Recorder Hero

6.111 Final Project Proposal

Andrés Romero, Paul Hemberger

November 4, 2012

1 Abstract

We aim to create an interactive music game featuring real instruments. The game will contain a number of songs that a user can play along with. The game mechanic will be styled after Guitar Hero: while a song is playing its notes will move across the screen and the player must play the notes at the correct time. However, unlike Guitar Hero, which confines itself to five "notes" and a plastic controller, we aspire to let users play along with real instruments instead. Gamers will be able to both engage with their music and gain actual musical skill.

By taking a FFT (Fast Fourier Transform) of a live instrument, the game will identify the pitch being played and score the note on its pitch and timing accuracy. The game will let users play along with almost any monophonic instrument.

2 Design Overview

The player will use a simple instrument, such as a recorder, to play notes in a rhythm-based music game. The game will load desired notes from memory, and display them on the screen. Those notes will be streamed right to left on the screen. Once a note reaches the left edge of the screen it will have a small denoted active region indicating that the player should play that note. The game will show how successful the player was in correctly timing the note by increasing their score proportionally to the time the pitch was correct. The current pitch will also be shown on a scale on the top-right corner of the screen to help the player learn their instrument.

Recorder Hero will be built from four core components: note identification, musical score loading, game logic and video display logic.

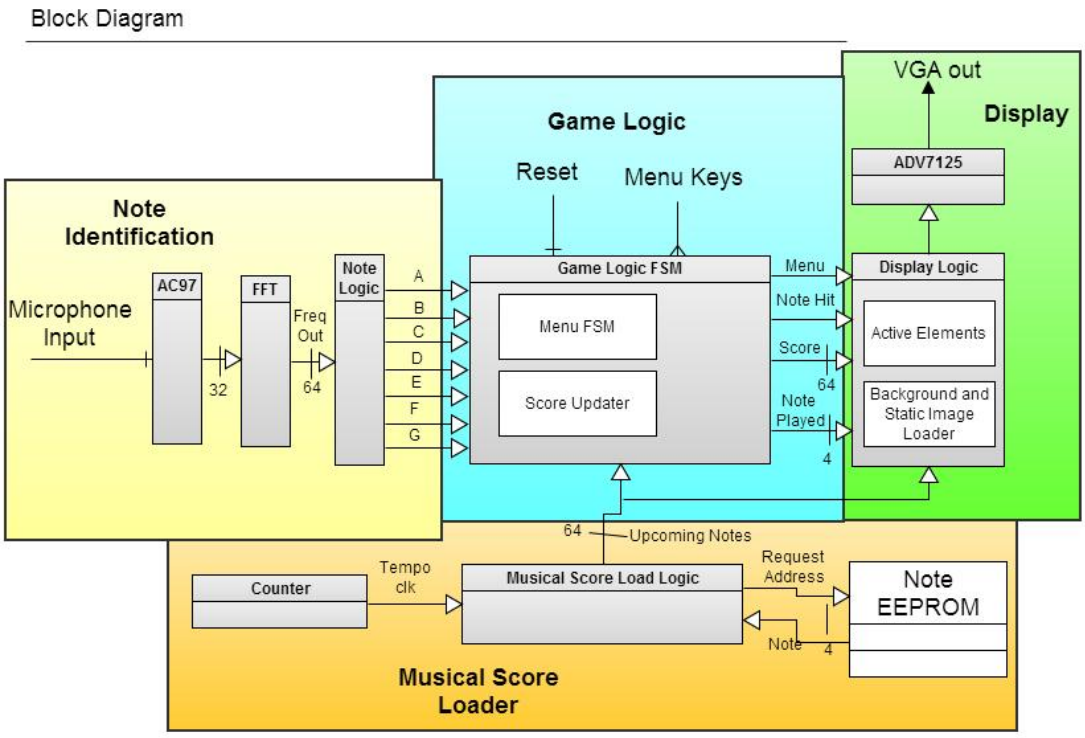

FIGURE 1: HIGH-LEVEL DESIGN / BLOCK DIAGRAM

2.1 Note Identification (Andrés)

The note identification module directly accepts and translates user audio input into usable digitized signals.

Input from the microphone is converted into a digital signal by the labkit's Intel AC97 chip. The AC97 converts the analog audio wave into digital samples by oversampling the analog input, and passes these samples to a Fast Fourier Transform (FFT) module for analysis.

The FFT module will convert these samples to the frequency domain. The frequencies of the audio input can be pinpointed so the pitch of the current note can then be identified. Because of timing constraints, this module will have to be a continuous pipelined FFT module, as we cannot afford to wait for value "bursts" between notes. As it processes the input, the FFT module will output magnitude values for a large frequency range and pass them to the note detection module.

The note detection module will have a lookup table (LUT) filled with threshold magnitude values for all the recognizable notes (A through G). It will compare each of these seven values with the incoming values from the Fast Fourier Transform and determine if a recognized note is currently being played. The output will be seven single bit wires that will show a valid one if that note is being played and a valid zero if that note is not being played. Having multiple wires allows the game to theoretically recognize multiple notes at a time, although testing will be done with monophonic instruments.

The note detection stores magnitudes for seven notes so it will require a single-port 16x8 LUT. The magnitudes will be tuned by hand after analyzing the FFT's output through the logic analyzer and spectrum visualizer demo.

2.2 Game Logic (Andrés)

The game logic will be a finite state machine comprised of two majors parts, a menu finite state machine (FSM) and a note finite state machine. These will interact by a simple enable signal which the menu state machine will apply to the note state machine once a song has begun.

The menu FSM handles the menu buttons, selecting a song and entering the game. It will take in debounced button inputs from the labkit to navigate through menus, and once the user starts a song it will reset the other modules to transfer into the game state.

The note FSM takes a list of current notes from the musical score, and determines from the note identification module's output if the correct note was played. If so, the score will be updated accordingly. It will also output a one-bit value to the display logic, indicating whether or not the correct note is being played.

2.3 Musical Score Loader (Paul)

The musical score module reads song information from Flash memory (EEPROM) and gives upcoming note information to the game logic and video output.

2.3.1 Song Encoding

Songs will be hardcoded into Flash memory. The songs will have two components: a 25-bit tempo followed by a series of 4-bit notes. To make playback tenable, the atomic note unit will be an eighth note, and the range of pitches will be restricted to A-G in whole step intervals.

The tempo will be a 25-bit integer such that the time it takes for the system clock to count up to it will be the duration of one eighth note.

Each note will be specified as a hexadecimal value; although seven notes only require three bits to encode, four bits will be used to encode each note to ensure that accidentals can be specified for future features. Each hexadecimal note value indicates the pitch that will be played for the next eighth-note interval. To create longer duration notes, the pitch can be repeated several times. A zero-value note will be considered a rest.

2.3.2 Note Loading

Once a song is selected, the musical score module will output the next sixteen notes in the song to the game logic and display. This will be done by reading 64 bits from the EEPROM and advancing by 4 bits, one eighth note, on each beat. The beat will be set by the tempo value of the chosen song.

The main complexity of the musical score module involves the Flash memory. A typical song might contain 30-50 notes of varying length, which translates to between 0.2-0.5Kbit per song. Given that the labkit has 128Mbit of Flash memory, there are no concerns about available space.

2.4 Video Display (Paul)

The video display module is responsible for taking game and note data and outputting a VGA video signal. There are several video elements to display: the incoming note data, the current score and the current pitch. Because of the quantity of information on the screen the VGA output will be set to a resolution of 1024x768; the system clock frequency will be set to 65MHz to accommodate this resolution.

The VGA signal output will be handled by the labkit's onboard ADV7125 Triple 8-bit Video DAC. The XVGA module used in Lab 3 will be reused to handle interactions with the ADV7125.

The musical score logic will provide this module with a sequence of the next sixteen notes. These notes will be rendered on the screen horizontally in time and vertically in pitch. Each note will be created with the rectangular blob module from Lab 3. There will be certain y-axis values for each pitch, and each note will initially occupy $1/15^{th}$ the screen width arranged horizontally. The notes will move horizontally leftward at a rate that after one tempo interval they will have moved across $1/15^{th}$ of the screen. As the 1^{st} note begins to move off the screen, the 16^{th} note will start to move in.

The user is supposed to play the pitch of the 1^{st} note (leftmost on the screen) at all times. The game logic module will supply a bit indicated whether or not the user is playing the correct note. Based on whether they are or not, the color of the note will change: blue indicating success, red indicating a missed note.

The display will also feature text at the top right of the screen using the cstringdisp module provided. The text will be the decimal value of the user's current score and the note that the game thinks the user is currently playing.

When the game is in the menu state, the cstringdisp module will be relied upon heavily to create a text-based menu to select a song. An alpha-blended rectangle will be superimposed on the currently selected menu button.

3 Testing

The audio processing will be tested through both simulation testbenches and manual tests. The FFT and note identification will be tested together in Modalism by generating a known frequency signal and verifying the note identification module's output. However real-world signals are significantly more noisy, so it will be necessary to compile and run the module live. The tester can then play known notes while the note identifier's output is shown on the hex-display.

Both the game logic and musical score modules can be tested (separately) in ModelSim since they concern exclusively digital inputs and outputs. By looking at the modules' outputs after several sequences of inputs it will be possible to verify correctness.

The video processing is not well suited for simulation because of its vast quantity of data. It will therefore be tested by inspection.