Guitar Hero: Fast Fourier Edition

Ryan Berg, Mitchell Gu | 6.111 Project Proposal | Fall 2015

1 Overview

Guitar Hero, a musical game concept pioneered at the MIT Media Lab, brought the dream of rockstar-dom to kids and adults worldwide by presenting popular song scores as a waterfall of combinations of five notes, scrolling intuitively in the time domain. Our Fast Fourier Edition aims to reinvent the genre by capitalizing on FPGA hardware to use an actual electric guitar's analog output as the game controller.

Instead of expecting some combination of button- presses on a traditional controller, the FF edition performs a fast fourier transform (on a 6.111 Labkit) of the analog signal from a guitar's pickups to determine what fundamental tones are being played. From the FFT results, the game logic (on a Nexys 4) will award the player points depending on their pitch and timing accuracy compared to what was supposed to be played. A graphical display of the notes to be played, points awarded, and engaging supporting graphics will all be displayed on a VGA monitor. These features will allow users to play their favorite songs and learn some actual guitar along the way.

Stretch goals of this project include: actual Guitar Hero-esque graphics, sounds, and other immersion effects, detection and scoring of entire chords (6 notes played simultaneously), detection of hold notes, "Rockstar Mode" by tilting the guitar, generation/amplification of guitar effects by the labkit, with audio out.

2 Design

2.1 High-level Overview

Our project design calls for both a 6.111 Labkit and a Nexys 4 FPGA board, seen in Figure 1, to allow for specialization of tasks and more efficient development. The Labkit is used solely for audio processing, the results of which are input to the Nexys 4. The Nexys 4 will then implement the game logic and generate video and audio output for the player.

This approach will allow us to play to the strengths of both FPGAs by utilizing the integrated AC97 on the Labkit and capitalizing on the greater memory and video processing capabilities of the Nexys 4.

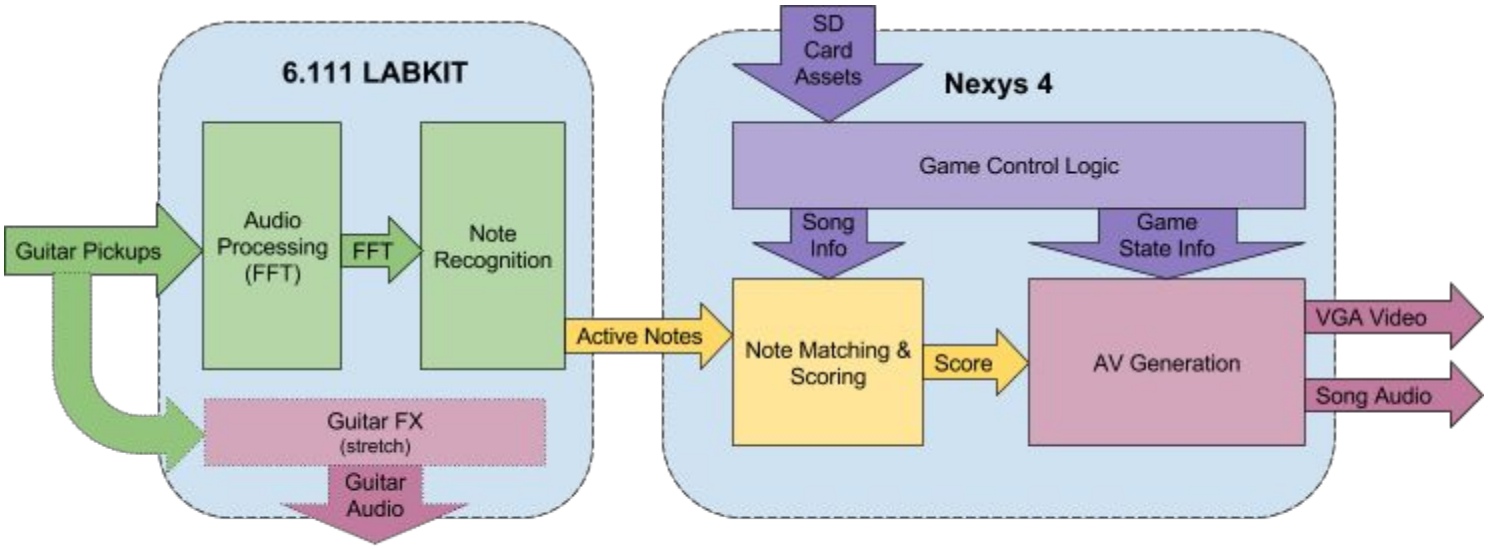

Figure 1: The high-level project design (with a stretch goal added)

Furthermore, having two platforms will allow us to develop and test each platform concurrently, which eases the workload in the tighter time constraints of this semester. Since the FFT module will be large and time-consuming to synthesize, isolating it on the Labkit from the majority of our game logic on the Nexys 4 will be a big time saver.

2.2 Audio processing on the Labkit

The Labkit features an on-board AC97 with 3.5mm stereo jacks for microphone input and speaker output. Since microphones and guitar pickups are similar passive audio capture devices, we plan to feed the guitar's pickup signal directly into the microphone input of the labkit. From there, the AC97 can amplify the weak pickup signal, digitize it with its 18-bit ADC, and output it for the Virtex 2 to process.

Using the FFT verilog module shown in class as a foundation, the FPGA will perform an FFT with 4Hz bin width, enough to discern between every note in the range of a typical guitar. Using the FFT output, the Labkit will then detect the dominant note(s) being played, serially encode that data, and send it to the Nexys 4. By building the note recognition functionality into the labkit, the serial data transmission from the Nexys 4 to the Labkit does not need to be particularly complex; it needs only to transmit the current notes being played.

2.3 Game logic and output on the Nexys 4

The Nexys 4 has much more memory available than the labkit, as well as SD card integration, which makes it the proper candidate for the non-audio portion of the system. Having an SD card and more memory means that the Nexys 4 will be able to load more song information and graphics assets than the Labkit.

The matching and scoring block of the Nexys 4 receives a serial-encoded list of the active notes currently being played from the Labkit and compares them to what notes should be playing currently, as given by the metadata of the current song loaded from the SD card. It scores the player's performance appropriately and forwards the score and matched note data to the video output.

The AV Generation block takes information about the current score, as well as the state of the game (playing, paused, song over). It combines graphical assets with this information and the song audio to produce a VGA output and an accompanying audio signal.

The Game Control Logic block is the backbone of the Nexys 4. It loads assets from the SD card, maintains the state of the game, and properly handles the timing of the scoring module.

2.4 Motivation

The primary motivation for this project is to create a platform with the entertainment value of Guitar Hero, but with the realistic educational experience of playing an actual guitar. As such, accurate note detection and scoring are critical for an entertaining frustration-free experience. Feedback provided to the player is important for proper educational value, and so an intuitive graphical interface is also very important.

3 Implementation

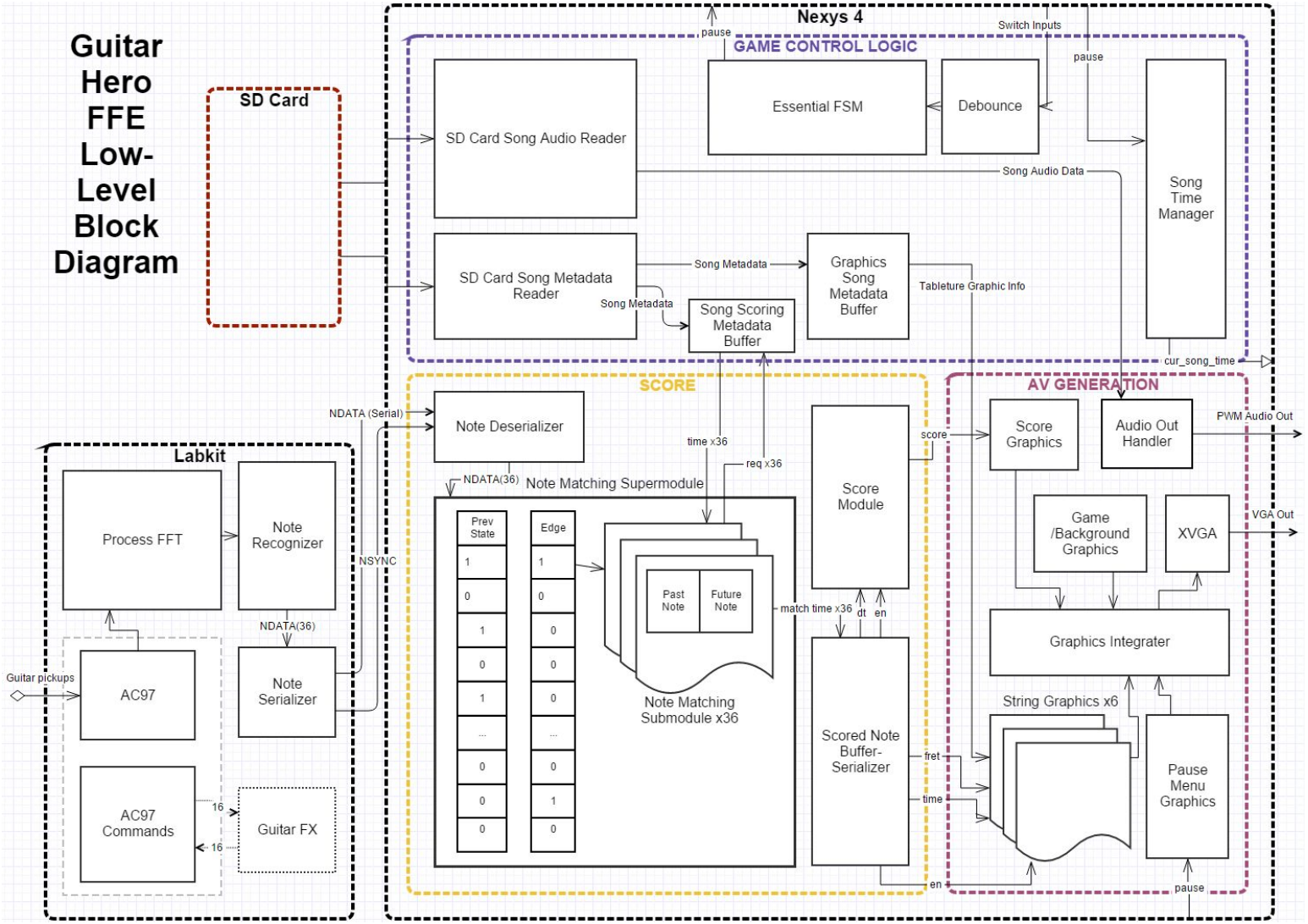

Figure 2: A low-level block diagram of the system

3.1 The guitar audio block (Labkit)

The guitar audio block will encapsulate the AC97 module and the AC97_commands module, adapted from the lab 5 on audio filtering. The block will handle the translation of the AC97 chip's microphone ADC output into a 16 bit digital bus (from_ac97_data) and also the translation of a 16 bit digital input (to_ac97_data) to the signals the AC97 feeds to its DAC and speaker output.

3.2 The process FFT block (Labkit)

The process FFT block will take input from the 16 bit data from_ac97_data line and perform the FFT on the signal using the IP core gen, as demonstrated in the fft example verilog. From the fft results, the module will output haddr, the address of the current bucket, and hdata, the magnitude of the spectrum in that bucket.

3.3 The note recognizer and serializer (Labkit)

The note recognition block will take in the histogram data from the process fft block and will identify the dominant frequencies present in the spectrum. It will also map these frequencies into notes and clip them to ensure they are in the guitar's range. Because the FFT won't be able to discern between the same note played in different positions on the guitar, it only needs to identify which notes in the guitar's 36 note range are currently active. Therefore the output of the note recognizer will be a 36 bit bus where each bit corresponds to a note in the guitar's range and represents if that note is currently being played.

To prevent rapid bouncing between active and inactive notes due to noise in the spectrum, we plan to implement some hysteresis on the active-inactive threshold. This will clean up the signal for the Nexys 4 and prevent one played note from being registered as multiple.

To simplify communication of the active notes to the Nexys 4, a serialization module will be used. The module will take the 36 bit note data bus and serializes it into a two wire interface: NSYNC and NDATA. NSYNC pulses high to signify the start of a packet, then the 36 bits of active note data are shifted out through NDATA. With serialization, only two wires are necessary to connect the labkit and Nexys 4.

3.4 The note matching block (Nexys 4)

The main objective of this block is to identify from the incoming NDATA when new notes have started to be played and to match the note's start with a corresponding note in the current song's metadata.

The first module in this block will be a note deserializer, which will first synchronize and then deserialize the data from the labkit into a 36 bit bus of active notes.

This bus will then be inputted into a note matching supermodule which will record which notes had 0 to 1 transitions (just became active) by comparing the input to the last timestep's input. The supermodule will also have 36 parameterized note matching submodules, each of which corresponds to one pitch in the guitar's range.

The job of each submodule will be to, if a 0 to 1 transition for its note is detected, match that played note to the nearest actual note (within an allowable time span) in the current song. It will do this by storing two critical state variables: the time of the closest actual note with the same pitch that has already passed, and the time of the closest note with the same pitch in the future. When a 0 to 1 transition is detected from the guitar input, the submodule will determine whether the past or future nearest note is closer, thus "matching" the played note with a song note. If the time difference with this match is outside acceptable human error, the match is denied. When a match is granted, the value of the state variable (either past or future) that was matched will be wiped to time 0, indicating it was played. The output of each submodule will be the song time of the matched actual note.

The matching submodules will also have to update their past and future note times using a separate module that serves as a table of the entire song's note time metadata. When the current time passes the future note's time or if the future note is played, the submodule will signal to the metadata table to send it the next future note and shift it into its future and past state variables. If, upon shifting, the past

state variable had a nonzero time in it, the submodule knows the player missed playing that note and will communicate that to the scoring block.

All the outputs of the matching submodules will be fed into a buffer/serialization consolidation module that ensures that simultaneous outputs from all 36 matching submodules are fed one by one into the subsequent score module and string graphics modules (see 3.4). Since the outputs of the matching submodules will be the song times that the played notes were supposed to be played at, the consolidation module will calculate the player delay by subtracting away the current song time, then sum across all played notes to arrive at a total player delay it can send to the scoring module (see 3.5). The module will also serialize the matched notes, translate them into fret positions for each guitar string, and output the fret and song time to each of 6 string graphics modules. This is needed because when the player plays a note, it should be popped off the video display.

3.5 The scoring block (Nexys 4)

Whenever the consolidation module reports a total delay for new played notes, the scoring block will assign the player a score increment or decrement based on the delay and increment the total score with it. It will also communicate to the score graphics module (see 3.5) what the increment and total score are so they can be displayed on the screen. Also, when the consolidation module indicates that the player missed a note, the scoring block will decrement the player's score and break any streak the player may have been on.

3.5 The graphics block (Nexys 4)

The graphics block will include six string graphics modules for generating pixels for each of six strings, a game graphics module for background game graphics, a score graphics module for rendering the score, and a paused game menu module to display a menu when the game is paused.

Each string graphics module will be responsible for rendering the upcoming and slightly passed notes on one guitar string, scrolling horizontally in the time domain. The screen display will resemble that of common guitar tablature, where six rows correspond to six strings, and notes have fret numbers attached. The horizontal axis represents time, so as the song progresses, notes with fret numbers will scroll from right to left on each string's row.

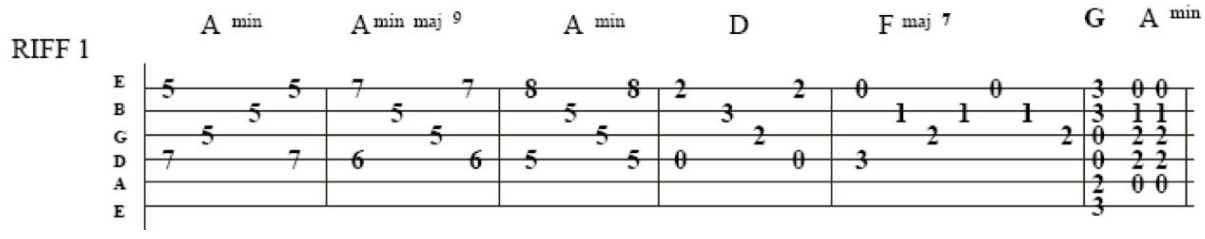

Figure 3: Sample tablature for Led Zeppelin's "Stairway to Heaven"

The string graphics module will have an array with enough slots to store a large number of current notes with their fret number and song time. The array will be updated with the help of another song metadata table that will provide new notes with their fret position and song time when they should

enter the screen. From this array, the module can produce output pixels using a string image as the background and stored sprites for fret numbers, shifted to the correct position in time.

The score graphics module will take score increments and also the total score from the scoring module and produce a pixel output using a set of number sprites for enhanced appearance. An overall game graphics module will provide a background of sorts such as an image of a crowd to set the atmosphere, Finally, a pause menu module will provide pixel output for an overlay menu in the case where the player pauses the game.

3.6 Tying it together: The game control block (Nexys 4)

The game control block includes a basic game FSM that will allow the user to pause the game, adjust settings, or change the song. It also includes the interface with the SD card for loading new songs into FPGA memory, including the two metadata tables that are used by the matching & scoring block and the graphics block. Finally, it includes the song time manager which generates the song time signal that nearly all modules use to know where in the song the player is.

4 Timeline

Figure 4 is a Gantt chart representing the current schedule for the project. Here are the stages in that Gantt chart:

1. The Scoring block involves taking in a list of active notes and a list of notes that should be played, and calculates an appropriate score.
2. The FFT/Note recognition block is the Labkit, which takes a pickup signal from the guitar and determines what notes are being played by the guitar.
3. Functional video is the implementation and testing of very fundamental game video, so that basic playtesting can occur.
4. Serialization and deserialization involves linking the labkit and the Nexys 4 to transmit active note data serially over only two wires.
5. The Game Control Logic is a big step in integrating the blocks, because it handles the loading of assets, and controls the flow of game information to the other blocks.
6. Nice video involves the creation of more aesthetically pleasing graphic assets than the basic debug video.
7. Integration and testing will take place once all the individual modules and blocks have been implemented and tested. Here is where basic playtesting occurs, and the player interface is critiqued.
8. An actual song with appropriate metadata is needed for demo purposes, and to test the capabilities of the system.
9. Here is time allotted for iteration, and addition of stretch goals if possible.
10. This is a block of time for comprehensive testing, to ensure that the system is working properly for demo.
11. Finally, the project will be demonstrated and checked off. The final paper also needs to be done this week.

	11/2/2015	11/9/2015	11/16/2015	11/23/2015	11/30/2015	12/7/2015
1) Scoring block						
2) FFT/Note Recognition						
3) Basic AV/Graphics						
4) Serialization/Deserialization						
5) Game Control Logic						
6) Nice Video (fonts, assets, etc..)						
7) Integration, testing						
8) Song for use and testing						
9) Stretch goals						
10) Buffer Time						
11) Project demo, final checkoff						

Figure 4: A Gantt chart for the project

5 Testing

General testing of modules will happen throughout the project as modules are developed.

To test the note recognition on the labkit, it will likely be easiest to start with the FFT demo verilog from class and have the note recognizer display the notes it recognizes on a VGA monitor in addition to the frequency spectrum. With this method we can visually debug any issues with the recognition by inspecting the spectrum on a real audio signal from a guitar.

The modules that will likely involve the most rigorous testing using test bench simulation are the note matching and scoring modules, as they are detailed and hard to debug in real time. The testing process for these modules will likely involve populating the song metadata with test data and simulating different cases of guitar input notes to ensure that those notes are matched to the correct song notes.

Finally, the video modules can be tested individually both using test benches and by inspecting the output on a VGA monitor.

6 Resources

Several audio connectors/cables are required for this project. We will need 2x ¼ female to 3.5mm male audio adapters (~$3 each).

7 Conclusion

Guitar Hero: Fast Fourier Edition is a technically complex, yet achievable FPGA game concept that has potential to change how people learn to play guitar. With an actual guitar as the controller and an interface similar to that of actual guitar tablature, a player can directly translate gameplay to real-life skills. By building this project, we will learn a lot about how fourier transforms, digital memory, and sprite-based graphics work. The two of us will be connecting what we have learned in class to a hobby that we both enjoy. We hope that the results of this project can inspire others to try it out and enjoy learning guitar.