CSEE 4840 Embedded System Design
Project Design

NUNY: Ninja University in the City of New York

Kshitij Bhardwaj, Van Bui, Vinti, and Kuangya Zhai
{kb2673, vb2363, vv2236, kz2219}@columbia.edu

I. Project Introduction

In this project, we will design and implement a Fruit Ninja like video game on the FPGA. Fruit Ninja is a popular video game where the player slices fruit using their finger(s) on the touch screen. The theme of our game will be based on undergraduate/graduate school life so that rather than slicing fruit, the object of the game will be to slice assignments, exams, qualifiers, advisor meetings, paper deadlines, thesis writing, food (like pizza), coffee, books, etc. The game will generate several moving objects on the screen and the player will destroy objects using an on screen blade or sword controlled by a wiimote controller.

NUNY will have three levels to the game representing each stage of higher education (i.e. bachelors, masters, and doctorate). Each level will have different program requirements and students are required to complete their bachelors degree before moving on to their masters degree and so on. The student will have to earn a minimum GPA and pass any other program requirements to achieve a particular degree. There will be several objects appearing and disappearing from the screen and the player will have to slice certain objects in order to increase their GPA score or complete some program requirement. There will also be objects that the player should not slice as it could either lower their GPA or will hurt their chances to graduate. There will be a three strikes rule, so that if the player slices an object that represents for example cheating, than that will be one strike against them. The entire game is won when the player completes their doctorate degree.

II. Software and Hardware Components

The major components in our design includes the game logic, video display, audio, and wiimote controller.

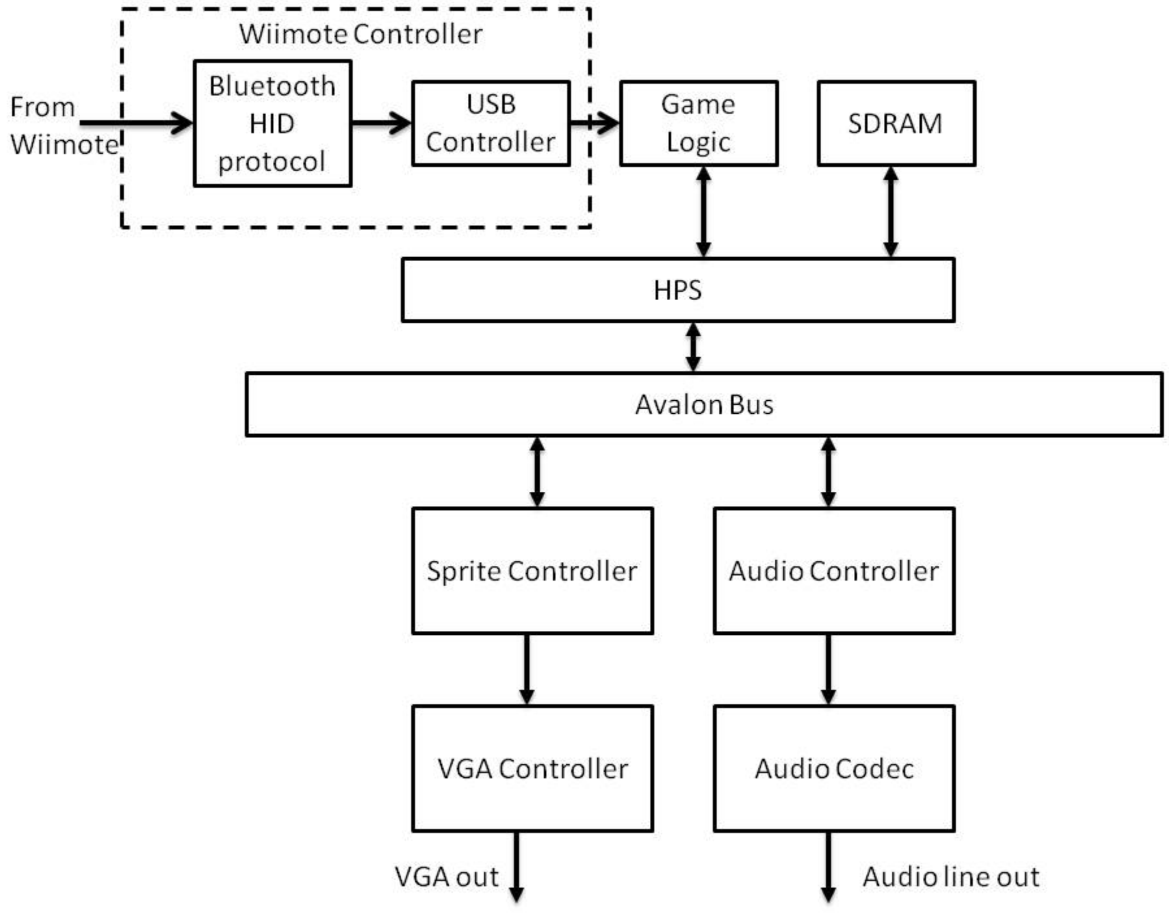

Figure 1. High level software and hardware design components

Game logic controller (Software)

Game logic will be implemented in software using C programming language. The key functions of the game logic controller are to control the generation of sprites (graphics), read location of Wii pointer through the Wiimote controller, generate appropriate audio when required during the game by interacting with the audio controller and finally implement the actual game logic, its rules and compute the player's GPA, based on how many program requirements he/she has fulfilled (or sliced!). Each of the above functions are implemented as a submodule of the game logic controller. As shown in the figure below, there are 4 submodules, which are described in more detail next.

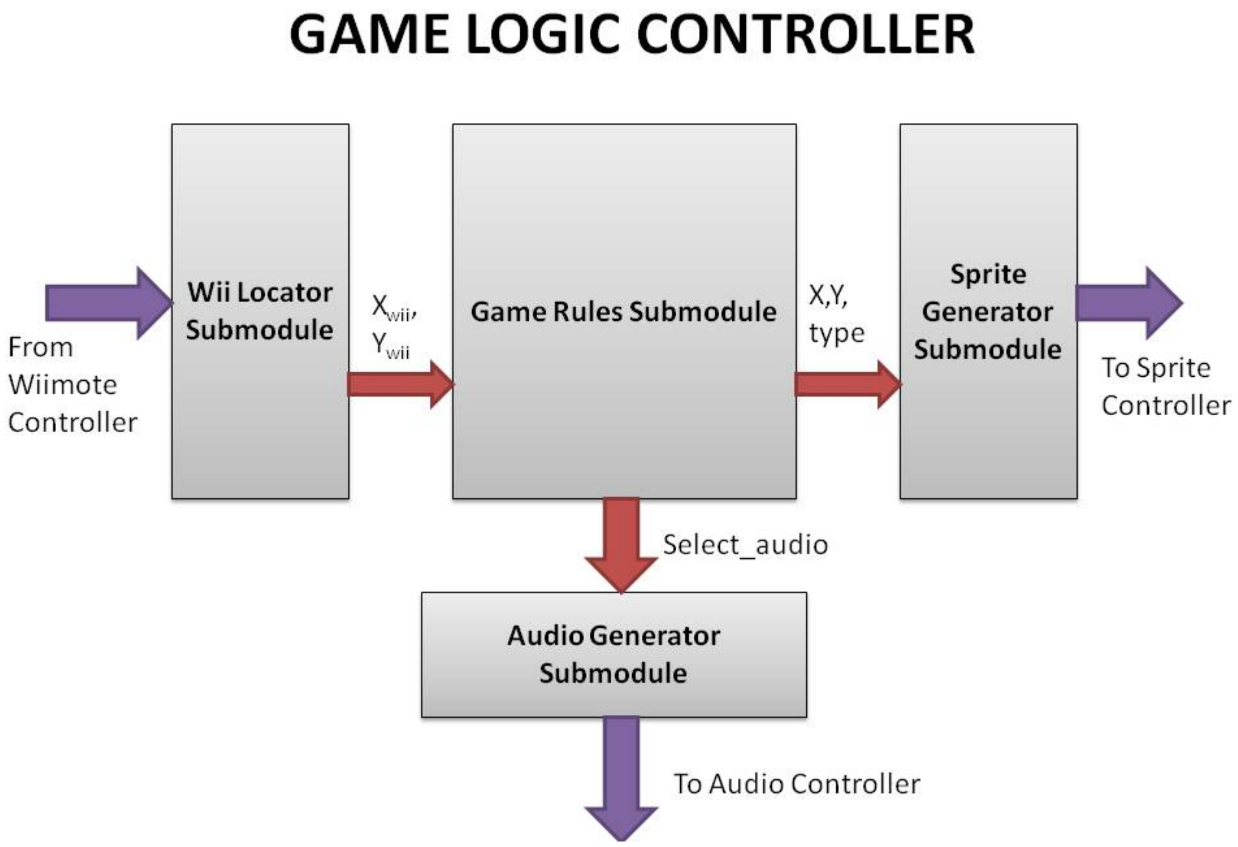

Figure 2. Game Logic Controller block diagram

1. **Game rules:** This is the main submodule of the game logic controller and it interfaces with all the other submodules, instructing them what to do and when based on the rules of the game. For example, to create the screen where a player selects a program (Phd, MS, or undergrad), the game rules submodule tells the sprite generator submodule to generate sprites such as “MS” / ”PhD” etc. It also tells the audio generator to interact with the audio controller to generate the background music for this opening screen.

   This submodule is responsible for the dynamic behavior of the game and keeps updating the screen according to the game being played. It also implements the logic to determine the speed of the various sprites on the screen. It calculates the final GPA of the user by mapping the no. of program requirement sprites he/she has sliced to an actual GPA score for the whole semester.

2. **Sprite Generator:** Based on the game logic, this submodule generates the X and Y coordinates of the different sprites that need to be displayed on the screen. These X and Y coordinates for each sprite are stored in memory (using iowrite calls), which gets

updated according to the actual game logic. This memory is accessed by the sprite controller through the address bits, which then displays the necessary sprites on the screen.

The X and Y coordinates for the moving sprites will be determined based on the current time step, velocity in the x and y direction, gravity, and the initial x and y coordinate positions.

x(t) = x_velocity * time + x_init
y(t) = -½ * gravity * time^2 + y_velocity * time + y_init

3. **Wii Locator:** Game logic controller interacts with the Wii controller to determine the location where the Wiimote is pointing. The Wii locator submodule also interacts with the game rules submodule (which then talks to the sprite generator) to select the appropriate sprite based on the Wii location. For example, if the X,Y coordinates obtained from the Wii controller (which are the coordinates of the sword) are within the dimensions of a sprite (say the homework sprite) then the homework sprite needs to be updated to a new sprite which shows a sliced homework. A more simpler example will be the movement of sword, which is displaying a sword sprite at the exact position where the Wii is pointing.

4. **Audio Generator:** Various audio sounds that need to be generated throughout the game (background music, slicing sounds, etc.) are encoded inside the audio generator submodule. Based on the game logic, this submodule tells the audio controller to generate the appropriate sound while the game is being played. For example, if the player successfully completes a level, the game logic will tell the audio controller to play the graduation music.

VIDEO DISPLAY CONTROLLER

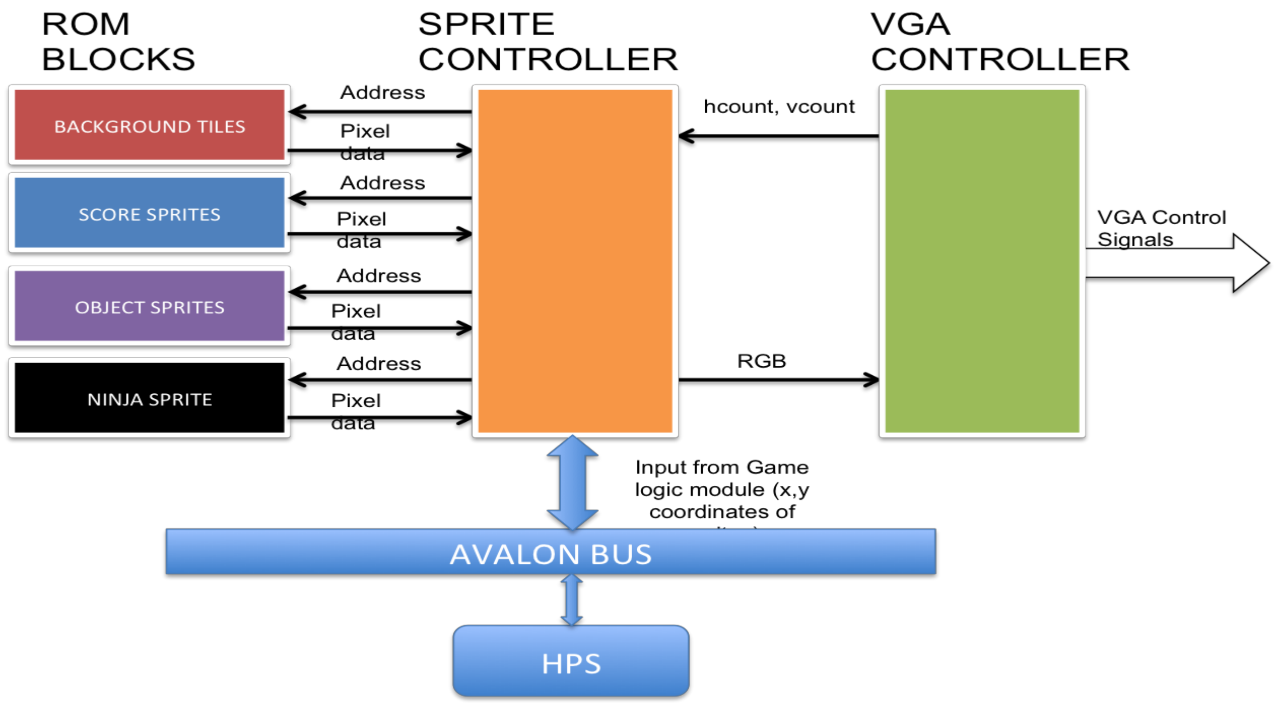

Figure 3. Video display controller block diagram

Video Display controller

The video display controller will have two submodules, the VGA Controller and the Sprite Controller. For the VGA Controller, we will be using the module from lab3 as a starting point. The Sprite controller will be implemented using two line buffers.

1. **VGA Controller:** This module will generate the VGA signals and also the hcount and vcount values that will be needed.

2. **Sprite Controller:** The sprite controller will send the RGB values of each pixel (depending on the current hcount and vcount value) to the VGA Controller. The inputs for the sprite controller will be the following:

    - Hcount and Vcount (current position of the pixel)

- X and Y coordinates of the Ninja
- X and Y coordinates of the object sprites

The game will consist of 4 layers. The order of the layers will be following:

- The background layer will have the lowest priority
- the score display layer comes next
- the object layer will be next and will have 6 to 8 sub layers, depending on the difficulty level of the game
- The topmost layer will be the ninja and it will have highest priority

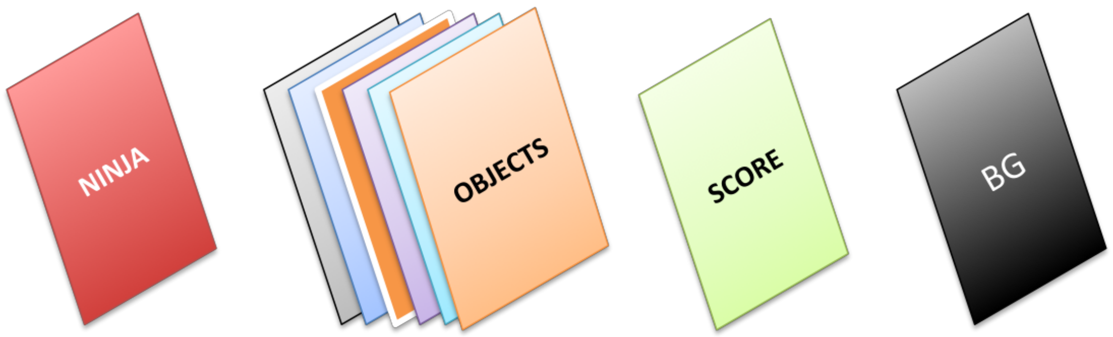

Figure 4. VGA Display Layers

The sprite controller submodule will get inputs from the game logic controller specifying the position of the sprites on the screen. It will have two line buffers of size 640 (for each pixel in a line of the VGA screen). At a given time, it will write the value of each pixel in this line buffer and read out the other line buffer to the VGA Controller. The read out will be at a clock frequency of 25MHz i.e. the VGA_CLK and for writing into the redundant line buffer, the 50MHz clock will be used.

Line Buffer Write Operation: The sprite controller will check the priority flags assigned to each of the layers and sublayers for each pixel in a line and then depending on the highest priority layer, fetch the corresponding sprite pixel. It will then check if the pixel is transparent or not and if not, write it into the line buffer, at the specific hcount position. To do this, we will have 1600 clock cycles per line.

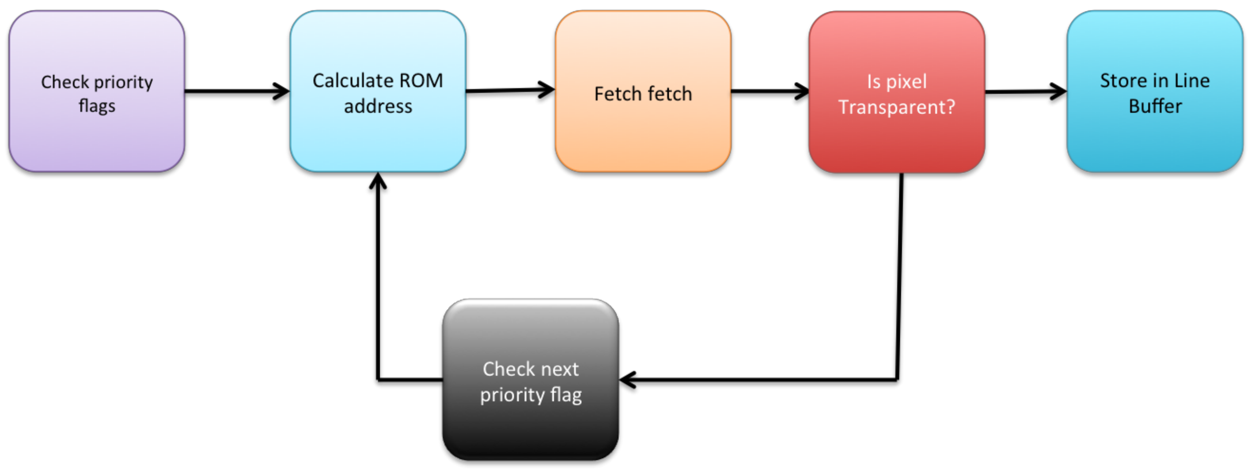

Figure 5. Flow chart for the line buffer write operation

Memory Budget:

Block	**Number of Sprites**	**Pixel size**	**Size(per block ROM)**
Background	~10 (.approx)	32x32	3.072KB
Score	~10	32x32	3.072KB
Objects	8+8	32x32	49KB
Ninja	4	32x32	12KB

Total = 67 KB (approx)
Each pixel is represented using 24 bits (8 bits each for R G B).

Memory requirment for game logic: 36Bytes (9 (8 objects + ninja) x 32 (16 bits each for X and Y coord)

Audio Controller

The SoCKit board supports 24-bit audio with the Analog Devices SSM2603 audio codec. SSM2603 has ports for microphone in, line in, and line out. The sampling rate supported is 8 KHz to 96 KHz and is adjustable.

NUNY will support sound for fast movement of the blade, object slicing, appearance of a new screen objects, game over, and graduation music for when the player passes a level. We will use 16-bit audio data to generate sound. We will experiment with different sampling rates starting with the lowest rate of 8 KHz and increase it as necessary for good quality sound.

The hardware components on the board that will support audio in our game includes the HPS and the SSM2603 audio codec. The HPS will retrieve audio wave files stored in DRAM and forward the data to the audio controller. The audio controller consists of a FIFO buffer submodule and the SSM2603 audio codec. The FIFO buffer submodule stores the audio data and is connected to the SSM2603 DAC. The SSM2603 retrieves audio data from the buffer and sends it through the audio line out port interface to produce sound. When the music buffer is empty, the codec will signal to the HPS that the buffer is empty.

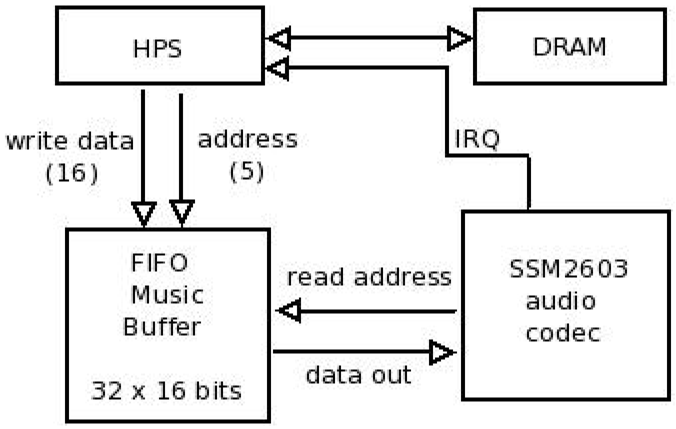

Figure 6. Hardware submodules for audio design

For the audio, we will support sounds that last anywhere between one second to several seconds long. The sound effects such as the sound of a sliced object, blade swiping, signaling a new screen object, and game over are expected to last about 1-3 seconds. Assuming the lowest sampling rate of 8 KHz and 16 bits per sample, the total amount of memory for the sound effects should be about 128 KB. The audio for the graduation music is expected to last longer, possibly 30 seconds, which would require as much as 480 KB. Sound data will be stored in DDR3 SDRAM on the HPS, which is 1GB in size and is plenty of space to store the audio data.

Wii Controller

There are three devices needed for the Wiimote Controller model: (i) Wiimote, (ii) Bluetooth USB Dongle, and (iii) Sensor Bar. The sensor bar emits infrared signal when powered and will be placed in front of the screen. The Bluetooth dongle connects to the SoCKit board through the USB interface and standard Bluetooth HID protocol, and receives Bluetooth signal sending from the Wiimote controller. There are two sensors built in the Wiimote: the accelerometer and the front digital camera. The accelerometer senses the acceleration of the Wiimote and the front digital camera senses the relative position of the Wiimote to the sensor bar. The Wiimote then sends the acceleration and position information to the SoCKit board through the Bluetooth USB dongle.

We will be using BlueZ [1] as the Bluetooth stack to communicate between the Wiimote and Linux host. libwiimote [2] is a C-library build on BlueZ that provides a simple API for communicating between the Wiimote and the Linux host. We can get the data of the accelerometer and ir-sensor of Wiimote by calling functions provided by libwiimote directly and save huge effort of doing nasty math computations. In this project, we will be using BlueZ and libwiimote together to make the developing of Wii Controller module easier.

Revised Milestones:

Milestone 1: Initial integration of the audio, video and game logic modules.

Milestone 2: Integrate wii controller code to the existing code base. A "Hello World" version of the game.

Milestone 3: Implementation of the game with three levels of difficulty. Test that the game console works properly via simulation and real-time testing.

Reference

[1] BlueZ, "Official Linux Bluetooth protocol stack." http://www.bluez.org, [Online; accessed 27-March-2014]
[2] libwiimote, "Simple Wiimote Library for Linux." http://libwiimote.sourceforge.net, [Online; accessed 27-March-2014]