8-1-2012

A Laboratory-Based Course in Real-Time Digital Signal Processing Using the TMS320C40

Scott E. Budge
Utah State University

Recommended Citation
S. E. Budge, "A laboratory-based course in real-time digital signal processing using the TMS320C40," in Proceedings of the DSPS Fest. Texas Instruments, Aug. 2000.

A Laboratory-Based Course in Real-Time Digital Signal Processing Using the TMS320C40

Scott E. Budge
Dept. Of Electrical and Computer Engineering
Utah State University, Logan, UT 84322-4120
E-mail: scott.budge@ece.usu.edu

Abstract

A three credit hour course in real-time digital signal processing (DSP) developed at Utah State University is described. This course has become mature as the result of incorporating five years of classroom experience and student feedback into its development. The educational goal of the course is to teach seniors/first year graduate students how to select, implement, and evaluate DSP systems for real-time signal processing. A major component of the class is a series of six laboratories in which the student must perform real-time processing on hardware based on the Texas Instruments (TI) TMS320C40 ('C40) DSP processor. These laboratories are followed by a DSP term project of the student's choice which shows a complete understanding of how to implement a relatively complex DSP algorithm on the 'C40.

1 Introduction

Recent surveys of industry leaders in the field of electrical engineering by the Department of Electrical and Computer Engineering at Utah State University (USU) have identified many of the abilities they look for in graduates from engineering schools. One of the current needs is for electrical engineers with experience in real-time systems.

To meet the needs expressed by industry, a three credit hour course in real-time DSP has been developed at USU. The goal of the course is to give graduating seniors and first-year graduate students experience in practical issues associated with working DSP systems, including how to select, implement, and evaluate DSP systems for real-time signal processing. To meet the need for practical experience working with currently available hardware, students complete a series of six laboratories and a capstone term project.

The course, ECE 5640, is divided into three main sections. The first section is intended to familiarize the student with the general architectures of DSP integrated circuits and DSP cores available on the market today. It is based on the material in "DSP Processor Fundamentals" by P. Lapsley, J. Bier, A. Shoham, and E. Lee [1]. A short section on Very Long Instruction Word (VLIW) architectures is included, which uses the TI TMS320C6x family as an example case. The second part of the course concentrates on the theory of DSP system design, including canonical, lattice, and state-space realizations, number representations (including the 'C40 floating point), fixed-point arithmetic, and finite word-length effects. The effect of coefficient truncation/roundoff, the need for scaling, and signal-to-noise ratio (SNR) degradation due to fixed-point arithmetic are particularly emphasized. The final third of the course is dedicated to completion of the term project. The students are given time to design, test, and complete a written report of the project they have selected. The requirements for the project are the it should take approximately 30 hours to complete, execute in real-time, and the algorithm should take advantage of the advanced features of the 'C40.

The laboratory portion of the class is based on the 'C40 and an A/D–D/A conversion system capable of sampling up to 48 kHz. The 'C40 was selected over other possible processors because of the capability of the processor to simulate fractional fixed-point arithmetic.

In Section 2 of this paper, we describe the curriculum of the course, including the architecture review and the theory of DSP system design. In Section 3 the six required laboratory exercises are described in

detail, including examples of typical programs. Section 4 describes the requirements for the term project, and in Section 5 a discussion of the educational experience and conclusions are presented.

2 Course Curriculum

The first ten to twelve weeks of the course are devoted to lecture presentation of the architectural features of several families of DSP processors and the theory of practical implementation of DSP systems. The texts for this material are Lapsley, *et. al.*, *DSP Processor Fundamentals* [1] and the introductory DSP text by Proakis and Manolakis [2].

2.1 DSP Architectural Features

The first topics in the course include a review of the material contained in the text by Lapsley. Lapsley's text is very tutorial in nature and provides a general overview of architectural features found in modern DSP families. Although it is a few years out of date, the architectural principals it contains are fairly independent of recent DSP processor development. A significant exception to this is the advent of the TI VelociTI VLIW architecture.

To supplement the text, and to prepare the students for the laboratory exercises they will be completing, architectural features are presented using the 'C40 DSP processor as a specific example. The students are given reading assignments in the 'C40 *User's Guide* [3] which correspond directly with the material contained in Lapsley.

The list of the topics presented in this section of the course include:

- Digital Signal Processing and DSP Systems. A brief review of the properties of digital signals and corresponding processing systems and applications is given. The students are presented a brief history of DSP system evolution and are introduced to the tradeoffs required by any application, including speed, cost, power, etc.

- DSP Processor Embodiments and Alternatives. These include processor implementations such as DSP cores, chips, multi-chip modules, VLIW, and both embedded and general purpose processor systems.

- DSP Processor Characteristics. This topic is introduced by analyzing the requirements to compute a simple FIR filter output. The resulting analysis leads to common processor features such as the Harvard Architecture (and its variants), pipelining, hardware multiply-and-accumulator (MAC), hardware loop control, circular (and bit-reverse) indexing, etc. Other features such as fixed- and floating-point, addressing, instructions sets (with special DSP-friendly instructions), interrupts, and peripherals are presented.

- Also included in this section of the course is a review of the features of the TI 'C40 DSP. We introduce the instruction set, memory map, cache and on-board RAM, addressing, and peripherals that are specific to the 'C40. This is presented early in the course in parallel with the general material from Lapsley so that the students are able to progress in the laboratory exercises from the first week of the course.

2.2 DSP System Design

After the general architectural features of modern DSPs are presented, the students are introduced to the principals of DSP system design. This section of the course is independent of the architecture of the implementation, except for the obvious differences in fixed- and floating-point implementations. The level of detail of the material presented at this point is much higher than the system implementation presented in our other introductory DSP course (ECE 5630), which is primarily a theory-based course. Material for this section are presented from Chapter 7 and Section 6.4 of Proakis [2]. Topics include:

- System realization types, such as Direct Form I and II, cascaded and parallel, lattice (both FIR and IIR), lattice-ladder, and state-space.

- Number representations. Floating- and fixed-point, including 'C40 floating-point and Two's-compliment are reviewed, and the effect of truncation and rounding of real numbers to fixed-point are discussed. We have also found it very useful to discuss the multiplication of both integer and fractional fixed-point numbers.

- Finite word-length effects. This topic includes coefficient quantization, limit-cycles, the need for scaling in fixed-point realizations, and SNR degradation due to finite word length through both canonical realizations and realizations using cascaded second order sections. The SNR degradation in FFT realizations is also discussed.

3 Laboratory Exercises

A major part of the course is a set of six laboratory exercises which enable the student to gain hands-on experience in implementing real-time DSP systems. Each student is required to complete the labs individually. The platform chosen for the labs is the TI 'C40 DSP processor. Although the 'C30 is less expensive and nearly the same in functionality for single processor systems, the 'C40 has the advantage that it can be used to simulate the fractional fixed-point operations. This feature allows the students to become familiar with both fixed-point and floating-point real-time processing without the necessity of learning two processor architectures and the corresponding development tools.

The lab environment consists of three Sun Microsystems workstations which are each connected by an S-bus card to a 'C40 motherboard from Traquair Data Systems, Inc. which has a single 'C40 TRAM and a stereo A/D–D/A TRAM mounted on it. The 'C40 TRAM has 1 MByte of single wait-state SRAM attached to both the global and local busses of the 'C40. Programs are compiled and debugged on the Sun workstation and downloaded and booted on the 'C40 using software resident on the workstation. The download software was developed at Vanderbilt University and at USU by Ben Abbott, and is called TICK. Additionally, a C software library is available to handle the communication ports and to initialize and use the A/D–D/A hardware. The lab workstations are shared by class members and are available in a secured lab area anytime during the week. A TI 'C40 simulator is used for debugging of the programs. An emulator would be preferred, but the cost of an emulator-based system was prohibitive to us. We have found that the stereo capability of the system is convenient to allow processing of two channels simultaneously, and is used in the first and fourth labs.

The six laboratory exercises are described in the following sections. They build upon each other so that the student acquires new skills with each lab that is completed. Each lab is completed in approximately two weeks.

3.1 Lab 1 - Signal Generation and I/O Using Dedicated Hardware

In this laboratory the students are familiarized with the 'C40 hardware and the software tools used to compile, assemble, debug, download, and boot programs on the processor. To make the lab more interesting, the students are required to write a C program that implements a sinusoidal signal generator, and generate sinusoidal outputs at several frequencies using a 48 kHz sampling rate. The equations for this coupled-form system are given by:

$$\begin{bmatrix} y_c(n) \\ y_s(n) \end{bmatrix} = \begin{bmatrix} \cos\omega_0 & -\sin\omega_0 \\ \sin\omega_0 & \cos\omega_0 \end{bmatrix} \begin{bmatrix} y_c(n-1) \\ y_s(n-1) \end{bmatrix}.$$

This lab also requires the students to become familiar with using the stereo outputs of the D/A so that both the in-phase and quadrature-phase sinusoids can be measured with an oscilloscope. To simplify the lab, the writes to the communication ports are done using calls to a provided C library which uses a polling routine.

The major educational goals of this lab are summarized as follows:

1. Be able to use the C compiler, assembler and linker.

2. Be able to load the linked code into the debugger/simulator.

3. Be able to download the code onto the 'C40 processor and boot the processor.

4. Be able to write a C program that implements a digital sinusoidal signal generator and writes to the communication ports.

5. Analyze the output of the system to determine correct performance (sinusoid, 90° phase difference, correct frequency).

3.2 Lab 2 - FIR Filtering

This laboratory gives the student the opportunity of implementing a 32-tap FIR filter in C code to run in real-time at a 44.1 kHz sampling frequency. The student is required to download specified bandpass FIR ladder filter coefficients to the processor and filter an input signal. The operation of the filter is verified by using a spectrum analyzer to generate a white noise input and measure the spectrum of the output signal.

The realization of the filter must be done with both a ladder and a lattice structure. To realize the lattice, the student must use either MATLAB or the 'C40 to convert the coefficients from the ladder filter.

This lab extends the student's ability by requiring the student to:

1. Realize a FIR filter in both ladder and lattice form, including the fast implementation of a circular buffer in C code.

2. Be able to download coefficients to the DSP to allow for arbitrary filter specifications.

3. Both read and write data to communication ports in real-time.

4. Be able to convert filter coefficients from ladder to lattice realizations.

An example of a program to implement the required filters are given in Figure 1 and Figure 2. Although these programs seem to be quite simple, we have found that ordering of the statements is critical in getting the programs to run on the hardware, which is precisely what the students need to gain experience with.

3.3 Lab 3 - IIR Filtering

This lab marks the point in the class that the students are exposed to the art of linking optimized assembly code to C code. They accomplish this by calling the IIR second order section assembly code provided on the TI ftp site from C code that initializes the filter coefficients and begins the filtering loop. The filter is an eighth order Type I bandpass Chebyshev filter, requiring four second order sections. The filter operation is verified as in the previous lab, using a spectrum analyzer to check the frequency response.

In addition to the typical second order Direct Form II realization, the students are required to implement in optimized assembly, using hardware circular buffering, a lattice-ladder structure realizing the same Chebyshev filter designed above.

The key educational goals of this lab are:

1. Be able to link critical assembly routines to C code.

2. Be able to initialize coefficient and delay arrays and access them from both C and assembly.

3. Be able to locate arrays on proper address boundaries so that hardware circular addressing can be used.

4. Be able to initialize and use hardware circular addressing.

An example of the C and assembly code required to implement the cascaded filter is given in Figure 3 and Figure 4. Note that the assembly portion of the program requires proper placement of the arrays at address boundaries to allow for hardware circular addressing. This is accomplished by using an appropriate linker map.

```
#include <defaults.h>
#include <audio.h>
#define Fs 44100
#define FILTER_LENGTH 32
#define MOD32 0x1f
#define MASK 0xffff

float fromieee(long ha);
void vfromieee(long *ha, float *x, long len);

main(int ac, char **av)
{
    int i,j,start;
    float x[FILTER_LENGTH];
    long int icoeff[FILTER_LENGTH],*pi;
    float filt[FILTER_LENGTH],y;
    long int sample, filt_len;

    /* Get the filter weights */
    for(i=0 ; i<FILTER_LENGTH ; i++){
        icoeff[i] = read_host();
        x[i] = 0.0;
    }

    /* Convert from IEEE 754 to 'C40 */
    vfromieee(icoeff,filt,FILTER_LENGTH);

    /* Initialize the A/D - D/A. */
    init_sampling(Fs);

    /* Start the filtering loop. */

    start=0;
    j = 0;
    start_sampling();
    while(1){

        /* Read the next data sample. */
        sample = *CHANNEL_INPUT_DATA(0);
        sample >>= 16; /* Use RIGHT channel input */
        x[start] = (float)sample;

            /* Filter the point. */
            y = x[start] * filt[0];
            for(i=1 ; i<FILTER_LENGTH ; i++){
                start++;
                start &= MOD32;
                y += filt[i] * x[start];
            }

        /* Write out the sample. */
        sample = (int)y; /* Use LEFT channel output */
        sample <<= 16;
        *CHANNEL_OUTPUT_DATA(0) = sample;
    }
}
```

Figure 1: Example C language program for implementing a ladder FIR filter.

```
#include <audio.h>
#define Fs 44100
#define FILTER_LENGTH 32

float fromieee(long ha);
void vfromieee(long *ha, float *x, long len);

main(int ac, char **av)
{
    int i,j;
    float b[FILTER_LENGTH+1],a[FILTER_LENGTH],K[FILTER_LENGTH+1];
    long int icoeff[FILTER_LENGTH],*pi,sample;
    float bnew,bsave,f;

    /* Get the filter weights */
    for(i=0 ; i<FILTER_LENGTH ; i++){
        icoeff[i] = read_host();
    }

    vfromieee(icoeff,K,FILTER_LENGTH); /* Convert from IEEE 754 to 'C40 */
    init_sampling(Fs);                 /* Initialize the A/D - D/A. */

    for(i=0 ; i<=FILTER_LENGTH ; i++){
        b[i] = 0.0;
    }

            /* Start the filtering loop. */

    start_sampling();
    while(1){

        /* Read the next data sample. */
        sample = *CHANNEL_INPUT_DATA(0);
        sample >>= 16;  /* Use RIGHT channel input */

        f = (float)sample;
        bsave = f;

        /* Filter the point. */
        for(i=1; i<=FILTER_LENGTH ; i++){
            bnew = b[i-1] + K[i] * f;
            f += K[i] * b[i-1];
            b[i-1] = bsave;
            bsave = bnew;
        }

                /* Write out the sample. */

        /* The following statement is necessary to match the direct form
           FIR filter to the output of the lattice.  Note that the
           true output of this filter is the value of y(n-1).*/

        sample = (int)f - sample;
        sample <<= 16; /* Use LEFT channel output */
        *CHANNEL_OUTPUT_DATA(0) = sample;
    }
}
```

Figure 2: Example C language program for implementing a lattice FIR filter.

```
#include <audio.h>
#define Fs 44100
#define NSTAGES 4
void iir_filt(void);
extern float delay[NSTAGES*4];
extern float fcoeffs[NSTAGES*5];
long int nstages = NSTAGES;

main(int ac, char **av)
{
    int i,j;
    long int icoeff[NSTAGES*6],*pi;
    float coeff[NSTAGES*6],b0[NSTAGES],b1[NSTAGES],b2[NSTAGES];
    float a0[NSTAGES],a1[NSTAGES],a2[NSTAGES];

    /* Get the filter weights */
    for(i=0 ; i<NSTAGES*6 ; i++)
        icoeff[i] = read_host();

    /* Convert to 'C40 floating-point. */
    vfromieee(icoeff,coeff,NSTAGES*6);

    /* Put the correct values into the arrays. */
    i = 0;
    for(j=0 ; j<NSTAGES ; j++,i++) b0[j] = coeff[i];
    for(j=0 ; j<NSTAGES ; j++,i++) b1[j] = coeff[i];
    for(j=0 ; j<NSTAGES ; j++,i++) b2[j] = coeff[i];
    for(j=0 ; j<NSTAGES ; j++,i++) a0[j] = coeff[i];
    for(j=0 ; j<NSTAGES ; j++,i++) a1[j] = -coeff[i];
    for(j=0 ; j<NSTAGES ; j++,i++) a2[j] = -coeff[i];
    for(i=0 ; i<NSTAGES*4 ; i++)   delay[i] = 0.0;

    /* Reorder the coefficients into memory properly for the assembly
    routine.  TI subroutine ordering:
    a20,b2,a10,b1,b0,a21,b2,a11,b1,b0,...,a2k,b2,a1k,b1,b0 */

    j=0;
    for(i=0 ; i<NSTAGES ; i++){
        fcoeffs[j++] = a2[i];
        fcoeffs[j++] = b2[i];
        fcoeffs[j++] = a1[i];
        fcoeffs[j++] = b1[i];
        fcoeffs[j++] = b0[i];
    }

    /* Initialize the A/D - D/A. */
    init_sampling(Fs);

    /* Start the filtering loop. */
    start_sampling();
    iir_filt();

}
```

Figure 3: Example C language program for implementing an 8th order IIR filter.

```
    .version    40
FP  .set        AR3

    .globl  _delay
    .globl  _fcoeffs
    .globl  _iir_filt
    .globl  IIR2
    .globl  _nstages

    .text
*****************************************************
* FUNCTION DEF : _iir_filt
*****************************************************
_iir_filt:
    PUSH    FP
    LDI     SP,FP
    LDA     @delay_addr,AR1         ; Starting address of delay storage.
    LDA     @_nstages,IR1           ; Load number of stages.
    SUBI    1,IR1
    MPYI    4,IR1                   ; Rewind for next sample (4*(N-1)).
    LDA     3,BK                    ; Load the circular buffer size.
*
L2:
    LDA     @in_addr,AR0
    LDI     *AR0,R9                 ; Load input sample.
    ASH     -16,R9,R9               ; Sample is from RIGHT channel.
    FLOAT   R9,R2
    LDA     @coeff_addr,AR0         ; filter coefficients in AR0.
    LAJU    IIR2                    ; Zero overhead call.
    LDI     @_nstages,RC            ; Load the repeat counter for
                                    ; nstages biquads.
    SUBI    2,RC
    LDA     4,IR0                   ; Load skip to next biquad storage.
*
    LDF     R0,R9                   ; Move filtered sample.
    FIX     R9
    LSH     16,R9,R9                ; Output is LEFT channel.
    LDA     @out_addr,AR0
    STI     R9,*AR0

    B       L2
*
    .globl  _delay
_delay  .usect "iircoef", 256

    .globl  _fcoeffs
_fcoeffs    .usect "iircoef", 256

*****************************************************
* DEFINE CONSTANTS                                  *
*****************************************************
delay_addr  .word   _delay
coeff_addr  .word   _fcoeffs
in_addr     .word   100041h
out_addr    .word   100042h
            .end
```

Figure 4: Example assembly language routine for implementing the 8th order IIR filter with four 2nd order sections. The routine "IIR2" can be obtained from the TI ftp site.

3.4 Lab 4 - Adaptive FIR Filtering

The purpose of this lab is to teach the student how to use the Interrupt Vector Table (IVT) to do interrupt-based processing. In addition, the student is exposed to one of the powerful applications of DSP – adaptive signal processing.

There is no attempt to develop the theory of LMS adaptive filtering in the class. A brief description of an adaptive noise canceler is presented and then the LMS update equations are given. The students are allowed to use the assembly code available on the TI site, or to write their own assembly routine to implement the filter. The filter must be a 256 tap FIR filter, and the sampling rate is 22.05 kHz.

An interesting part of this lab is the experimental setup used to demonstrate that the adaptation is taking place. Two microphones are used to acquire audio signals from a signal generator. The generator produces a 300 Hz sine wave from one speaker and a 310 Hz square wave from another speaker. The "desired" channel microphone is placed between the speakers to simulate a desired input corrupted with noise. The "reference" channel microphone is placed near the square wave speaker and farther from the sine wave speaker. If the filter is operating properly, the students will observe the spectrum of the "error" output of the filter to initially contain both the 300 Hz signal and the 310 Hz signal (with its corresponding harmonics). After a short time, the unwanted 310 Hz sine wave spectrum will decrease in magnitude as the filter adapts. We typically see about 30 dB of filtering produced.

New skills acquired during this lab include:

1. Ability to write an entire program in assembly.
2. Ability to locate, initialize and use the IVT.
3. Ability to implement interrupt-based processing to process the signals between samples.
4. Ability to use the on-board RAM blocks and the instruction cache to optimize processing speed.

3.5 Lab 5 - Finite Word-Length Effects

This lab teaches the student about the effects of using fixed-point arithmetic, solidifying the theory taught in class lectures, and allows the student to thoroughly understand how to simulate fixed-point arithmetic. To accomplish this, the student is required to implement the same FIR and IIR filters used in Labs 2 and 3. In this lab, however, they must simulate different fractional fixed-point word lengths by rounding the floating-point coefficients to a specified number of bits, and using fractional fixed-point arithmetic in the computations. This can be accomplished due to the MPYSHI instruction on the 'C40, which allows fractional fixed-point multiplies to be simulated by keeping the higher order bits after a multiplication. The student must also apply the theory taught in the classroom regarding scaling to reduce overflow effects.

The students use 16, 12, 8, and 4-bit fractional fixed-point arithmetic with a sampling rate of 44.1 kHz and observe the effects of the different word lengths on the performance of the filter.

The educational goals to this lab are:

1. Be able to simulate fractional fixed-point computations for arbitrary word lengths.
2. Be able to convert floating-point coefficients from a filter design to fixed-point of arbitrary word lengths.
3. Become familiar with the need and procedure for scaling the input samples to avoid overflow.

3.6 Lab 6 - Real-time Spectral Analysis Using the FFT

The final lab in the course allows the student to put together all of the skills learned to this point to perform a significant real-time processing task. Additionally, the student becomes familiar with using one of the most important algorithms in DSP – the FFT.

To complete the lab, the student must display the output of a 1024-point FFT on an oscilloscope, and demonstrate that it represents an estimate of the spectrum of an input signal. A sinusoid is used as the input test signal, and the students observe the "peak" in the spectrum move on the oscilloscope as the input frequency is changed. To simplify the lab, calibration is not required, and the FFT routine can be downloaded from the TI ftp site.

The educational goals of this lab are:

1. Be able to understand and use an assembly-coded FFT routine.
2. Be able to buffer the input signal, block-process the signal, and write out the samples while the next block is being processed.
3. Be able to use bit-reversed indexing.
4. Be able to use the on-board RAM as a "ping-pong" memory to process the blocks.

4 Term Project

The capstone of the course is a term project in which the student uses all of the skills acquired during the laboratory exercises to complete a significant processing task. The student is allowed to choose the algorithm and work on the project during the entire semester. The last 3–5 weeks of the course are reserved for concentrated work on the project by completing class lectures before that time and allowing the students to work on their projects during scheduled lecture time.

The students submit a proposal of a few paragraphs to the instructor by the end of the third week of class, and begin work on the project after approval is obtained. Guidelines for the project include:

- The project should require at least 30 hours to complete, and should take advantage of many of the following capabilities of the 'C40 DSP hardware:
  1. Communication ports.
  2. DMA.
  3. Circular and/or Bit-reversed indexing.
  4. On-board RAM and cache.
  5. Both Global and Local RAM.
  6. Parallel instructions.
  7. IEEE format conversions.
  8. Audio I/O.
- The project should test the capabilities of the 'C40 by requiring that as much processing as possible be performed during each A/D sample time so that the processor does not remain idle for significant periods of time.
- A short report of the project is required. This report should contain a description of the DSP task to be accomplished, results of the processing, and a listing of the code used. At the conclusion of the semester, each student has a chance to demonstrate his project to the class and report on it orally.

We have had many successful projects from students over the five year period the course has been taught. Examples of projects include:

- Simulation of radar return processing to determine range.
- Audio processing to produce sound effects such as delay, echo, musical key shift, surround sound, etc.
- Frequency domain adaptive filtering.
- RLS adaptive filtering.
- Real-time encryption.

5 Discussion and Conclusions

After five years of experience with the course, we are very pleased with the skills the students develop over the semester. Many of the students who have taken the course have reported that they were able to use the background obtained from the course, and even the laboratory notebook from the course, to impress recruiters and obtain employment. We have also received feedback that these skills are in great demand at present in industry.

One of the challenges of the course has always been limiting the demand of the laboratory, lecture, and term project to a reasonable number of hours of work. Students in the course have been highly motivated to complete the assignments because of the interest in DSP they bring into the classroom, but experience has shown that students with inadequate preparation in assembly programming spend more time than is expected to get the programs to run. We continue to search for methods to achieve the educational goals of the course while reducing the demand on the students' time.

In addition, we face a challenge to keep our laboratory exercises up-to-date and relevant to the needs of modern DSP employers. We are currently investigating ways to migrate to more modern, less costly platforms than the 'C40 for the laboratory work, without sacrificing the ability to teach both floating- and fixed-point implementation. One direction we are currently investigating is migration to the TI 'C6x series of processors.

In conclusion, we are convinced that our real-time processing course, ECE 5640, is a significant contribution to our DSP course offerings, and provides useful and up-to-date skills to our graduating seniors and first year graduate students.

References

[1] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals*. IEEE Press, 1997.

[2] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*. Upper Saddle River, New Jersey: Prentice-Hall, third ed., 1996.

[3] Texas Instruments, *TMS320C4x User's Guide*, 1993.