SNGH's Not Guitar Hero

Rhys Hiltner Ruth Shewmon

November 2, 2007

Abstract

Guitar Hero and Dance Dance Revolution demonstrate how computer games can make real skills such as playing the guitar or dancing fun to learn and practice. Unfortunately, the skills that these games teach are too simplified to translate well into their real world analogs. Guitar Hero uses a small abstract guitar with five fret buttons and a strum paddle to simplify note detection to a purely digital task. The goal of SNGH is to use sophisticated audio processing to create a guitar-playing game that is both realistic and enjoyable. Therefore, the core functionality of the game is the ability to detect chords being played on a standard electric guitar in real time and compare them to chords displayed on the screen. Users can learn the guitar chords for real songs, or work through preprogrammed lessons that cover a variety of common chords. The game also provides quantifiable measures of progress, so players can watch as well as hear themselves improve.

Contents

List of Figures

1 Overview

SNGH is designed to make learning the guitar fun and easy. To create an immersive user experience, three external devices will be required: an electric guitar, a pair of headphones, and a screen. While the headphones and screen are available in the 6.111 laboratory, we will provide our own electric guitar. Audio input from the guitar is fed through an AC97 audio codec and a series of FFTs, then processed so that information about chords can be displayed on the video screen. A rough block diagram is shown in figure 1.

1.1 Modes of Operation

FFT Debug

So that we can debug the signal processing and note detection modules, the FFT Debug mode displays a combination of raw FFT data and the results from the Frequency-Domain audio processing on the LCD screen.

Training Mode

In training mode, SNGH shows the user a chord in standard notation, prompting the user to play it. When the chord is played or the time limit expires, a new chord is displayed. The time limit can be set to match with the player's skill, or disabled entirely. If the played chord isn't the one requested, SNGH will tell the user what chord they actually played.

The list of chords presented to the user can be generated randomly from several difficulty sets or match up with the guitar tabs for some preprogrammed songs. A menu screen will allow users to select which sequence of chords they'd like to learn.

Optional Modes

If time permits, a Guitar Tuning mode will be created where SNGH displays a small portion of the FFT and a half-step window to tune it inside of. An in-tune guitar is an important part of learning how to play and would improve SNGH's accuracy in detecting chords. Additionally, a Song mode may be created that plays prerecorded music from a flash ROM and allows users to play along using the chords they've learned.

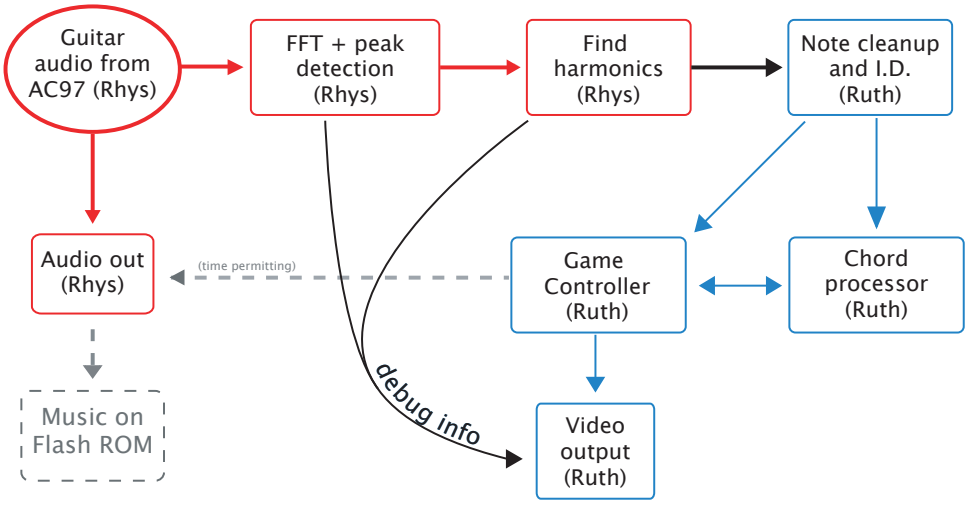

Figure 1: Block diagram

2 Description of Modules

A high level block diagram is shown in figure 1, to show how the modules fit together.

2.1 FFT and Peak Detection

This module's job is to convert the sound coming from the guitar into frequency space and search for peaks in the signal.

Inputs

The input to this module comes directly from the AC97 audio codec. The module receives a 36 bit wide stereo audio sample at 48kHz.

Outputs

The peak detection module finds the first 64 peaks and sends them to the *Harmonics Processor*. It sends the 6-bit index of the peak number, the 14-bit location of the peak, and 8 bits of the peak's intensity 64 times per FFT frame, with FFT frames happening at every 1024 audio samples, so peaks are sent, on average, at 3kHz.

Description

FFT: Audio comes into the module as two 18-bit signals at 48kHz. The two signals are averaged and cut down to 8 bits wide. A buffer of the last 5120 samples is stored so that several overlapping FFTs can be run. The most recent 4096 samples are read from the buffer and scaled to a 4096 entry Hanning window to avoid introducing high-frequency artifacts. The 4096 sample frame is then padded with zeros to a length of 32768 and fed into a 32768-bin FFT. The upper 16384 bins, corresponding to frequencies above 24kHz, are thrown away. The remaining samples are then scaled and buffered. The sample with the greatest power is noted.

A new FFT is run every 1024 samples, so each sample is sent through 4 consecutive FFTs.

Peak Detection: The 16384 bins from the FFT are then sorted through to find local maxima. The first 64 local maxima above the noise threshold ($1/16^{th}$ of the biggest sample) are recorded to be passed along to the *Harmonics Processor*.

The peak detector also sends its output to the *Video Display* for debugging and testing.

2.2 Harmonics Processor

This module searches through the detected peaks for peaks that are integer multiples of each other. This groups fundamentals with their harmonics. The fundamentals this module finds are used later to find what chord is being played.

Inputs

The *Harmonics Processor* receives a list of the first 64 peaks and their intensities in the frequency-space representation of the audio from the *FFT and Peak Detection* module.

Outputs

This module sends a list of up to 6 of the best fundamental frequencies and their intensities to the *Timespace Filtering and Note I.D.* module.

It also sends the fundamentals and their intensities to the *Video Display* module for debugging.

Description

To find harmonics, this module attempts to fill up and reduce a matrix of fundamentals and their harmonics. The matrix is 64 entries long (for the 64 maximum peaks), 10 entries wide (for the fundamental through the 9^{th} harmonic), and each entry is 6 bits deep (to store a pointer to the list of 64 peaks).

The *Harmonics Processor* moves two pointers through the list of peaks, one for the fundamental and one for the harmonics. It checks if the harmonic pointer is close to being a prime multiple of the fundamental, and if so makes a note in the appropriate element of the harmonic matrix.

The module then searches through the harmonic matrix to fill in the composite multiples of the fundamentals. Now that the matrix is complete, the module finds which frequencies aren't a harmonic of any other. These are the fundamental frequencies that get passes along to the *Timespace Filtering and Note I.D.* module.

2.3 Timespace Filtering and Note I.D.

This module takes a set of fundamental frequencies from the *Harmonics Processor*. The output of the *Timespace Filtering and Note I.D.* is a 12-bit vector, where the bits correspond to the 12 musical notes in each octave. Since notes separated by an octave are equivalent for our purposes, notes are shifted so that they all reside within the same octave.

A fundamental must exist at roughly the same frequency for several clock samples in a row before it can be considered a note. Once a fundamental passes this test, the module decides what musical note corresponds to the detected frequency and sets the corresponding bit in the output to 1. If a fundamental disappears for several consecutive samples, then the corresponding bit in the output is reset to 0.

The output, which is essentially a list of all notes currently active, is passed to the *Chord Processor*. The timer module also outputs a ready signal to the *Chord Processor* when 3 or more notes are detected and sufficient time has passed that the user is likely to have finished strumming a chord.

2.4 Chord Processor

The Chord Processor module takes a 12-bit number from the *Timespace Filtering and Note I.D.* module. Each bit corresponds to one of the 12 notes in a musical octave. This number is cyclically bitshifted and compared to a lookup table of chords stored in BRAM. If there is a match for any such permutation of the chord, then the *Chord Processor* alerts the *Game Controller*.

2.5 Game Controller

This module controls gameplay and changes the mode of the game between training mode, tuning mode, and debug mode(s). It requests chords from the user and is aware of what chord the user is currently playing. It also handles chord timing and the different modes of operation.

Inputs

The primary input is the current chord being played, generated by the *Chord Processor*. It also gets signals from the *Timespace Filtering and Note I.D.* module regarding special single notes being played.

Outputs

This module tells the *Chord Processor* what chord to expect so it can decide close ties. It gives the *Video Display* module the current chord, the next chord, the time remaining to play the current chord, the current score, what the current chord being detected is, and if the correct chord is being played.

2.6 Video Display

The *Video Display* module can operate in one of several modes, as described in the Overview section of this report. This meta-module takes control signals from the *Game Controller*, and receives the current FFT data from the *FFT and Peak Detection* module as well as a list of detected frequency peaks from the *Harmonics Processor*. The *Video Display* module outputs R, G, and B intensity for the current pixel,

hsync, vsync, and blanking signals to the ADV7125 Triple 8-bit high-speed video DAC built into the labkit. The DAC then generates the correct signals to drive an LCD monitor.

The SNGH video interface has several modes of operation, though the vast majority of the user's time will be spent in the Training mode. Switches on the FPGA will initially be used to move between the states, a process that is mediated by the *Game Controller*. Each mode is described in the overview section of this report.

FFT Debug

Code for the FFT debug mode will be adapted from Chris Terman's lecture demonstration on FFT's but with additional features added. First, peaks in the FFT that have been detected as harmonics and fundamental notes will be highlighted in different colors from the background. Second, a scale will be drawn across the bottom of the screen that marks the location of musical notes. Finally, the display can be frozen at a single frame to observe a slice of the FFT.

Training Mode

The video display for training mode will be generated by several parallel processes, each of which writes to its own section of the screen. First, a miniaturized version of the *FFT Debug* module will displayed across the top of the screen. Second, a fingering for the current chord will be rendered in a standard guitar tab notation, which can be constructed from lines and circles. The name of the current chord will be written above this fingering. The next chord to be played will be displayed in miniature on the opposite corner of the screen. A third of the screen will be dedicated to providing the user with feedback, and showing a score. When the user plays the correct chord, a celebratory bitmap will be displayed. Otherwise, it will display the name and fingering for the user's incorrect chord. A fourth area of the screen will display the adjustable timer, which will be a full rectangle that empties like an hourglass.

Optional Modes

The optional tuning mode would be an extremely zoomed-in slice of the FFT Debug display, centered around the frequency the note being tuned.

3 Testing and Debugging

Our first priority will be setting up a debugging framework. The ability to detect notes and chords in real time is of crucial importance to the project, but is also difficult to debug since it processes a large amount of information. Therefore, a video output of FFT data and subsequent audio processing results will be created at the very beginning of the project development. A model of the frequency-domain signal processing has already been created and tested in MATLAB, and it works quite well. Detailed visual displays were found to be very useful in the debugging and optimization of this model.

Each module will be built and incrementally tested before joining the SNGH system, and in the process there will be a multitude of debugging outputs from each module. State information will be output to the single LEDs, logic analyzer, and the labkit's LED matrix displays.

4 Division of Labor

Rhys will handle the frequency-space processing (*FFT and peak detection* and *Harmonics Processor* modules). Ruth will develop the time-space processing (*Timespace filtering and note I.D.* and *Chord Processor* modules), the *Video Display* module (including the video debugging mode), and the *Game Controller* module.