Real Time Implementation of a Tuning Device Using a Digital Signal Processor

Joseph Reagan, Sedig Agili and Aldo Morales
E E/ E E T Programs
Penn State University at Harrisburg
Middletown, PA 17057

Abstract

In this paper, an example of the use of a digital signal processing (DSP) board for teaching purposes and project implementation is demonstrated. Project implementation is carried out through the design of a guitar tuner using a digital signal processor (DSP) platform. The goal of the design is to optimize the DSP process so that the best results are obtained for the guitar tuner. This involves the proper selection of DSP parameters and use of DSP techniques to provide an accurate guitar tuning such as frequency resolution and sampling rates. The application was implemented in a target hardware system, the TMS320C5402 DSP Starter Kit (DSK). The program for the guitar tuner is written in the C programming language and makes use of DSP assembly functions provided by Texas Instruments. The program developed for the guitar tuner can be run in the Code Composer Studio (CCS) Integrated Development Environment (IDE) and is able to accurately tune a guitar in several common tunings. This project is interesting for the students to learn real time implementation issues of different DSP techniques.

I. Introduction

Penn State Harrisburg offers BS EE, BS EET, and ME degrees. The Bachelor of Science degree in Electrical Engineering provides an opportunity for students to pursue interests in electrical and electronic circuits, including digital circuits and VLSI and its fabrication, microprocessors and their applications, electromagnetics, communications, control systems, digital signal/image processing and computer vision. The BSEET program provides similar experience however, its strengths include: an applied, hands-on approach and extensive laboratory experience. Through a senior capstone design project, both curricula emphasize written as well as verbal communication and a teamwork approach among students to attain a common goal.

The Master in Electrical Engineering offers an education in the advanced aspects of modern electrical engineering. A candidate for the Master of Engineering in Electrical Engineering must write a scholarly report or engineering paper and defend it before three faculty members.

Thus, students in the DSP, communication, senior project courses and master paper have ample opportunities to use real-time DSP techniques. In fact, the design and implementation of this guitar tuner using a DSP board was accomplished by a graduate student. The program has acquired a number of real-time TMS320C6x DSK DSP boards to be used in undergraduate and graduate teaching and research. Currently, several students are involved in implementing real time DSP applications such as guitar tuner using newer boards, and test generation patterns for interconnects.

The following is a demonstration in how digital signal processing (DSP) techniques can be used to develop and implement a guitar tuner. A guitar tuner is a device that accepts musical signals as inputs and compares these to standard pitch notes. It displays the pitch of the input signal relative to the standard pitch. This display aids the user in the process of tuning the input signal so it equals the standard pitch. The process of tuning a guitar is an iterative process where a string on an instrument is plucked until the user sees that its frequency is as close as possible to a desired concert pitch. Guitar tuners are often used in either a manual mode, where a fixed note is being tuned to, or in an automatic mode where the tuner detects a note and locks onto it and allows the user to tune to that note. In this paper, both modes are developed. The guitar tuner provides visual feedback through an UART on the DSP kit and a terminal program resident on a PC, such as HyperTerminal. In order to demonstrate the flexibility of DSP, several guitar tunings used by guitarists are provided, such as Standard, Open G and Open D. The program provides a menu from which the user can control various modes of operation.

Central to the guitar tuner are signal processing functions, which acquire a signal and determine the pitch of a note. To determine the note's frequency, it must be sampled and then converted to the frequency domain using the Fast Fourier Transform (FFT). Additional signal processing operations are also performed to improve the results. These include filtering, windowing, and zero-padding. In addition, an algorithm that can determine whether a note is being mistaken for the harmonic of another note is presented. The goal of the design is to optimize the DSP process so that the best results are obtained for the guitar tuner. This involves the proper selection of DSP parameters and use of DSP techniques to provide an accurate guitar tuning such as frequency resolution and sampling rates. To make this paper self-contained, a brief discussion on guitar fundamentals is presented in section II. Hardware and Software selection is presented in section III. Implementation of the guitar tuner using the TMS320C5402 DSP Starter Kit (DSK) is discussed in section IV. The performance of the guitar tuner is evaluated and discussed in section V. Conclusions are given in section VI.

II. Musical Tuning and Guitar Fundamentals

A note is a name given to describe a musical frequency[1]. The chromatic scale is typically used in Western music and consists of the following twelve notes:

A , A# , B , C , C# , D , D# , E , F , F# , G , G#.

The "A" note has been chosen as a standard frequency upon which the other notes' frequencies are calculated. These 12 notes repeat for each octave[1]. The frequency of the notes doubles for each successive octave. Notes are designated by their letter, and the octave they reside in. For example, A = 440 Hz is in the 3rd octave and is called "A3". Table 1 lists the chromatic scale of notes over a three octave range which contains the notes of the guitar tunings used in this paper.

Note	Octave1	Octave2	Octave3
A	110.00	220.00	440.00
A#	116.54	233.08	466.16
B	123.47	246.94	493.88
C	130.81	261.63	523.25
C#	138.59	277.18	554.37
D	146.83	293.66	587.33
D#	155.56	311.13	622.25
E	164.81	329.63	659.26
F	174.61	349.23	698.46
F#	185.00	369.99	739.99
G	196.00	392.00	783.99
G#	207.65	415.30	830.61

Table 1 Chromatic Notes

A guitar is a stringed instrument, and typically has six strings. The musical notes which are played on a guitar are created by the plucking of one or some number of its strings at different locations on the guitar. These locations where notes exist are created by frets. A guitar's fretboard is designed so that the 12 notes of the chromatic scale are positioned from the open position to the 12th fret. After the 12th fret, the notes repeat at the next octave. The 12th fret is designed so that it is always double the frequency of the open string frequency. The desired notes are obtained by applying the proper tension to the guitar strings by means of the tuning pegs. The tuning pegs are turned in either direction to increase or decrease the string tension. A guitar must be tuned when new strings are put on a guitar, and periodically thereafter to maintain the concert pitch of the instrument. The main reason a guitar comes out of tune is the playing of the guitar, which will cause the strings to stretch. Other factors, which affect the tuning, include humidity, temperature and time.

A guitar is typically tuned to what is called Standard tuning. A tuning is always defined by the notes which exist when the strings are left open, or unfretted. The Standard tuning has grown to be the most popular tuning because of the many chord variations which can be easily formed in it. There are many other guitar tunings in use, but many of these are for more specialized forms of music[1]. For example, Open G is often used in folk music and especially by guitarists using a slide (bottleneck), since a Major chord can be formed by placing the slide straight across the fretboard. Open D is also very popular with folk guitarists. Table 2 lists the notes, which occur in these three tunings, all of which can be tuned to with the guitar tuner in this project, and their corresponding frequencies.

Standard Tuning		Open G Tuning		Open D Tuning	
Note	**Frequency**	**Note**	**Frequency**	**Note**	**Frequency**
E1	164.81	D1	146.83	D1	146.83
A2	220.00	G1	196.00	A2	220.00
D2	293.66	D2	293.66	D2	293.66
G2	392.00	G2	392.00	Gb2	369.99
B3	493.88	B3	493.88	A3	440.00
E3	659.26	D3	587.33	D3	587.33

Table 2 Notes for guitar tunings used in this project

A guitar tuner is a device that provides the desired musical pitch and provides some feedback to the user as to what the present pitch is. Two of the simplest guitar tuning devices are a tuning fork and a pitch pipe. A tuning fork is a simple tuning device that resonates at a known frequency. The user must tune the string by ear to this tone. A pitch pipe works in a similar fashion. Both of these devices rely on the user's ability to hear slight differences in frequency in order to tune with this method. Since many users do not have the natural hearing ability to tune this way, guitar tuners have been developed which provide feedback that indicate the tuning status. Of course, there are other reasons more sophisticated tuners have been developed. Some of these reasons include speed, the ability to tune all of the open string notes on a guitar or even the entire chromatic scale of notes. Many of the tuners also provide an input jack so an electric guitar can be tuned without other sounds disturbing the process, which is an important feature for a musician on stage.

III. Hardware and Software Platform

The hardware used for implementation is the TMS320C5402 DSK from Texas Instruments[2] (TI). The DSK is a PC Board which contains the TMS320C5402 (C5402) DSP processor and supporting circuitry. Figure 1 shows parts of the functional block diagram of the DSK that have been implemented in this project[3]. In what follows, some of the main function blocks are explored.

The C5402 is a 16 bit fixed point DSP[2,3] with an architecture that has one program memory bus and three data memory buses. Separate program and data spaces allow simultaneous access to both program instructions and data. The C5402 CPU has a 40-bit Arithmetic Logic Unit (ALU) and can access 192K words of memory.

The audio input to the board is established through a microphone or a phone jack to a pre-amp IC. This signal is then presented as an input to the codec. The codec (AD50) provides both Analog to Digital Conversion (ADC) on receive, and Digital to Analog Conversion (DAC) on transmit. The codec requires a one-pole passive anti-aliasing filter, which is implemented on the DSK board. The codec interfaces to the C5402 DSP via the serial port named MCBSP 1. The codec is only being used as a receiver in this project.

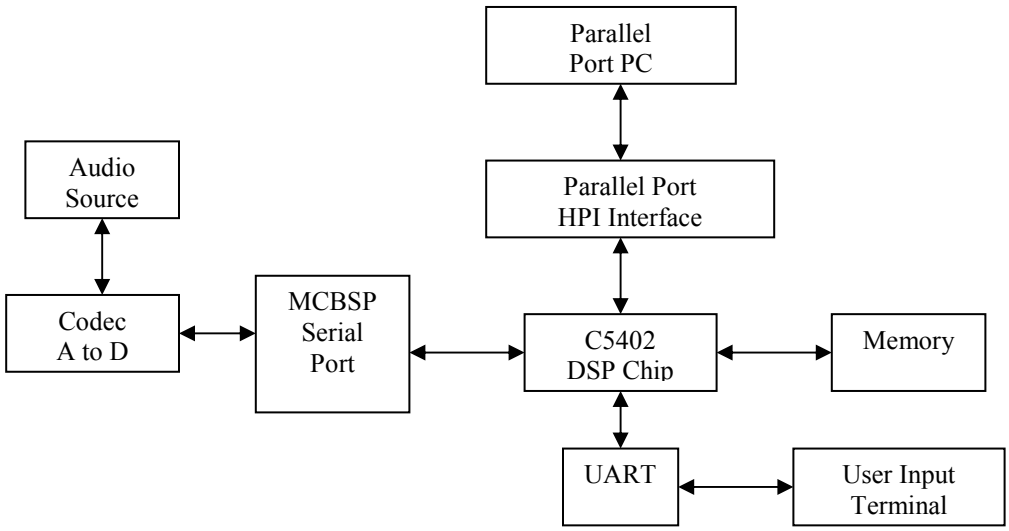

Figure 1 DSK Functional Block Diagram

The DSK board interfaces to the parallel port of the PC through its host port interface (HPI). Programs are downloaded to the target via the HPI interface to be run and debugged. The HPI allows Code Composer Studio to communicate with the target hardware. An on-board UART allows serial communication to an external device. In this project, the UART is used to communicate to the serial port of a PC using HyperTerminal. Refer to the TLC16550 Data Sheet for more details on the UART[4].

Code Composer Studio (CCS)[5] is the development environment provided by TI for C5000 family DSP development. CCS was used to implement the guitar tuner on the DSK board. Functions provided by CCS include: compiling, linking, debugging, loading data files, input signals and filter coefficients. When using the DSK board, signals can either be input to hardware through the AD50 codec, or from a file input, which is called File I/O in the CCS environment. File I/O is useful in program development as test signals can be generated via software programs such as Labview or MATLAB and entered into memory. This technique can also be used to output results to a file, where they can be analyzed in another software program.

The DSK is included with libraries of functions[6,7] written in C by TI which allow various peripherals such as the codec, UART, LEDs, serial port, and other hardware on the board to be operated. The header file for a group of functions must be included in the source program to use them. Also included are several DOS utilities which test the board, reset it and program the flash memory. Texas Instruments provides a library of C callable assembly functions for DSP called dsplib. In this project, the program is written in C and most of the DSP functions are from the dsplib library.

IV. Implementation

In the CCS environment, a project file must be created. A project file is a file which manages the various files used as well as option settings for compiling, linking and

assembling the program[5]. The Load Program command in CCS is used to download the compiled and linked object file to the DSK board. Before the guitar tuner code can be run, the hardware must be initialized. The function *brd_init()* sets the 5402 CPU frequency to 40 MHz and performs other initialization tasks for the DSK board. The codec is initialized through the use of the *setup_codec()* function, which sets codec input gain, sample rate and output gain. The UART is also initialized by the *uart_reset()* function.

Once an analog signal is sampled, it must be stored in memory so that the signal processing functions can be performed on it. The following arrays were used for signal processing in this paper:

- The input array *x* that the audio signal is stored in after it is acquired by the codec. The maximum input size is equal to 1024.
- 11 arrays used for filter coefficients, designated as hE1, hA2, etc, to indicate which note the filter is designed for. These are stored in program memory as constants.
- The array *r* used as output of the *fir()* filter function, and as an input to other functions.

Figure 2 shows the audio signal devices that were used to create and modify the input signal. The signal generator used was a commercial guitar tuner which outputs the chromatic scale of notes over three octaves. The frequencies of these notes provide an accurate tuning reference that can be used to evaluate the guitar tuner. The distortion pedal is used since many guitar players utilized it to add harmonic content to the signal spectrum. The equalizer is useful for shaping the spectrum of the signal.

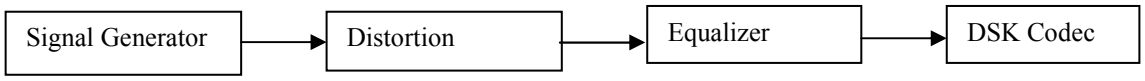

Figure 2 Audio Signal Devices

The signal processing operations performed on the input signal, which was acquired into *x*, are detailed below:

- Clear the output buffer, *r*
- FIR Filtering
- Copy function
- Windowing
- Zero-Padding

The filtering, windowing and zero padding operations are not required to perform the basic frequency determination, but they are used in the program to improve the performance. The filter and window functions can be switched in and out of the program. The *fir()* function is from the dsplib library, and the remaining functions were written for this project. The following C functions correspond to the above operations, respectively:

- *void Zero_Buff (u16 count,DATA *input);* Clear the output buffer *r*.

- *fir(x, h, r, &dbptr, NH, NX);* This is the *fir* filter function from dsplib. It takes as input the array *x*, which is the input signal; and the array *h*, which is the filter coefficients. *NH* and *NX* are the sizes of the arrays. The output of the filter is put into the array *r*. The arrays for *h* are circular buffers, and must be aligned on a memory boundary such that the *k* Least Significant Bits (LSB)s of the starting address are zeros; where $k = \log_2(NH)$.

- *void Copy_Buff(u16 count,DATA *input,DATA *output);* This function is required to copy the input buffer *x* to the output buffer *r*, when the fir function is not used. This is because the FFT functions were designed to always operate on the *r* buffer.

- *void Window(u16 n,DATA *in_ptr);* This function takes as an input a pointer to the array to be windowed and multiplies it by a Hamming window function. To perform this function the cosine function must be used on data of type double. This data is type-cast to float, and then the dsplib function *flt2q15* is used to convert the data to Q15 format (used in the DSK). The data must be shifted by 15 since the result from the multiplication is 32 bits and needs to fit in a 16-bit word.

- *void Zero_Pad (u16 input_size,DATA *in_ptr);* This function provides zero padding to the input signal. This is required to ensure that the input is a power of 2, as required for the FFT. The FFT size is fixed at 1024 so this function always outputs an array of this size.

So far, we have discussed how to setup the project, define variables, initialize hardware, acquire input, and perform the basic DSP process of determining a frequency of a note. In the following, the implementation of the guitar tuner is discussed in detail.

It is an objective of this paper to provide the three guitar tunings listed in Table 2. The highest frequency note to be tuned is E3 at 659.26 Hz. To avoid aliasing, the minimum sampling rate must be greater than twice the E3 frequency; which is 1318.52 Hz. A sampling rate of 2000 Hz, which is the minimum the codec can be operated at, satisfies the Nyquist criterion of these notes if the signals are pure sine waves. However, the signal input is not guaranteed to be a pure sine wave. Many guitar effects, especially distortion, will change the wave shape of the signal. Therefore, the highest signal present would not be exactly E3, but higher. The highest frequency component present would be 660 * 3 = 1980 Hz, if the 3rd harmonic is significant. This would in turn require the sampling rate to be set at 4000 Hz. Experiments were performed with the sampling rate and with the type of audio signal to determine if these parameters need to be adjusted.

IV.1 Frequency Resolution

Frequency resolution is defined as: $\Delta f = \frac{f_{samp}}{N}$. It is desirable for Δf to be as small as possible for greater accuracy. The sampling rate must be chosen to avoid aliasing as discussed above, but also to keep Δf low. Since time domain wave shape accuracy is not required, a high sampling rate is not optimal. The more points, *N*, used as input to the

FFT, the lower Δf will be. However, processing speed and memory requirements will increase as N is increased. The maximum N is limited to 1024 due to the dsplib FFT function *rfft*. The approach taken is to fix N at 1024 and attempt the design with a sampling rate of 2000 Hz, which will yield Δf = 2000/1024 = 1.95 Hz, which is the best obtainable resolution in this design. The input signal size acquired is programmable to determine if an input frame of 1024 is too slow. Zero-padding is used to assure the signal is a power of 2, as required for the FFT. Table 3 summarizes these parameters.

Analog Bandwidth	~660- Hz fundamental
Input Size x	128-1024 variable
FFT Size N	1024 fixed
Sampling Rate	2000-8000 Hz
Δf	1.95 Hz min- 9.80 Hz max

Table 3 Guitar Tuning Specifications

As shown in Table3, different input sizes were used to determine if speed was an issue. Since the frequency of the desired note is known in advance, a filter used at that note's frequency passes notes in that range and discriminate against others. A FIR filter was used with the option of switching it in or out of the program. This flexibility allows experimenting with the filter's effects, and to determine if it was helpful to minimize unwanted signal content such as harmonics, notes outside the range of interest, and other noise. A window function that minimizes leakage was also used, and switched in and out to observe its effects in the program.

IV.2 Range Determination

Once a notes frequency was determined, the objective was to determine the pitch relative to the concert pitch of the desired note. This was accomplished by establishing the following four ranges:

- InTune - the range around the center frequency of the note that can be considered in tune, or at concert pitch.
- Sharp - this range is above in tune but below the High In-range value.
- Flat - this range is below in tune and above the Low In-range value.
- Out of Range- the note is outside the range of interest.

Table 4 shows an example of range definitions used for a center frequency of 440 Hz.

<418.00 Hz	418-435 Hz	435-445 Hz	445-462 Hz	>462 Hz
Out of Range	**Flat**	**In Tune**	**Sharp**	**Out of Range**

Table 4 Tuning range example for 440 Hz

Two methods can be used to determine these ranges. They can be calculated ahead of time and entered into the program as constants for each note, or they can be dynamically calculated in the program. This calculation is based on the center frequency, f_c, and two factors that define the range around the central frequency.

The function *CalcInTune* is used to calculate the range of the present note being played. This function uses a center frequency array (*CenterFreq*) from which the standard pitch of the note is selected using the values of table 2. In addition, two variables are used to calculate the ranges: *InTuneFactor* and *InRangeFactor*, to determine the four variables in the Note structure described in table 4. The *InTuneFactor* was set to .01, which was about as accurate as the tuner can get since Δf was constrained to 1.95 Hz minimum. In addition to frequency ranges, an amplitude threshold variable was used. This helped in preventing small noise signals from triggering the guitar tuner. This value was calculated with the dsplib function *maxval(r, NX/2)*.

A function called the Harmonic Detection Algorithm in the DSP library, uses existing software to determine if a detected frequency is a fundamental tone or a harmonic. It does this by filtering a time domain signal, and taking the FFT and determining the frequency. This is repeated with the filter removed. If both of these frequencies *f1* and *f2* are nearly equal, then it can be deduced that the frequency is the fundamental and it is passed to the CalcInTune function. If these frequencies are not equal, then it can be deduced that the filtered tone is a harmonic. The non-filtered frequency is then passed to the CalcInTune function. The assumption being made is that the fundamental frequency will always have a higher magnitude than any of its harmonics

There are a total of eighteen center frequencies to be used, counting all three guitar tunings (refer to table 2). A filter, which is an array of coefficients, is required for each note. The coefficients for each filter must be included in the program. The FIR function accepts as input a pointer to an array of these coefficients. Since many filters are to be used, and the FIR function is called only once, a 2D array of pointers was used. Each row of the array contains pointers to the filters of the 6 notes of a guitar tuning, in the same way as the 2D array of center frequencies does. As can be seen in Table 2, several of the notes in the table are used multiple times. This leads to an efficient implementation, since it allows for some of the filters to be reused. It turns out that a total of eleven filters need to be implemented out of the eighteen total notes for the three guitar tunings. It was decided based on trial runs that a 63-tap FIR filter provides adequate filtering. Filters with 25 coefficients provided only minimal attenuation. Memory size is another reason for selecting a 63-tap filter. 63 is under 64, which is 2^6, then the coefficients can be easily aligned on a memory address with the 6 LSBs = 0. If a number greater than 64 were selected, then the next power of 2 is 128. Then memory would be wasted between these 128 word segments. For 11 filters, memory saving requirement is significant. In addition, FIR filters become computationally expensive as the number of coefficients becomes large.

For a guitar tuner, we also need to display tuning results in a user interface. It also allows the user to select various modes of operation and control the guitar tuner setup. When a note is played on the guitar, it will decay within a certain period of time. Then it is important that the display is updated fast enough so the user can respond to the results and adjust the tuning accordingly. This is the only real-time constraint of this design. One indication of program execution speed is the LEDs which are on the DSK board. The LEDs can be toggled at certain places in the program to give an idea how long certain sections of code take to execute. This is not by any means an exact measure of the speed.

For exact timing, CCS has built-in functions in DSP BIOS for accurately measuring execution time. The LEDs themselves cannot convey enough tuning information, although three LEDs could be used to indicate Flat, Sharp and In Tune. The following additional parameters can also be displayed: Concert pitch of desired note, Mode of Operation, Guitar Tuning, String being tuned, Frequency of concert pitch, and Frequency of current note. In addition, a menu is built to allow the user to put the guitar tuner in the desired mode, and provide user options. The user interface uses a PC's serial port, and the HyperTerminal program.

V. Results and Discussion

In this section, results are presented which validate the functionality of the guitar tuner developed in this project. In order to obtain these results, the program was compiled, loaded into hardware, and run using CCS. Next, the tuning of an actual guitar is presented.

The guitar used was a Fender Telecaster Deluxe, retrofitted with Dual EMG switchable active pickups. The harmonic algorithm was switched in for this test.

- Standard Tuning: With an A2 already tuned to 220 Hz, and the D2 out of tune at 300 Hz (f_c = 294); the two strings are noticeably out of tune. The D2 was out of tune by 6 Hz only. When tuned to 294 Hz, the strings sounded good. When tuning the E1 note, the frequency resolution is probably not quite adequate since the frequency will jump from 166 to 164 Hz, and even 1 Hz off is noticeable. The harmonic algorithm was switched out and the filter switched in, and an A2 harmonic can be mistaken for the E3 note. The harmonic algorithm prevented this when it was in. In Auto Mode, all of the 6 notes were correctly identified. The tuning reverts to the Manual mode once this occurs, and allows the found note to be tuned. It reverts to scanning once the timeout occurs.
- Open D Tuning: The Open D Tuning was selected via the Menu. The guitar was tuned to Open D. There was some trouble for the Auto Mode to detect the A3=440 Hz note, it was often detecting this note as A=220 Hz. This is due to the harmonic content of the guitar string. When dropping the tuning of a string by a whole step, it is probably better to use the guitar tuner in manual mode as it is difficult to know where a string is as it is dropped this far.
- Open G Tuning: The Open G Tuning is also selected via the Menu. There were no problems when tuning to Open G.

VI. Conclusions

The performance of the guitar tuner met the requirements. The Δf accuracy of 1.95 Hz was achieved and provided adequate tuning. The anti-aliasing filter removed some harmonic content of notes that were over the Nyquist frequency. A window function was programmed which showed that leakage could be reduced. However, the results of this project did not seem to rely to heavily on windowing. This is because the FFT analysis was straightforward and only the peak value was being detected.

The harmonic content of guitar signals can be troublesome for guitar tuners. The harmonic detection function provided a simple but effective algorithm to determine if a detected note was a fundamental or a harmonic component.

While all of the tuning concepts could have been simulated, it is important to implement them in real time. This was successfully done on the TI DSK target. Some of the key issues to deal with in the implementation were the number formats required for the signal; in this case, the Q15 format. It is also important to convert data types of variables when performing certain operations. For example, the *window* function required typecasting data from double to float to Q15 format. Another important issue is the use of arrays for signals, and alignment of signals in memory where necessary.

This project demonstrates the use of real-time DSP in an educational environment. Through the recent acquisition of several DSP TMS320C6X DSK boards, a number of graduate and undergraduate students are currently involved in implementing real time DSP applications such as guitar tuner, using the newer board, and test generation patterns for interconnects. The authors feel that the guitar tuner application example, given in this paper, can be used as a model for real-time DSP projects in other engineering schools.

Bibliography

[1] http://www.tyala.freeyellow.com

[2] Texas Instruments, TMS320C54x DSP, CPU and Peripherals Reference Set Volume 1, Literature Number SPRU131F, April 1999.

[3] R. Chassaing and D. Horning, *Digital Signal Processing with the TMS320C25*, 1990, John Wiley & Sons, pp. 105-146.

[4] Texas Instruments, TL16C550C UART Data Sheet, 2001

[5] Texas Instruments, Literature Number SPRU328B, Code Composer Studio Users Guide, February 2000.

[6] Texas Instruments, *TMS320C54x, Optimizing C Compiler,* Literature Number SPRU103D, December 1999.

[7] Texas Instruments, *TMS320C54x DSP Library Programmers Reference,* Literature Number SPRU518, April 2001.

JOSEPH REAGAN

Joseph Reagan was a graduate student in the Master of Engineering program at Penn State Harrisburg. He obtained his M.E. degree in the Fall 2002. He works as Senior Electrical Engineer at Dentsply Professional Division, York, Pennsylvania.

SEDIG AGILI

Dr. Agili received his BS, MS, and Ph.D. in Electrical and Computer Engineering from Marquette University in 1986, 1989, and 1996, respectively. He has been teaching and conducting research in electronic communications, fiber optic communications and fiber optic sensors at Penn State University at Harrisburg since August 2001.

ALDO MORALES

Dr. Morales received his electronic engineering degree with distinction from the University of Tarapaca, Arica, Chile, and M.S. and Ph.D. degrees in electrical and computer engineering from the State University of New York at Buffalo. His research interests are digital signal and image processing, and computer vision. He is now an Associate Professor of Electrical Engineering at Penn State University at Harrisburg.