Single Board Computer System Undergraduate Education: Design and Fabrication of a mixed signal automated Guitar tuning system

Charles Duvall Asst. Professor SPSU, Ross Pettingill GTRI

Abstract – Undergraduate students at Southern Polytechnic State University experience the system development cycle from design to fabrication in a semester length course based on single board computer system projects. System guidelines are provided and students define projects in areas of interest. This document describes an interesting and creative project designed and fabricated for Southern Polytechnic State University's ECET Digital 3 class by Ross Pettingill, a recent graduate, completed as a senior student and will be discussed as an example of how computer systems are taught to our students. The project was a combination of a microcontroller based system and analog signal processing with the overall task of determining a guitar's frequency and actually turning the tuning keys of the guitar with a stepper motor. The menu driven user interface, the operating instructions, hardware, and software descriptions are described in the paper. Programming code, screen flow layout, parts list, memory map, and schematic of the design will be presented and discussed for educational benefit.

Keywords: Micro-controller, mixed signal, single board computer

INTRODUCTION

The task of the course was to design and build a student proposed project which must follow certain guidelines. Some of the guidelines include using a microcontroller, constructing the system similar to a larger computer's architecture, utilizing an LCD display for user information and menu display, making use of a sixteen button keypad, communicating with RAM external to the microcontroller, and implementing speed or position control with a motor. The operation of the system is to be relatively user friendly, the hardware used is relatively inexpensive, and the software is designed to maximize precision.

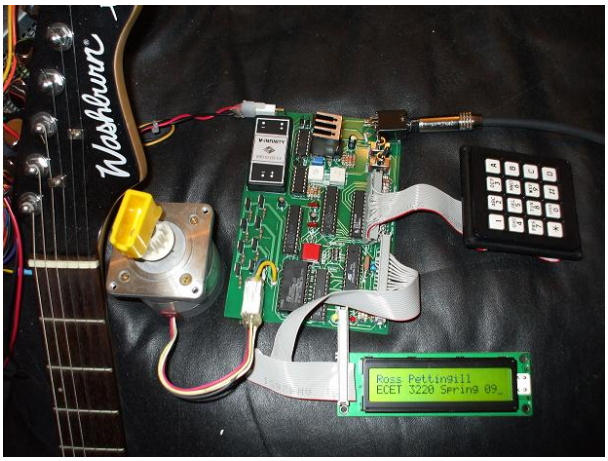

Figure 1. Picture of Completed Project

The development of the system is structured to follow a Systems Engineering Process typically used in industry. Students are initially charged with a thorough requirements plan and decomposition of their project into functional blocks, subsystem identification and associated hardware or software interfaces, physical modeling and development of a test plan. Much emphasis is placed on the development of a clear user interface document and a flow chart to ensure the screen flows create an appropriate and cohesive user interface. Students are required to identify the parts to be used and layout the board(s) prior to any assembly or testing.

OPERATION INSTRUCTIONS

This section covers the system operation instructions as well as use of memory to implement an advanced feature used by many guitarists. Students are required to have a useable and clear human machine interface that does not require detailed system knowledge for operation.

Tuning a Standard Guitar

Standard EADGBE tuning can be achieved by plugging a guitar cord into the guitar cord input jack, navigate through the display menus by using the keypad to select "Tune Guitar" and "Standard". Next, cycle through which numbered string on the guitar you would like to tune by pressing A or B on the keypad (by convention, string 1 is the thinnest string closest to the floor and string 6 is the thickest string closest to the ceiling). Next, place the stepper motor on the tuning head of the applicable string. Finally, strike the individual string and wait as the system continuously measures and tunes your guitar. When the signal dies or the system has achieved the desired tune, the information screen will appear showing what the desired frequency and the measured frequency. If the two values are not very close, consider striking the string again and letting the system continue to tune the guitar.

Creating a Custom Tuning

If there is an alternate tuning preferred other than standard (standard being EADGBE), system memory is used for unique guitar tunings. Users can either enter the frequencies for each string or pluck a string and perform a frequency measurement (which stores the result in memory for future re-tuning). When done, the system will ask you to name your tuning using the keypad, which is similar to typing a text message in a cell phone. Finally, choose a save state to store your tuning into system memory and choose overwrite.

Tuning with a Custom Tuning

After going through the creation steps, the system will retain the custom tuning frequencies in an EEPROM even if the system is not powered. To tune a guitar to the custom tuning, navigate the display menu using the keypad to "Tune Guitar" and "Custom." Then tune the guitar following the LCD display instructions.

HARDWARE DESCRIPTION

Semester length projects allow the students to implement mixed signal systems utilizing knowledge gained in other courses. In this case, previous courses completed by the student in audio, motors, electronics and digital course sequences were essential.

Amplification and Filtering

In order to be able to accurately measure the frequency of the guitar signal in a digital system, numerous analog processing methods must be implemented. The techniques used include amplification, filtering, rectification, Schmitt triggering, and using an analog to digital converter.

Both amplification and filtering must take place due to the fact that a guitar signal's amplitude is very miniscule compared to digital ranges and the fact that guitar signals contain relatively high powered harmonics which give a guitar its sound[1]. These harmonic timings are difficult to predict and can seriously impede the system's ability to measure the frequency in the time domain without filtering. The method of amplification and filtering is a gyrator based two band equalizer which can alter the boost and cut of both the upper and lower strings (standard tuning). The reason for the two bands of filtering as opposed to a single band pass filter is the fact that the 6th string (the low E) contains high powered harmonics which are within the scope of the 1st string's (the high E) frequency. If a bandpass filter was implemented, either the low E string would still contain harmonics and would be difficult to measure, or the high E string would be filtered off and difficult to measure. Another advantage of the two band equalizer is that the boost and cut can be customized for unique guitar gains. One disadvantage of using the equalizer is that if the user wants to make a custom tuning with very high frequency (not typical), the high frequencies will be filtered off and maybe require additional gain stages.

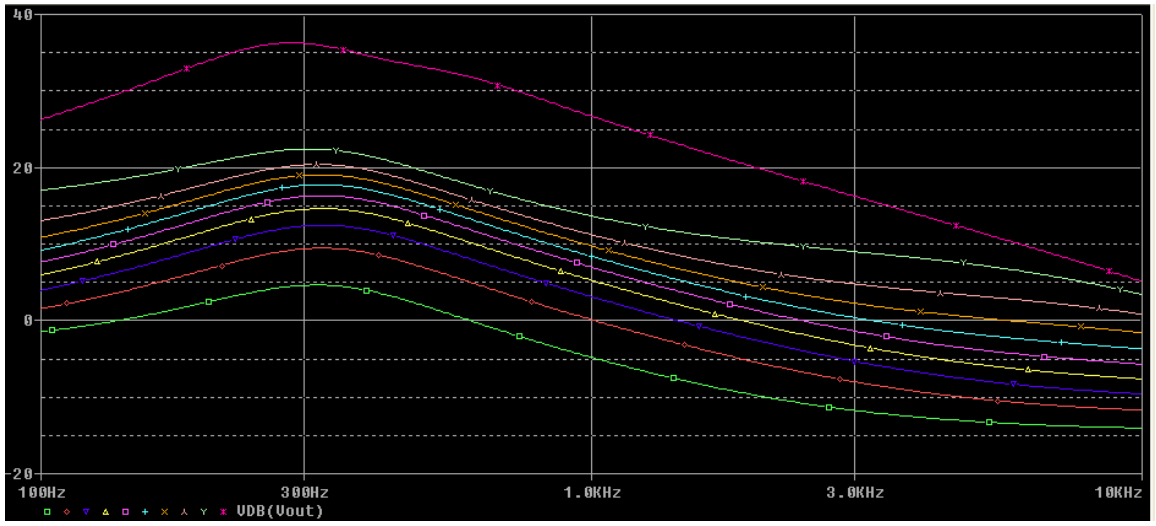

Figure 2. Frequency Response of the 2 Band Equalizer with Changing Upper Band Potentiometer Value

Figure 2 shows the frequency response of the filter as the upper band potentiometer is changed from fully cut to fully boost.

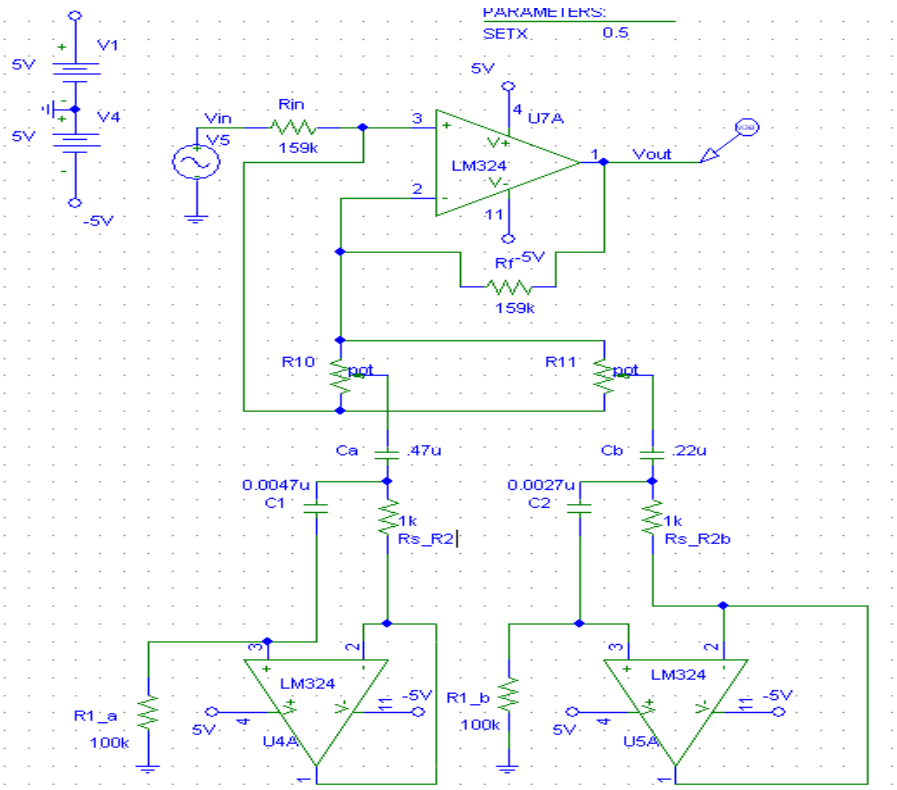

Figure 3. Two-Band Gyrator Based Equalizer Circuit

The calculations for the maximum boost of the gyrator circuit[2]:

Boost = Voltage Swing/$V_{p\text{-}pMinimum\_String}$ = 10V/62.5mV = 160V/V [44dB]

$R_{in} = R_F$ = Boost*R_S – R_S = 160*1kΩ - 1kΩ = 159kΩ

The center frequency for both bands were calculated by using the average frequencies of the top three strings and the bottom three strings:

f_1 = 365Hz

f2 = 648Hz

Giving each band a quality of 1, the capacitor values were calculated as follows:

$L_1 = (R_S * Q)/(2\pi*f_1) = (1k\Omega * 1)/(2\pi * 365Hz)$ = 436.041mH

$C_1 = (L_1/(R_2*(R_1 – R_2))) = (436.041mH/(\ 1k\Omega(100k\Omega - 1k\Omega)))$ = 4.404nF

$C_A = (1/(2\pi*f_1)^2)/L_1 = (1/(2\pi*365Hz)^2)/436.041mH$ = 436.041nF

$L_2 = (R_S * Q)/(2\pi*f_2) = (1k\Omega * 1)/(2\pi * 648Hz)$ = 245.609mH

$C_2 = (L_2/(R_2*(R_1 – R_2))) = (256.609mH/(\ 1k\Omega(100k\Omega - 1k\Omega)))$ = 2.481nF

$C_B = (1/(2\pi*f_2)^2)/L_2 = (1/(2\pi*648Hz)^2)/245.609mH$ = 245.609F

Rectification and Schmitt Triggering

Half Wave Rectification was used in order to place the signal within the digital TTL logic range (0-5V). Schmitt Triggering was used in order to transform the analog signal to digital pulses and to lessen the effect of harmonic distortion and noise. The half wave rectifier chosen was an active op-amp based one which was chosen for its high speed characteristics. The figure below shows both circuits.

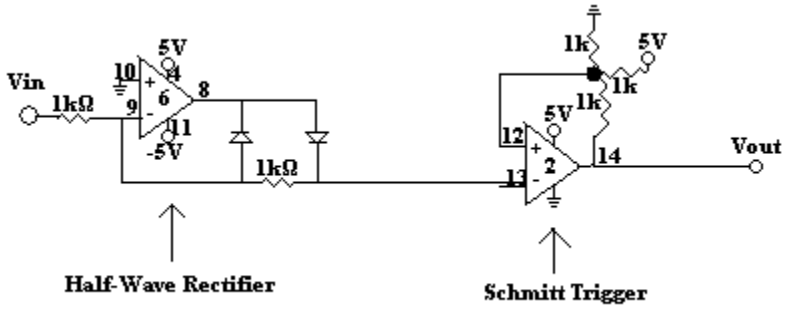

Figure 4. Half-Wave Rectifier and Schmitt Trigger Circuits

The Schmitt Trigger utilized a LM324 Op-Amp for its single power supply ability such that the output would represent TTL logic level ranges, and thus could be read by the microcontroller.

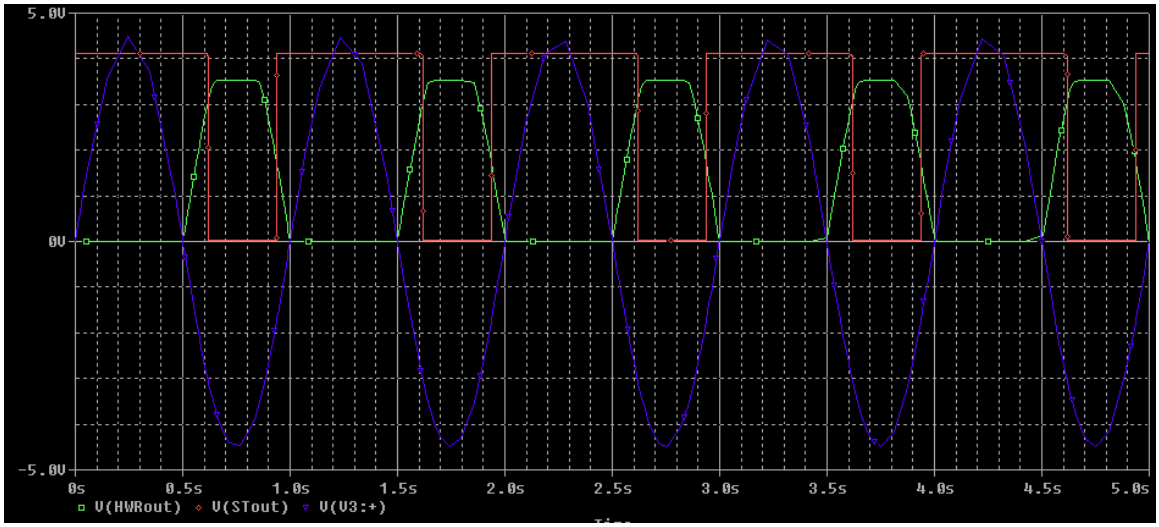

Figure 5. Simulated Waveforms for Half-Wave Rectifier and Schmitt Trigger

Figure 5 shows the simulated waveforms for the circuit in Figure 4. The blue waveform is the input coming from the 2-band equalizer, the green waveform is the signal after rectification, and the red signal is the TTL-logic digital pulses produced from the Schmitt trigger.

Analog to Digital Converter

An analog to digital converter was implemented in order to sample the guitar signal and let the system program know whether the signal's amplitude had decayed too far for proper measurement. If the signal decayed to a voltage smaller than 3V(peak), then the signal would not be able to produce an output of the Schmitt trigger. The ADC integrated into the microcontroller was used for this conversion. The maximum sampling rate of the ADC is around 75 kHz which is well above the suggested Nyquist sampling frequency of twice the maximum frequency. Choosing the ADC onboard the microcontroller had the advantage of having a lower cost, more circuit board real estate, and less programming space.

H-Bridges

In order to run the stepper motor in both directions and at different speeds, two custom H-Bridges were designed. The BJT transistors used were carefully chosen so that the required motor current could pass through without heat or breakdown issues, and they were also carefully chosen so that the controller (a GAL PLD) could provide enough current to put the transistors into saturation.

In order for the stepper motor's H-Bridges to share the bus, a driver needed to be placed which could enable to disable the H-bridges. A Programmable Logic Device was programmed to act similar to manufactured Stepper Motor Drivers in that it could step in both directions at different speeds, lock the motor in place, or de-energize the coils and save power. The Stepper Motor Driver was designed to work as a state machine for the transistor outputs.

Atmel Atmega168 Microcontroller

The Atmel Atmega168 was chosen for its versatile features, 2-cycle multiplier, free C language compiler, and in-system programming. Among the many features it is capable of, the features used were the timers with the different modes of interrupts which were used for timing analysis, 10-bit ADC with a maximum sampling frequency of 77 kHz for measuring the guitar signal's amplitude, Generic Pin Interrupt for timing analysis, and Port-Pin Change Interrupt for keypad processing.

Bus Architecture

In order to reduce the amount of pins needed, many things were placed on a common 8 bit bus. The devices that accessed the bus were the keypad, the stepper motor driver, the LCD, and the external RAM chip. Each device had an I/O Address in the system's memory map which made use of a Programmable Logic Device as an address decoder and bus arbiter. This approach lends itself well to the instruction of computing architectures in general for the students' benefit.

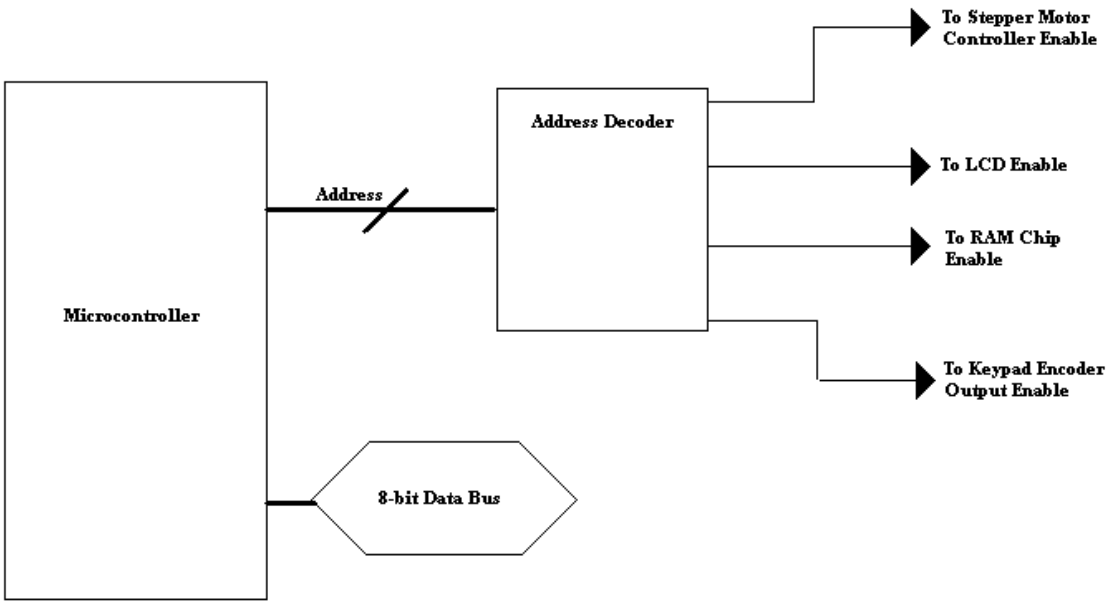

Figure 6. Bus Architecture Diagram

SOFTWARE DESCRIPTION

This section describes the general main code flow, the algorithm for determining frequency, and moving the stepper motor accordingly.

Main Code

The main loop of the code is interrupt driven and will wait for user inputs determined through a flagging system in the keypad's interrupt service routine, handle the system and LCD state accordingly, perform any tuning or EEPROM writing (if that is the current selection), and eventually clear the flag and continuing looping. The interrupt service routines are usually only a few lines of code, which include saving a single variable (such as the keypad input) and raising the interrupt flag. The pseudo code below shows a very simplified main loop.

```
Main Loop
{
        New Key? ~> Handle Keypad
                        {
                                Debounce();
                                    Update Text input or Command input
                        }
                         Handle LCD
                         {
                                Refresh_LCDscreen();
                                ...
                                    Need to Measure Frequency? ~> Select String
                                                                  Wait for Guitar Signal...
                                                                    Measure();
                                                                  Run_Stepper_Motor();
                                    Need to save to EEPROM? ~> Store Guitar Frequencies
                                                                  Store Save State Name
                         }
}
```

Frequency Measurement

When the program reaches the point where it is ready to measure the frequency, it will reset the timer value to zero and enable the interrupt coming from the processed guitar signal. When the first interrupt is received it starts the timer, and will not stop the timer until a predetermined number of oscillations have occurred. As the timer reaches its maximum value of 255, the overflow interrupt is accessed, and an overflow counting variable is incremented. After the predetermined number of oscillations has occurred, the clock will pause and the overflow count will be stored into an array along with the leftover time in the timer. The whole process is repeated several times in order to get an average number. This number is processed with simple math in order to calculate the average period, and thus the value is inverted and the frequency is known. All math functions utilize the double data type in order to receive decimal precision. The figure below shows an abstraction of the process.

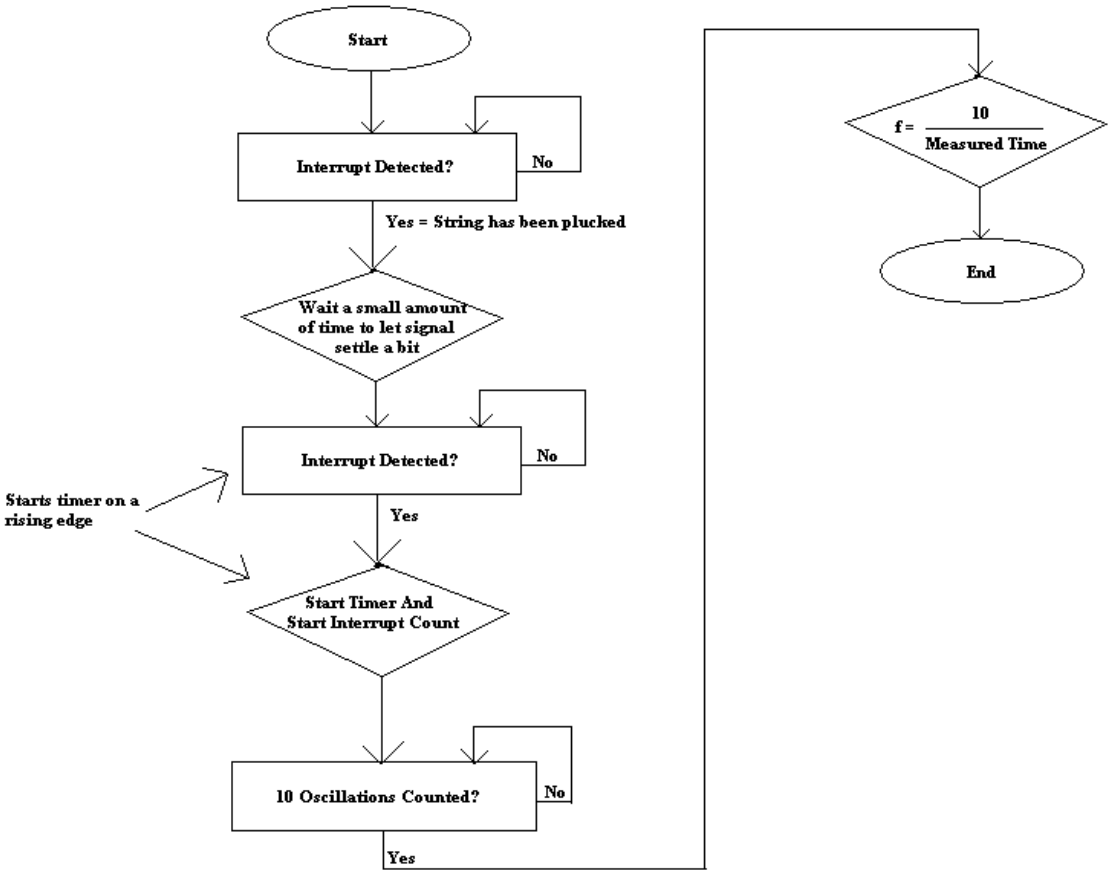

Figure 10. Determining Frequency

To ensure precision, the system will repeat the measuring process until it has either measured the frequency 20 times or the measurement is drastically different than the first measurement, suggesting the signal has become tainted.

Tuning With the Stepper Motor

After the frequency has been measured, an error signal is calculated and used to determine the number and direction of steps to minimize the error. This calculation is a simple mathematical product of knowing that each step of the stepper motor changes the frequency about 0.4 Hertz (determined experimentally) around the guitar's typical range. Interfacing with the driver (a PLD state machine) is as simple as setting which direction bit to activate, and toggling a clock input. For higher torque needs, a frequency of 25Hz was chosen as the stepping frequency.

CONCLUSION

On demonstration day, the system was running very satisfactory and the student could orally describe accurately and concisely the user interface and key functional blocks as guitars were actively tuned. Every aspect of the project worked with the exception of EEPROM saving the user's floating point precision frequencies. The tuner was able to determine frequency with 0.1Hz precision, a precision that only well trained musicians can hear. Improvements to the gyrator based equalizer Q that could effectively eliminate all the harmonics was identified as a potential improvement to the system frequency precision. Also, the stepper motor had a stepping resolution that approximately equaled 0.4Hz in guitar frequency, which is still agreeably pleasant to the ears. This feature could be improved by upgrading the stepper motor driver to achieve half-stepping, a feat that the GAL22V10 did not have enough hardware to perform.

Student summary of learning: " The project was able to satisfy every class requirement given, and in making it has not only enhanced my digital knowledge, but also my analog processing skills, my understanding of computer architecture, and my appreciation for the power of microcontrollers."

Professor summary of learning: Project met all requirements and by following the systems engineering approach to the planning and documentation the student enhanced his organizational skills and created a great project document to use during job interviews that displayed his technical breadth and depth. The tradeoffs of filtering and frequency counting musical signals, in this case guitar strings, was a particularly interesting application to reinforce many concepts taught is other courses. This project exceeded requirements by fabrication of a through hold board for the construction of the system and much was learned about board layout and some of the board layout software tools which are typically used in industry. Mr. Pettingill is now leading microcontroller projects in industry shortly after his graduation from SPSU.

References

[1] Pierce, John R, *The science of musical sound*, Scientific American Library, New York, 1983
[2] Talbot-Smith, Michael , *Audio Engineer's Reference Book*, Focal Press, Great Britain, 1984

Charles Duvall **is an Assistant Professor in the Electrical and Computer Engineering Technology department at Southern Polytechnic State University located in Marietta, Ga. He has been teaching for seven years after working as an R&D developer at Bell Labs and as an optical network architect for a number of optical component and subsystem companies. He holds a Bachelor of Science from Southern Polytechnic State University and a Master of Science from Georgia Institute of Technology.**

Ross Pettingill **is a development engineer with Georgia Tech Research Institute**. **He holds a Bachelor of Science from Southern Polytechnic State University.**