EC581 DSP Projects Laboratory

Policies and Overview
Department of Electrical and Computer Engineering
Rose-Hulman Institute of Technology
Spring Quarter 2003

General Information:

Instructor:	Mark A. Yoder
Office:	C-209
Office Phone:	(812) 877-8291 (If no answer, please leave message on my voice mail.)
Home Phone:	(812) 443-0200 (Call any time before 9 PM)
E-mail:	Mark.A.Yoder@Rose-Hulman.Edu

Required Texts:
"'C6000 Integration Workshop Student Guide", from bookstore.
Good References:
"Communication System Design Using DSP Algorithms", Steven A. Tretter.
"DSP Applications Using C and the TMX320C6x DSK", Rulph Chassaing.
A large number of various Texas Instruments (TI) C62/67 Data Manuals, available as PDF files, can be found on our AFS network at:

G:\ee\yoder\ece581\ECE581 documents

These TI publications are listed below:

spru313.pdf	CODE COMPOSER STUDIO QUICK REFERENCE
spru301.pdf	CODE COMPOSER STUDIO TUTORIAL
spru328.pdf	CODE COMPOSER STUDIO USER'S GUIDE
spru296.pdf	CODE COMPOSER USER'S GUIDE
spru303.pdf	TMS320C6000 DSP/BIOS USER'S GUIDE
spru189.pdf	TMS320C6000 CPU AND INSTRUCTION SET REFERENCE GUIDE
spru190.pdf	TMS320C6000 PERIPHERALS REFERENCE GUIDE
spru198.pdf	TMS320C62X/C67X PROGRAMMER'S GUIDE
spru187.pdf	TMS320C6000 OPTIMIZING C COMPILER USER'S GUIDE
spru186.pdf	TMS320C6000 ASSEMBLY LANGUAGE TOOLS USER'S GUIDE
spru197.pdf	TMS320C6000 TECHNICAL BRIEF
sprs051.pdf	TMS320C6201, TMS320C6201B DIGITAL SIGNAL PROCESSORS
sprs067.pdf	TMX320C6701 FLOATING-POINT DIGITAL SIGNAL PROCESSOR
spdu082.pdf	MICROPROCESSOR DEVELOPMENT SYSTEMS CUSTOMER SUPPORT
spru269.pdf	TMS320C6x Evaluation Module Reference Guide
spru273.pdf	TMS320C6x Peripheral Support Library Programmer's Reference
spru305.pdf	TMS320C6201/6701 Evaluation Module Technical Reference
spra478.pdf	TMS320C6000 EVM Daughterboard Interface
spra567.pdf	TMS320C6000 Multichannel Vocoder Application Design Kit (ADK)
4231.pdf	Multimedia Audio Codec
	Texas Instruments FTP SITE: `ftp://ftp.ti.com/pub/tms320bbs/00welcome.htm`
rtsd workshop.pdf	Real-Time Workshop Course Notes

<u>Important Note</u>: I consider wrong to print more than a few of the most relevant pages from the PDF files below on school-owned printers. Of course, you may print all the pages you want on your own personal printer, using your own paper and toner! In general, I suggest that you get used to reading these .PDF files on your laptop, or on the PCs in the B200 DSP lab. Please save trees.

Course Description:

There are three major goals for this project-oriented laboratory course. The first goal is to supplement the EC580 DSP theory class with practical "hands-on" DSP projects that will, we hope, bring the DSP theory "to life."

The second goal is to acquaint you with the architecture, programming, and hardware interfacing of the TMS320C6x family of special-purpose "DSP chips." A DSP chip is actually a special-purpose microprocessor whose architecture, instruction set, and addressing modes have been specifically designed to support real-time digital signal processing (DSP) applications usually involving signals in the audio frequency range and below (0 - 44 kHz).

Only C language programming projects will be assigned in this class, since TMS320C6x assembly is taught in our Computer Architecture II class, EC331. With today's highly efficient optimizing C compilers, most DSP algorithms can be implemented in C with no more than a 20% loss in performance over an assembly-language implementation.

DSP chips are becoming increasingly popular in a large range of application areas. The cost of DSP chips has fallen dramatically in recent years. Therefore DSP chips are being used in many new "low-end" products, such as voice-commanded toys, talking dolls, instructional systems, consumer electronics, multimedia PC's, machine tools, etc. DSP chips are used in these products to provide such services as audio data compression/decompression, spectrum analysis, noise reduction, audio equalization and companding, real-time servo-mechanism control, speech and music synthesis, and even speech recognition.

The third goal is to acquaint you with various high-level DSP tools, including the ones listed below:

1. MATLAB and its "Signal Processing Toolbox"

MATLAB is a universally-used program, from Mathworks, which, when used with its companion "Signal Processing Toolbox", can perform off-line digital signal processing on data files using the resident processor (in our case, the 400-MHz Intel Pentium II) in the host computer (PC).

2. Hyperception VAB

The Visual Application Builder (VAB) software application is a program (from Hyperception, Inc.) that is capable of simulating algorithms in a graphical user interface "block diagram" environment using a library of software building blocks. Once the application is simulated and run on the PC, it may be compiled into an executable program and downloaded onto the target (C67x) DSP board and run in real time. The VAB allows real-time DSP to be performed on a PC-based DSP/Acquisition board, and then exported to a standard COFF file for use with an embedded DSP, without the need of writing any textual software.

The best way to accomplish these goals is through a course that meets for two lectures a week, and has two regularly-scheduled laboratories that meet each week.

In addition, it is expected that you will need to spend approximately one additional hour (for two credit-hours earned) or approximately five additional hours (for four credit-hours earned) per week in the DSP laboratory.

As long as everyone takes good care of the laboratory equipment, keeps the area clean and neat, and obeys all laboratory rules, the B200 DSP laboratory will be accessible on a ``walk-in" basis throughout the week during the normal Moench Hall operating hours between 8 AM and 11 PM.

During the lectures, we shall discuss TMS320C6x architecture and its programming in C. I shall also make use of this time to introduce the week's lab exercise assignments, and to talk about different practical DSP-related topics. It is expected that, upon entering this class, each student has had exposure to C.

The scheduled laboratory times will be used for lab quizzes, demonstrations of various software tools, and of course, for project demonstration check-offs. Please note that you are expected to do much of the project development work ***outside of*** the regularly-scheduled lab periods!

Computer Usage Policy:

Each two-person EC581 lab team will be assigned a specific computer in B200 for use in this course. It is very important that any problems with equipment in the lab to be reported to me (via Email) as soon as they are noticed.

Although you are welcome to use these computers for other legitimate academic purposes, work for EC581 must naturally take precedence over any other activities. Please refer to the IAIT Institute Policy on legitimate computer use.[1] **You should be particularly aware that this policy requires you NOT to alter the configuration of these machines**, to add unlicensed software to these machines, or to use them in any other way that might be damaging to RHIT.

In anticipation of any potential configuration problems during the quarter, you are strongly encouraged to keep your files for the course on the AFS network, on a personal floppy disk, and also on the C: drive of your assigned machine. Please understand that the management reserves the right at any time to wipe the C: drive clean. For example, if there are software problems during the quarter, it may be necessary to completely reload the software on all machines, which would destroy any files that you leave on the local hard drive.
Consider using CVS[2] to manage your revisions.

Do not attempt to "fix" the autoexec.bat or config.sys on your machine or install any software (except for the assigned programs you write for this course) on the hard drive. While you may be able to correct an immediate problem, you are very likely to create others. If your "fix" is a change that must be made, it should be made by the instructor or the department, so that ALL the machines will be fixed, and not just one. Therefore, if you find that a modification needs to be made to the configuration of a machine, report it to your instructor or Mr. Cottom immediately. Do NOT "do it yourself".

[1] (http://www.rose-hulman.edu/TSC/about_iait/policies/)

[2] http://www.rose-hulman.edu/Class/cs/cs120/install/jdk-install.html and http://www.rose-hulman.edu/Class/cs/cs120/exercises/programming/javaeyes/intro.html

Grading Policy:

For those of you enrolled for two credits, your final grade in this course will be based upon the first of the two items listed below. For those of you enrolled for 4 credits, your grade will be based upon both of the items listed below:

1. **Lab Exercises (Nine weekly project assignments, at 10 points each)**
   Short experiments designed to expose you to various DSP-related topics concerning MATLAB, Hypersignal RIDE 4.2, or TMS320C6x C language programming. Most of these lab exercises are to be completed by two-person teams.

2. **DSP Project (90 points)**
   As with the lab exercises, this project is to be completed by a 2-person team. It may be chosen from the list that appears at the end of this document.

Final Letter Grades:

Final letter grades will be assigned using the following percentage scale:

94 - 100%	A	70 - 77%	C
88 - 93%	B+	66 - 70%	D+
83 - 87%	B	60 - 66%	D
78 - 82%	C+	0 - 60%	F

Be sure to save all graded material that is returned to you, to fully document your grade and to keep track of your cumulative score. Using this percentage scale, you will be able to calculate your grade at any time in the course.

Lab Project Policies

All laboratory project work is to be done in 2-person teams. You are allowed to collaborate fully with your lab partner, but you may not collaborate (beyond discussing general problems and the use of the DSP development software) with any other members of the class!

Project operation must be demonstrated during the next regularly scheduled lab period after which the project was assigned. Remember that you are expected to complete most of the project development work outside of the scheduled lab period.

Each lab exercise as well as the final project must be written as a ***memo-style*** report[3], one report per two-person lab team. The body of your memo will typically be 1 page long, with as many attachments as are needed. Each attachment *must* be labeled, sequentially numbered, and captioned. For example, "Attachment 1. Listing of FIR Butterworth Bandpass Program". Each attachment must be explained and *specifically referred to* in the body of your memo. Your report memo should not be just a collection of your results; it should also contain well-written text that describes the work performed, its purpose, what was done, and an interpretation of your results.

[3] http://www.rose-hulman.edu/Class/ee/HTML/Documents/ECEWritingStandards11-27-00.PDF

Be sure that your report memo contains the following information:

1. HEADING

Project Number	(Enter 1 – 9)
Lab Station Number	(Enter 1 – 15)
Date:	(Date lab was submitted)
To:	(EC581 instructor)
From:	(Printed names *and* initials of both members of the lab team)

2. OBJECTIVES OF THE EXERCISE

A brief description of the exercise and a statement of the lab exercise's objectives - list the ways in which you expect to benefit from doing this experiment!

3. DESCRIPTION OF EXERCISE

Briefly describe the procedures used to solve the assigned problem, including an outline of the experimental procedure followed, any necessary derivations, etc.

4. RESULT

Your results should be discussed in general in the body of your memo, and any detailed algorithm derivations, program listings, graphs, tables, or figures should be listed as attachments. Each attachment MUST be sequentially numbered and labeled ***with an accompanying caption***. Hand-lettered attachment titles and captions are permissible. The caption should be complete enough to offer a "*stand-alone*" explanation of what is contained in that attachment. Finally, each of your attachments *must be referred to at least once* within the text of your report memo. Reference to a specific attachment must be made using capitalized words, for example: "The observed filter frequency response plotted over the theoretically expected frequency response is shown in Attachment 3. . ."

5. CONCLUSION

Summarize what you have learned from the results of the exercise. Comment upon the accuracy of your results, discuss any sources of error if your observed results are not in line with theory, compare theoretical and experimental results, discuss any difficulties encountered, and how they were solved, etc.

6. ATTACHMENTS (as many as needed)

Several attachments will generally be needed for completeness. Your appendices may include such information as detailed algorithm derivations, *well-commented* program listings, graphs, tables, drawings, etc.

7. LAB HANDOUT

The lab handout should be appended to the end of your report memo for the sake of completeness.

Only one written report need be submitted per team. However, I expect each student to retain a copy of each submitted report for his/her own personal records. While lab notebooks will not be graded, I expect each team to keep an informal diary-style notebook that will prove useful when it is time to write the report.

Attendance Policy:

Good class and laboratory attendance is considered mandatory. One-half letter grade will be deducted from your final letter grade for every 3 class or laboratory periods that you miss without a valid pre-excused absence.

All nine of the assigned lab projects must be completed in order for a team to earn a passing grade in the course.

List of Laboratory Exercises[4]

(1) **Code Composer Familiarization, Audio Sampling, Reverberation, Comb Filter, Flanger**
The student is guided through the use of Code Composer Studio to perform editing, compilation, linking, downloading, debugging (single-stepping, setting breakpoints, setting up watch variables, etc.), and execution of a simple C-language program which contains *printf( )* and *scanf( )* functions that perform simple data processing operations. Next, a basic, interrupt-driven, sampling program is introduced. This basic program and its companion interrupt routine will serve as a "template" for many of the following DSP lab projects. The student is then asked to modify this basic sampling template to turn it into an audio reverberation program, and then later into a comb filter. Finally, the comb filter delay may be made continuously variable, to implement an audio flanger.

(2) Floating Point and Fixed Point FIR filter implementation
A MATLAB *M-file* (which calls the "*FIR1*" MATLAB FIR digital filter design function) is used to design and plot the frequency response of 15th order bandstop, bandpass, and highpass filters. Then a real-time, C digital filtering program is written that uses floating point filter coefficients to implement the various FIR filters on the C67 board. The resulting real-time filter is tested using a function generator and an oscilloscope, with the observed results plotted over the frequency response curve predicted by MATLAB. Next, the program is rewritten so that it uses only integer math (prescaling the floating point filter coefficients by multiplying them by a large power of 2 and then truncating them to integer form).

(3) IIR Filter Implementation and Digital Wah-Wah Effect
Use Momentum Software's QEDesign Digital Filter Design program to design an IIR bandpass filter to meet given specifications. Graphically interpret the resulting pole-zero plot (using ruler measurements) to verify the predicted magnitude response. Next, write a C program that implements the IIR filter in "Direct Form II", cascaded 2nd-order biquadratic sections. Experimentally record observed, real-time filter performance using an oscilloscope and a function generator, and plot the experimental measurements over the predicted magnitude response. Finally, implement a series of 10 second-order bandpass digital filters whose passbands span the audio spectrum. Write a program that periodically (every 50 ms) switches filter coefficients, resulting in a filter of continuously varying passbands. Listen to the effect when a low frequency (100-Hz) square wave (spectrally rich) is played through the system. Note the classic "Wah-wah" effect.

[4] I'm adding some TI workshop material during the first few weeks of class, so we'll have to cut back on some of these.

(4) Audio AGC With Silence Threshold (Digital Audio Amplitude Compression)

Write a real-time C-language program for the C67 that stores a 30 ms audio "frame" in a circular buffer. This program must also calculate the average of the absolute values of the audio samples in the frame. Finally, the program must scale each value in the frame by this average magnitude to adjust the average magnitude of the frame to a constant, pre-specified value. Then the resulting adjusted audio frame should be sent out to the loudspeaker, while the next 30 ms frame is being recorded. Silent frames are detected by comparing the average frame magnitude against an experimentally determined "silence threshold" value. If the frame's average magnitude is below this threshold, the frame is zeroed rather than scaled. The program must be written efficiently enough that no (or very few) audio samples are missed between frames. The resulting speech heard in the loudspeaker should sound natural and continuous and be of constant average amplitude, even when the speaker moves several feet away from the microphone.

(5) **Audio VU Meter, with Separate Target and Host C Programs Communicating Through PCI Interface**.

Modify the AGC program from preceding assignment to pass the average frame magnitude (a new one every 30 ms) to a simple companion Microsoft Visual C++ "terminal application" program. Use the C67 EVM board's PCI interface hardware, where the average frame magnitude will be displayed with ASCII graphics. Use the "*dma_*" function call in the C program that runs on the C67 EVM board, and use the "*evm6x_read*" Windows API function in the companion C++ program that runs on the PC. First study the example C67 C program and the companion example C++ PC program. These illustrate the proper method for DMA transfer through the C67 EVM's PCI interface.

(6) **Radix-4 FFT Spectrum Analyzer**

Study the 64-point radix-4 FFT algorithm explained in the handout. Draw the simpler 16-point radix-4 FFT butterfly pattern and indicate the value of each node in the butterfly for the specific test pattern given. Compare the final results with those from MATLAB. (The results should agree.) Next run the C67 radix-4 FFT program that has already been coded using the same test data. Note that it is already set up to calculate the same 16-point FFT. Verify the proper results. Now modify this FFT program for 64 points and test it with two test cases (verified against MATLAB). Now integrate this 64-point FFT routine into your interrupt-driven, sampling program template. In the main program, a global index variable (incremented in the interrupt routine) is used to keep track of when 64 points have been stored. Then the FFT is called, and the frequency corresponding to the position of the peak magnitude value is printed. Proper operation can be verified using a sine-wave function generator whose frequency is slowly varied between 0 Hz and half of the sampling rate.

(7) Real-Time, Narrow Band Noise Reduction via Adaptive Filtering

Implement an LMS adaptive filter, and then pass an audio signal (consisting of speech corrupted by a strong, additive, periodic interference signal) through a delay line. The delayed speech + noise signal is delivered to the reference input, while the non-delayed version of the speech + noise is delivered to the signal input of the LMS adaptive filter. The error signal becomes the de-noised output. First implement this filter using MATLAB and imported WAV files. Then demonstrate significant noise reduction by playing back the processed WAV file. Experiment with the adaption coefficient value and the necessary "decorrelation delay time". Finally, convert your MATLAB program to a real time C67 program.

(8) **Use of Hyperception VAB to Perform Digital Filtering and Audio Scrambling/Descrambling** Study and run the various block diagram DSP examples. Also study the instructions for using Hyperception's companion "Hypersignal Digital Filter Design Program". Now construct your own block diagram system that uses the concept of mixing and filtering to invert an audio spectrum. Cascade two of these spectral inversion systems to realize a simple audio scrambler and unscrambler. Demonstrate by first playing the scrambled audio from the first spectral inverter and then playing the unscrambled audio from the second spectral inverter.

(9) DSP Scavenger Hunt
Use Hyperception RIDE 4.2 digital filter blocks and/or your adaptive filter program to remove noise from a series of noise-polluted digital audio (WAV file) clips which indicate the location of various $5.00 bills hidden around campus. Since the nature of the noise varies from one clip to another, you will have to apply different filters to each clip. You may want to observe the spectrum of various frames within the clip before deciding how it should be filtered.

List of Term Projects (For those taking the course for 4 Credits)

By the end of the fifth week, you will be expected to pick one term project from this list. You are then expected to implement the project using either TMS320C6x C and/or assembly language. You are expected to work on the project in teams of two. Note that the assigned weekly lab projects get somewhat easier in the last 4 weeks of the class, to allow more time for work on the term project!

1. Chirp Sonar: Real-time Distance Measurement Using Acoustic Pulse Delay

It is possible to measure the distance between a speaker and a microphone, given the speed of sound in air. Your project must operate in real time on the DSP board. To make this system work well in the presence of other noise, a frequency-swept audio tone burst pulse (audio chirp) should be used along with cross-correlation.

2. Touch-Tone (DTMF) Telephone Monitor

A telephone line monitoring system is to be designed that can be connected to the phone line in order to determine the telephone number that is dialed form a "touch tone" (DTMF) keypad. This system might be used to keep track of toll calls made in a fraternity house.

3. Musical Instrument (Guitar) Tuner

Your project must operate in real-time, providing feedback to the user on how far the played note is from the closest one of the six tuning notes: Low E A D G B High E. This project involves real-time display on the PC screen. You will have to write two programs, one that runs on the DSP board (in assembly or C) and one that runs on the PC (in C) that manages the real-time result display on the PC monitor. You may have to experiment with several different approaches to measure the note frequency. I suggest you start by investigating an FFT approach as well as a band-pass filtering / zero crossing counting approach.

4. Real-Time Audio Spectrum Analyzer

You have probably seen these in an audio store, where 6 - 12 bar displays (that represent logarithmically-spaced frequency bands within the audio spectrum) move up and down with the music in "real time", yielding a frequently-updated view of the spectral magnitude content of the audio signal. Just as with the guitar tuner project, because this project requires the use of the PC display, two communicating programs will have to be written.

5. Underwater Ultrasonic Receiver / Transmitter

Develop an DSP transmitter that can modulate a 300 Hz – 3 kHz audio speech waveform onto a 20 kHz (ultrasonic) "carrier" wave. This waveform can be recorded on audio tape. Then show that the signal can be played back into an AM receiver that can recover the audio information. Such a system is currently being used to provide underwater communications between divers.

6. Voice-Operated Security Lock

This project will play a 440 Hz (middle A) tone for 1 second when the lock entry button is pressed by person seeking to gain admittance. Then the person who wants admittance must sing that note "Ah...." for one second. The DSP board will analyze the note that is sung and, based upon ratios of the harmonics to the fundamental, the DSP software will decide if that person is to be allowed admittance to the room.

7. Real-time Adaptive Noise Filter

Implement the MATLAB algorithm developed in Lab Project 7 in "real time" using the C67 DSP chip. Demonstrate that it can improve intelligibility of a voice recording that has been subjected to additive narrow-band noise.

8. Audio Effects Program

Integrate Reverb, Flanger, Distortion, Frequency shifter, Tremolo, etc. audio effects with an interactive graphical interface using the PC keyboard and monitor.

9. Midi Synthesizer

Take the MIDI stream from the PC host and send it to the DSK. Have the DSK synthesize the music.

10. Midi Interface

Use the DSK as a MIDI interface to the Rogers Organ in my living room. See me for details.

Many thanks to Keith Hoover who did the original version of this handout. I've only made a few tweaks to it.

--Mark A. Yoder