Percussive Visualizer / Body Drums

Germain Martinez, Gerzain Mata, Michelle Qiu
6.111 Final Project Proposal
Fall 2015

Overview

The rhythm game genre of videogames has been around for years. Since the release of games like beatmania, Guitar Hero, and Dance Dance Revolution in the 2000's, rhythm games have had one common issue: The consumer often has to buy external peripherals to play the games. Some rhythm games use more common peripherals, like computer keyboards and touch screens, to mitigate this issue. However, they don't completely eliminate this "barrier" that keeps new players from playing because it can take awhile to figure out how to use the peripherals.

The goal of our project is to create a rhythm game that can be enjoyed without having to master some strange peripheral. We hope to reduce the learning curve by having the player make different bodily sounds, like slaps, claps, and snaps to the beat displayed on a screen. These sounds will be picked up by a microphone and interpreted by the game depending on their frequency content. Higher-pitched sounds like finger snapping will hit one kind of note, and lower-pitched sounds like tapping a table will hit another kind of note. The hope is that the player can gain an intuitive understanding of rhythm through kinematic motion.

Block Diagram

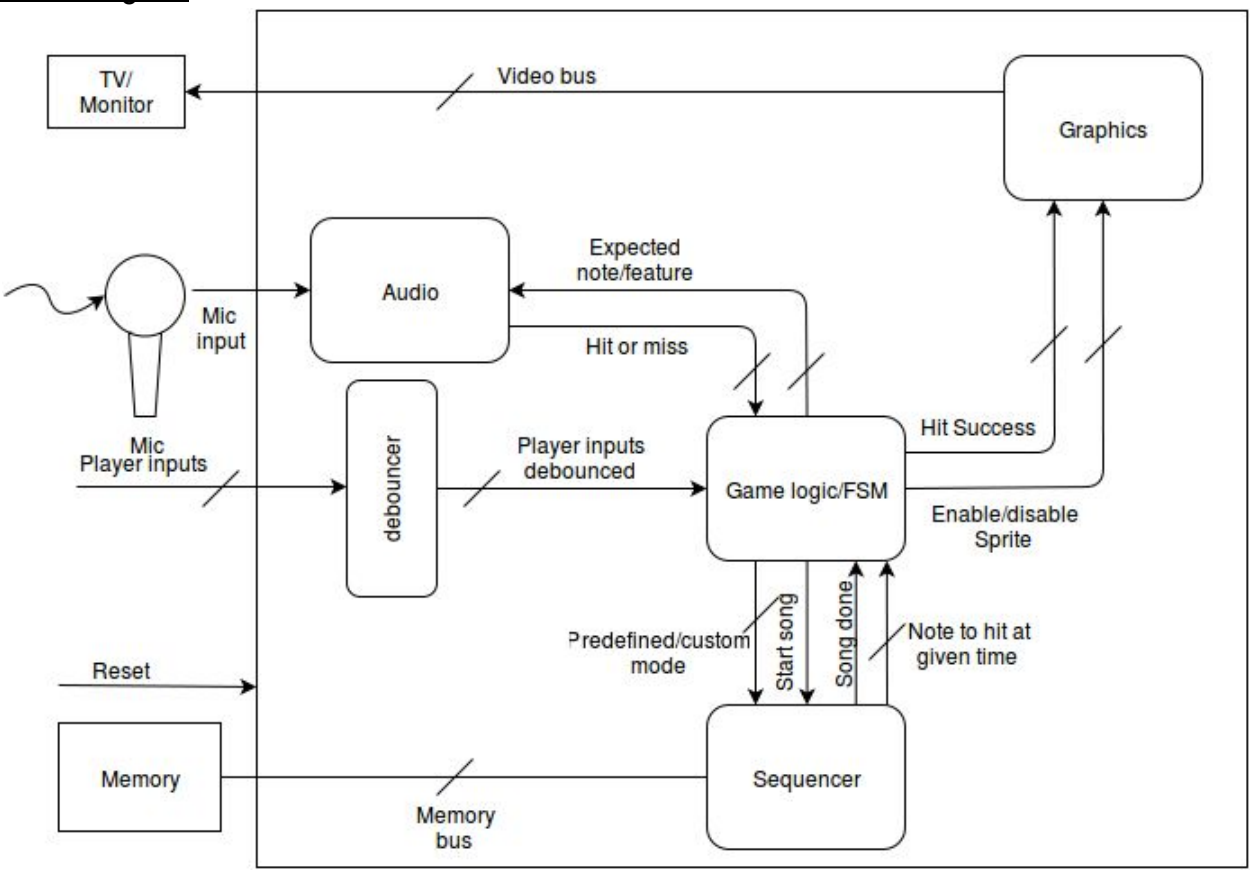


Figure 1: Overall block diagram

The audio module will extract features from audio (lows, mids, and highs), determine if input frequency approximates expected output result, and lets the game FSM know if a note was hit or miss. The graphics module will display the game information to the player - the notes that the player must hit, the state of the game, and whether the notes have been hit or missed. The Sequencer will read/write game song files from/to memory and relay the information to the game FSM. The game FSM/Game logic determines what mode the game is in, controls and processes information from the audio and sequencer logic, and determines what the graphics module should display.

Design

The game is expected to work in the following fashion. The player selects a song available from memory in a menu screen mode. The game switches to the “game mode” and the screen changes. The game will then read off saved data for the songs from memory using the sequencer module. The saved data contains different notes corresponding to the different sounds the player has to make. The game will draw scrolling graphics on the screen corresponding to these notes in real-time.

As these graphics scroll down the screen, they will eventually hit a bar on the bottom of the screen, which we will call the Judgment Line. At this time, the player must generate the

sound that corresponds to the note that hits the Judgment Line. These notes can correspond to a slam, a clap, and a finger snap as examples. A microphone will pick up this sound and output it to the sound module. The sound module will filter the sound to decide what the sound is, and then this decision goes to the game logic. If this decision matches what the game expects at the Judgment Line, then a “success” graphic is drawn on the screen; otherwise, a “note missed” graphic will show up.

The design will be split into multiple modules to facilitate distribution of work and debugging processes. Germain will be responsible for the module that handles audio inputs, Michelle will work on interfacing the sequencer and memory modules, and Gerzain will be tasked with handling the graphics output. All group members will cooperate on handling the game logic so that their individual modules communicate correctly with each other and with the game logic.

Module Implementation

There will be 3 main modules that will interact with the game logic (ideally this will be a finite state machine) which are as follows:

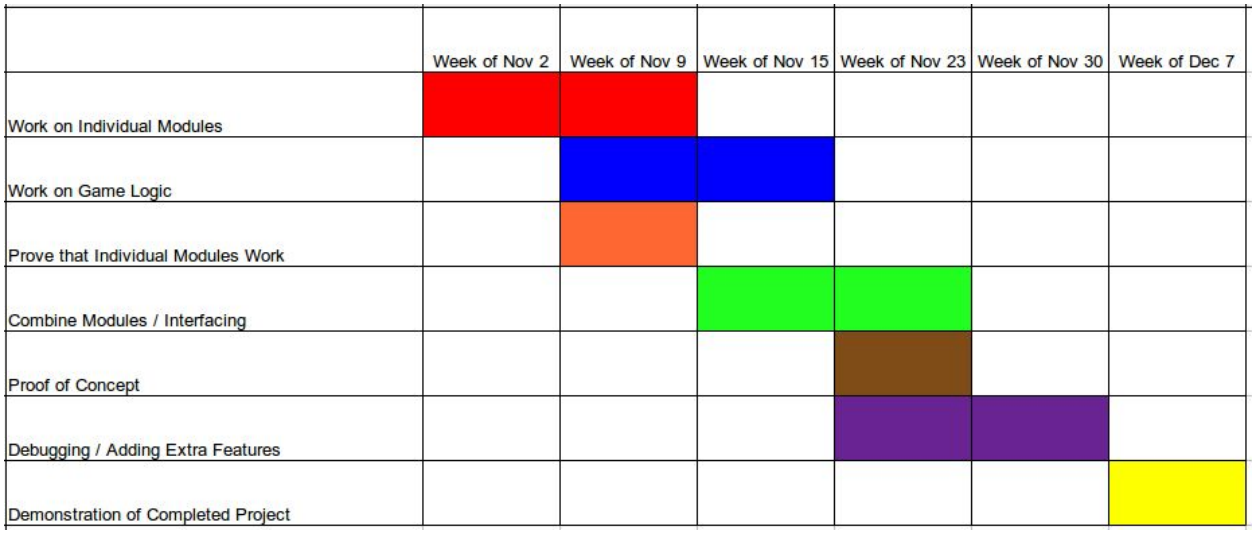
Sound Input Module: The input from a microphone will be processed in the frequency domain. This input will be converted from an analog waveform to the digital realm using the AC97 sound board on the 6.111 Labkit. The objective is to process a limited set of frequency bands (e.g. low, mid, and high frequencies). The module will use either a set of FIR (Finite Impulse Response) filters or a FFT (Fast Fourier Transform) submodule. These groups will perform the processing of the raw audio samples. The module will also take in a predefined frequency range that will serve as reference to compare the input sounds against. If the input sound’s frequency content falls within one of the predefined frequency ranges, then the module will output a SUCCESS signal for that particular frequency range.

Memory Encoder/Decoder Module: This module will be responsible for writing and reading to the available memory banks on the FPGA. Depending on the game mode (CUSTOM / PREDEF), the encoder will write the sequence processed from the microphone and the *Sound Input Module* to memory or otherwise output the predefined sequence to the game logic. To facilitate initial development of the project, songs will be separated into specific memory locations (i.e. each song will be limited to 1024 bytes of memory). The module will be tasked with reading successive memory locations when the START_SONG signal is pulsed high. Once the song has been read in its entirety, the module will pulse the FINISHED signal. This can either be triggered when 1024 bytes have been read or when the encoder reads 0xFF from a memory location.

Graphics Output Module: This module takes in information from the game logic indicating whether a note was hit successfully within the given time tolerance (±35 milliseconds). This module will also take the sequence input from the game logic and output the correct notes the player is required to hit. The Judgement Line will momentarily flash green if the player correctly hit a note when the it crossed the line; otherwise, the Judgement Line will flash red when the player either fails to hit the correct note or the player does not hit any note (player “misses” a note). Optionally, the *Graphics Output Module* can also keep track of the current score and output the score on the screen (if time allows).

Game Logic Module: This module controls and translates signals so that they can be used by the 3 other main modules. This includes reading the note sequence generated by the sequencer, feeding that information when necessary to the audio module, reading whether each note was hit or missed, relaying the information to the graphics module so that it knows what to display, and sending the correctly encoded notes back to the sequencer when in a record mode. In addition, this module controls and transitions the game state and mode based on inputs from the user and other modules.

Project Timeline



Week	Tasks
Nov 2nd	Basic graphics (scrolling notes, simple rendering) ; reading / writing to memory; sampling from microphone, modeling discrete-time filters
Nov 9th	Incorporating images in graphics; encoding patterns to data; implement filters, sound module proof-of-concept; incorporate game logic
Nov 15th	Incorporate game logic; graphics finalized; memory / sequencer completed; filtering finalized
Nov 23rd	Graphics, sound, and memory modules synchronized to game logic; project behavior finalized; debugging
Nov 30th	Final debugging, meeting stretch goals (time permitting)
Dec 7th	Demonstration of Completed Project

Testing

For testing, we will create testbench modules to verify that a particular piece of the project will work. For instance a test bench module will be created demonstrating functionality of the memory sequencer, whereby one sends ENABLE, WRITE, READ, and START signals and the module processes memory read/ writes successfully. Another testbench could be created whereby sequences can be created and the graphics module should successfully output sprites/images onto the screen successfully at every rising DRAW_CLK edge. Once the sound modules have been completed, we will make a test project that will tell us if the code can pick up the microphone audio and discern between the different sounds. On the FPGA the sequencer/memory module can be tested in real time by using the switches as the inputs and manually checking to see if the output is as desired.

Resources Needed

Most of the hardware necessary for the project is available in the 6.111 Labkit. The only aspect that could be of concern is the sensitivity of the microphone. Depending on the success of the individual sound module, it might become necessary to acquire a 3.5mm microphone of better quality. Everything else can be created from the FPGA.

Stretch Goals

These are our stretch goals that we are thinking of implementing once we have the project working at the bare minimum. They vary in feasibility and complexity, and we will not be able to implement all of them. However, these serve as a good reference for making our project more complex. Modules in parenthesis are modules relevant to the goal.

- Fancy graphics read from a memory -- We would like to be able to read images from memory and use those as graphics for the game (graphics)
- Improve filter to be awesome (audio)
- Translate recorded audio into a sequence that can be played back -- This would bring the game to a whole new level. One could possibly take in a raw sound file from memory, process the audio through an instance of the *Sound Module*, and finally use that processed data as the sequence the player should play. (sequencer and audio and FSM)
- Play music audio to go along with song -- This could possibly be “hacked” if the aforementioned bullet point is completed. One could output the sound to the FPGA and to external speakers. - this could also probably be synced up with the beat sequence by having both the audio and beat sequence be triggered by the “start song” signal - so they can be relatively separate from one another (sequencer)
- Display “song info” - for example, song # or potentially title or length or how far into the song we are (sequencer and graphics and FSM)
- Song “high score” (sequencer and graphics and FSM)
- Changing the rate at which notes are read (sequencer, FSM, graphics)
- “game over” mode - miss enough notes (FSM and graphics)
- Wider range of filter inputs, like more than 3 notes (audio, FSM, (sequencer))
- Changing the scroll speed of each note (FSM, graphics)
- A “pause game” feature (FSM, graphics, sequencer)
- A “quit game” feature (FSM, graphics)

- Graphics mode/settings? aka change a sprite from one to another (FSM, graphics)