Digital Debussy
A Hardware-Based Music Composition Tool

JORDAN ADDISON
ERIN IBARRA
Masssachusetts Institute of Technology
October 30, 2014

Introduction

Music composition programs are powerful tools for both beginning musicians and experienced professionals; however, most of these programs are expensive and may cost hundreds of dollars because of the immense number of features included. While this may be a worthwhile investment for serious musicians, most amateur musicians do not need the powerful features and tools of professional music composition software. Our goal is to develop a cost-effective, hardware alternative that is simple and easy to use for musicians of all ages and skill levels. This project can be easily expanded to include as many features as we are capable of implementing in the time allotted to complete the project.

The focus of our project is the music writing and playback system. A user will be input a melody using a graphical representation of sheet music and tool bar to adjust a variety options, such as note duration. Once the user is satisfied, he or she will be able to playback the melody through the labkit speakers. We intend to utilize existing sound files that represent various instruments so the user can customize his or her music further. Additionally, we will implement a control system for the tempo of the music so a user can speed up or slow down his or her music. Ideally, a user will be able to compose a melody in any key or time signature.

If time allows, we have developed a list of additional features we would like to implement in our system. We would like to create a transposition feature, so that a user can compose a melody in any key signature, but can

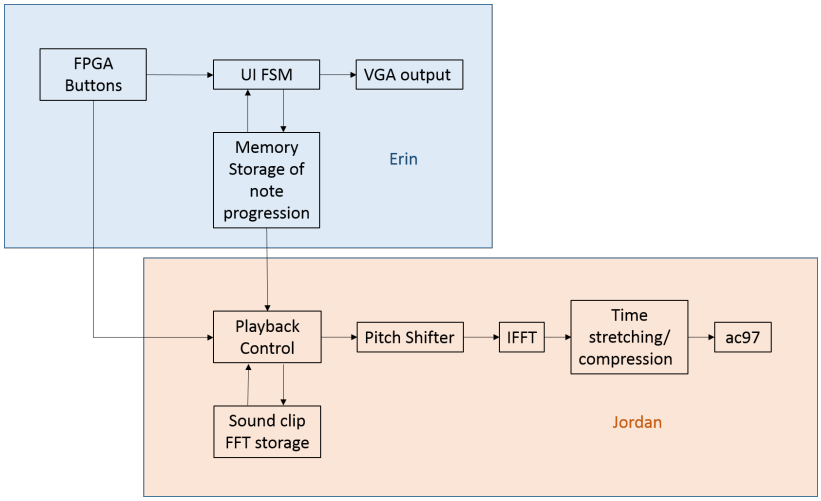

Figure 1: *A high level block diagram of the system architecture*

transpose it up or down to a new key signature to suit his or her desires. We would also like to design a system that automatically harmonizes a given melody based on classical music theory constructs. Finally, we could also find a way to export the piece of music as an image file so a user can print his or her music.

In terms of the actual implementation, we see two divisions. Erin will be developing the user interface, which includes how the user will select note duration, type, and other features. Jordan will develop the sound playback, tempo control, and transposition.

User Interface

The user interface should be a simple, intuitive tool for any musician to utilize and understand. It will respond to user inputs from the buttons on the labkit, creating notes that will be stored in memory and displayed on screen. A user will be able to adjust certain parameters related to a specific note: pitch, duration, accidental adjustments, octave, and quality of attack. These values will be encoded in a 16-bit data structure, with different portions of the structure encoding different aspects of the note. This data structure will be the primary form of communication between modules of the system.

Within the user interface module, there will be two major divisions: a module devoted to the user's interactions with the buttons and a module for graphics. Both the user interaction module and graphics module will interact with memory as well as interacting with one another.

The video display will have two major components: the displayed sheet music and the interactive tool bar. The sheet music will display multiple staves in pairs of one bass clef and one treble clef staff. For simplicity, even if only one staff is used, the system will always display a bass clef and a treble clef staff. The tool bar itself will display each of the note attributes and the user will be able to scroll through various options using the labkit buttons. A user will be able to interact only with the tool bar or with the sheet music at any given moment. A button will be designated so that the user can switch between the two modes.

In order to maximize flexibility, each piece of music must have several inherent attributes, which include its key and time signatures. The key signature attribute will be especially important for playback, but will also be necessary for displaying the correct type of note. The time signature attribute will affect how a user can place notes of varying duration in his or her piece of music.

The tool bar will be implemented as a series of layered state machines. Because the system will utilize buttons for user input, each button press will signal a state transition. At the highest level, each selectable note attribute will be a state. Within each of the high level states, there will be another state machine specific to that note attribute; button presses will again signal state transitions within these inner states.

The graphics module will most likely utilize sprites for each of the notes and will always display the staves and tool bar. As the user interacts with the tool bar and the sheet music, the graphics module will highlight the specific portion of the tool bar or measure of the sheet music that the user has chosen.

Sound Playback

At the heart of the sound playback module is the goal of producing a nice sounding audio output that accurately represents the musical decisions made at the user end. The first inherent challenge of this module is being able to playback anything at all! In order to keep the design as simple

as possible, our sounds will not be synthesized from scratch in the FPGA. Rather, we plan to take known samples of digital synth voices (i.e. keyboard, strings, trumpet, etc.) and process them to fit the desired pitch, attack, and duration selected by the user.

One of the largest challenges of this approach will be finding a suitable technique for altering the pitch and duration of a note independently. In the time domain, our most instinctive view of signals, these two properties are inherently linked. If you play a clip faster (like fast forwarding a cassette tape or record) the pitch becomes noticeably higher (imagine a chipmunk type effect on a voice). However, we can decouple the pitch and duration by using a frequency domain approach (fast Fourier Transform or FFT) to control pitch and using a time domain approach to control duration. This kind of approach is a very popular DSP technique known as a phase vocoder.

The basic functionality of a phase vocoder is not too difficult to understand. First, we window our sample clip into 'frames' - time delimited segments of sound - of equal length. It is important for these frames to overlap to some degree so that our output sample sounds smooth, it can also help to use a hamming window rather than a sharp rectangular one to help with this issue.

The way we can choose the duration (or speed) and pitch of our playback sound is by carefully deciding how to reconstruct the frames. Altering the speed of the note can be done simply by changing the amount of overlap between frames - more overlap corresponds to a shorter note, less overlap to a longer one - this way, we can change how long it takes for a sound to finish playing without changing the FFT information of the sound.

Conversely, a phase vocoder allows us to change the pitch of a note without having to change its duration. This gives us a way to know the degree to which each frequency is present in the sample. Shifting the pitch of the sample is as simple as multiplying these frequencies by some scale factor (while keeping their relative amplitudes in the signal composition). For example, if our sample were a pure sine wave of a 440Hz "A" it's FFT would be a delta function at 440Hz. If our user wanted to instead play the same note one octave higher, we would just produce a sine wave at 880Hz - effectively scaling the frequency its FFT by two. For the same note one octave lower, we would scale the frequencies by one half. This technique can be extrapolated to apply to more complex sound samples, like those coming from real musical instruments, whose FFTs contain a variety of frequencies

with different amplitudes that give instruments unique tonal qualities that we would like to preserve.

Implementation of the phase encoder will be a bit tricky, and involves many trade offs in speed, memory size, and sound quality. The plan for now is to compute the short time Fourier Transform (STFT) of the sound clips ahead of time in MATLAB and store them in the FPGA. According to the parameters selected, some processing will be done to the STFT, followed by an inverse transform to return the signal to the time domain. Even with this basic outline of operation, more design considerations will be worked out in the coming weeks. For example, what should be the frame size of the windows, the sampling rate of the sound clips, the depth of the sample (8 - 16b), and the resolution of the filter.

Though note duration and pitch are the backbone of the playback module, there are many other features that can really elevate the quality of digitally produced music. The end goal for any digitally produced music is to sound as realistic as possible. Some important qualities of a sound that can make it sound more are less real are the attack, decay, sustain, and release. The Attack is the first portion of the sound, from when it starts to build up to the point it reaches its peak. Decay is the part of the envelope where the signal drops from its peak value to an almost constant value which would be held during Sustain. The Release is the final part where the sound fades out. As an example when you press a key on the piano, the sound will hit in (Attack), then fall fast (Decay) to a rather constant tone (Sustain) and when you take your finger off, the sound fades away (Release). The most straightforward way to control the parameters is applying an envelope shaping filter to the output sample before it's played. This envelope filter could also be shaped by the user in some ways. A reachable goal is to allow control for accented notes, meaning a more pronounced attack or staccato notes - ones that end abruptly rather than gently tapering.

Testing

Testing the user interface and graphics module will be reasonably simple. By creating test benches that simulate user button presses, it will be clear if the module is performing to specifications. The graphics module can be tested in a similar way and by simply viewing its output on the computer monitor. All of these features can be tested without the functionality of the

playback module.

Similarly, the playback module can be tested without the user interface or graphics. First, it will be written and debugged in MATLAB to ensure proper functionality of the algorithms and remove any visible bugs that can be found with the more intuitive debugging system. Afterwards, the code will be transcribed to Verilog and tested using simulation. We can then create a simple sequence of notes and preload them into memory, which the playback system will read and output. We can also use this method to test the functionality of the memory itself.

1 Project Goals

1. (Minimum):single note melodies, any pitch/time signature/note duration with some constraints, with playback
2. Multiple notes (chords) all with the same temporal duration. i.e one voice may not have quarter notes while another has eighth notes.
3. Mixtures of temporal duration across voices
4. Additional features: Notes tied across measures, accented and staccato notes, dynamics (crescendo and decrescendo)
5. Multiple pages of music that can be scrolled through