is that the Wiimote will communicate through a Bluetooth connection to a USB Bluetooth receiver. The receiver will then capture the inputs and write the result into memory mapped RAM. The game logic will read the contents of the RAM, update the game state, and will display the result onto a monitor using a sprite engine.

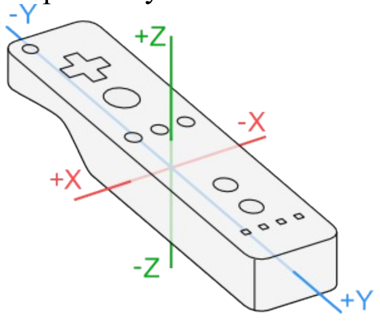
Wiimote Details
The Wiimote has a number of different data reporting modes which map to certain core data features of the remote, including data from external peripherals. These reports are sent as a stream of bytes with the following format:

```
            1 byte               1 byte     (1 byte each)
(data input packet id) (reporting id) (data) (data) ... (data)
```

It sends these reports to the host through one of the report ID's, which are determined by the particular mode the wiimote is in. We can set the wiimote to a reporting mode that will send reports only when data has changed and what channel to use via a reporting id. A list of the reporting ids that contain accelerometer specific data are as follows:

0x31: Core Buttons and Accelerometer
(a1) 31 BB BB AA AA AA
0x33: Core Buttons and Accelerometer with 12 IR bytes
(a1) 33 BB BB AA AA AA II II II II II II II II II II II II
0x35: Core Buttons and Accelerometer with 16 Extension Bytes
(a1) 35 BB BB AA AA AA EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE
0x37: Core Buttons and Accelerometer with 10 IR bytes and 6 Extension Bytes
(a1) 37 BB BB AA AA AA II II II II II II II II II II EE EE EE EE EE EE
0x3e / 0x3f: Interleaved Core Buttons and Accelerometer with 36 IR bytes
(will not be used since this contains a lot of superfluous data)

Each triple AA AA AA corresponds to the acceleration in the x, y and z directions, respectively. The axes can be seen in the figure below.



The linear accelerometer (Analog Devices ADXL330) measures linear acceleration relative to a free fall frame of reference. Thus, the acceleration data returned by the wiimote will have to be translated at the host to correspond to moves on the screen.

In addition to relying raw acceleration data provided by the wiimote, the motion calculations will have to make use of the zero point and gravity values that the particular wiimote is calibrated for. This calibration data is stored on the wiimote's onboard flash memory, which can be read in a similar fashion to the accelerometer data. The output report must first be set to 0x17, which tells the wiimote that the host will be reading data. This command is formatted as follows:

```
(52) 17 MM FF FF FF SS SS
        where MM is a bitmask that selects the address space (flash or control registers)
                FF FF FF is the offset
                SS SS is the size to read in bytes
```

Once this is set, the data can be read through input report 0x21 in the following format:

```
(a1) 21 BB BB SE FF FF DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD
DD
        where BB BB is the state of the wiimote buttons
                SE is the size in bites and error flag respectively
                FF FF is the offset
                DD .. DD is the actual data
```

The calibration data that we need will have to be read from the following addresses:

- 0x16: 0 point for X axis
- 0x17: 0 point for Y axis
- 0x18: 0 point for Z axis
- 0x1A: +1G for X axis
- 0x1B: +1G for Y axis
- 0x1C: +1G for Z axis

There are some C api's that enable a programmer to read accelerometer data, as well as flash memory, from a bluetooth connected wiimote, such as libwiimote and cwiid. They provide basic connectivity to a wiimote (given an established HID bluetooth connection) but have limited functionality that we will have to expand on.(For example, libwiimote can read accelerometer data but provides no functionality for reading flash memory).

Wiimote References:
http://wiibrew.org/index.php?title=Wiimote
http://www.wiili.org/index.php/Motion_analysis
http://libwiimote.sourceforge.net/
http://abstrakraft.org/cwiid/

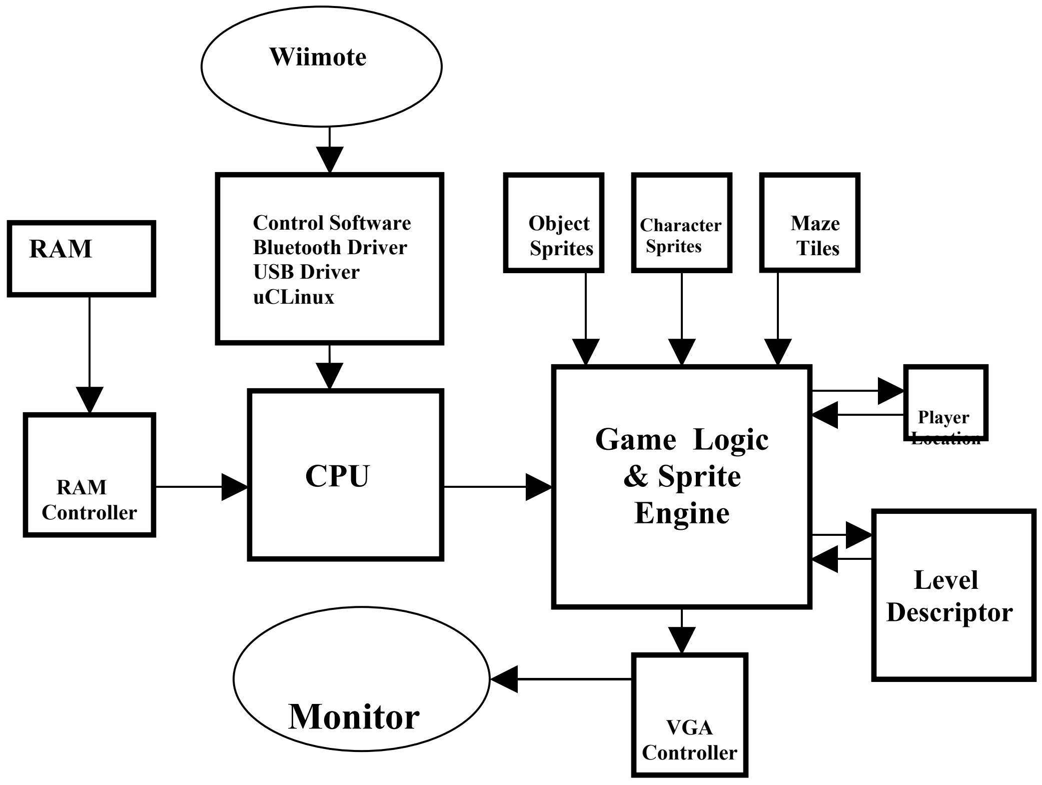
Software Details
We will be communicating with the Wiimote through a Bluetooth USB receiver. We will write a program that uses the open source Bluez stack to 1) establish a Bluetooth connection with the Wiimote and 2) use the HID profile of the Bluetooth spec to receive inputs from the accelerometer. We're interested specifically in the accelerometer inputs and will intercept this with software and write to a memory-mapped SRAM location that can be accessed directly by the hardware.

Game Logic

The game that we create will be a simple, on-screen maze game. Each level will consist of a maze in the background with a Wiimote-controlled character. The character will begin in the upper left corner; when the player tilts the Wiimote to the right, the character will move right, and when the player tilts the Wiimote left, the character will move left. If the character ever stands directly on top of an opening in the ground, it will fall down until it reaches the next solid floor. There will also be colored keys and colored doors throughout the maze. When the player moves on top of a colored key, it collects it and all doors of that particular color will open. Each level will essentially be a puzzle – the player will have to navigate the character through the maze to the endpoint without getting stuck.

We will implement the majority of this game in hardware. We will have multiple areas of the hardware devoted to storing our pre-programmed maze tiles, object sprites, and character sprites. There will also be an area to store the layout of the current board (i.e. which maze tiles go where), as well as a memory location which will store the current location of the player. All of these things will interact with a module containing the actual game logic.

Block Diagram



Milestones
The first milestone for the project will be to compile a version of ucLinux for the DE2 board with drivers for Philips ISB1362 USB Controller as well as the Bluez bluetooth stack so the Bluetooth tools are present.

The second milestone of the project is to be able to write a program in uCLinux that could use the Bluez stack and tools to establish a connection to the Wiimote and also to display the game board sprites onto the screen

The third milestone for the controller part of the project is to be able to intercept accelemeter motions from the Wiimote and display a simple sprite moving on the screen.

The final deliverable of the controller part will be a program that can respond to the Wiimote input, write it correctly to a memory location that will be picked up by the game logic, and can correctly display the sprites correctly based on the input.