A LABORATORY-BASED COURSE IN REAL-TIME DIGITAL SIGNAL PROCESSING IMPLEMENTATION

Scott E. Budge

Dept. of Electrical & Computer Engineering
Utah State University, Logan, UT 84322-4120
scott.budge@ece.usu.edu

ABSTRACT

A four credit-hour laboratory course in real-time digital signal processing (DSP) implementation is described. This course has been developed and taught at Utah State University for over 12 years, incorporating feedback from students and extensive classroom experience. 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 seven laboratories in which the student must perform real-time processing on hardware based on modern digital signal processors and FPGAs. Student feedback has indicated that the course has been very successful in helping them become effective in real-time DSP-related jobs.

Index Terms— education, DSP, laboratory, real-time

1. INTRODUCTION

Feedback from industry leaders developing products and applications for digital signal and image processing has indicated a need for electrical engineers with experience in real-time systems. This need includes expertise in programming real-time digital signal processors and field programmable gate arrays (FPGAs), and in understanding the effects of practical implementations on the signal processing algorithm performance.

This paper describes a four credit hour course in real-time digital signal processing (DSP) which has been developed and taught at Utah State University for over 12 years. 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 seven laboratories in which the student must perform real-time processing on hardware based on a modern digital signal processor (the TMS320C6713) and FPGA (the Xilinx Virtex II).

The course, ECE 5640 - Real-Time Processors, is divided into three main sections, or modules. The first module is intended to familiarize the student with the general architectures of DSP integrated circuits and DSP cores available on the market today. The second module of the course concentrates on an overview of FPGA architectural features and includes a tutorial on the FPGA programming language Handel-C, which is a parallel, hardware-based extension of the C language. The last module of the course concentrates on the theory of DSP system design and the effects of finite word-length on processing performance.

The laboratory portion of the class is based on the TI 'C6713 Developer's Starter Kit (DSK) and a Vertex II developer's board from Celoxica. The 'C67xx was selected over other possible processors because of the capability of the processor to simulate fractional fixed-point arithmetic so that both floating-point and fixed-point processing can be implemented.

The remainder of the paper is structured as follows. In Section 2, each of the three modules of the course curriculum is discussed. This is followed in Section 3 by a brief description of the laboratory exercises performed by the students. Finally, Section 4 contains a discussion of the educational aspects and conclusions drawn from the course.

2. COURSE CURRICULUM

The three course sections, or modules, in the one semester course of 15 weeks are divided into seven to eight weeks on the architectural features of DSP processors, followed by three weeks on FPGA architectures and programming using Handel-C, and concluded with a four week module on the theoretical aspects of implementation and computational performance.

2.1. DSP Architectures

The first module in the course covers the general architecture of single-core DSP processors. The material is presented from Lapsley's general tutorial overview of DSP processor families[1]. Although the text is fairly out of date, the architectural features presented are applicable to many of the processor families in use today. The material is presented with an emphasis on the features that were pioneered in DSP processors or that are particularly applicable to DSP algorithms.

To introduce these features, the computations, memory accesses, and looping overhead required by a simple FIR filter are enumerated. A comparison is then drawn between execution on a general purpose processor and a DSP chip. From this discussion, the following architectural features are covered in detail:

- Fixed- and floating-point data paths, including the advantage of a multiply-and-accumulate unit (MAC).
- Memory architectures, including the Harvard and Modified Harvard architecture. Also included in this topic are different cache designs, including single and multiple sector caches, direct-mapped and set-associative caches, and instruction and data caches.
- Addressing modes, including the hardware supported circular and bit-reversed indexing modes.
- Execution control, including hardware loop control, interrupts, stacks, and zero-overhead branching.
- On-chip peripherals.
- Pipelining.
- Power management strategies.

Also included in this module is a review of the corresponding architectural features of the TI TMS320C6713 processor from the TI documentation[2, 3]. This is presented in parallel with the material from Lapsley so that the students see a concrete example of these features in the DSP processor they will be using in their laboratory work. The additional features of the Very Large Instruction Word (VLIW) architecture of the 'C6713 are presented as an example of the multiple parallel execution unit processors that are becoming common in the market.

Finally, examples of special DSP architectures are presented, including FFT processors and high throughput FIR filter processors.

2.2. FPGA Processing for use in DSP Applications

The second module in the class covers the general architecture and use of FPGAs for DSP applications. The basic concept of an FPGA is covered, followed by the advantages and disadvantages of FPGAs over DSP processors, and then the trends in FPGA design are discussed. These include the move to larger and larger resources embedded within the FPGA fabric, such as block memory, large LUTs, fast I/O support, multiple clock domains, multipliers, and even MAC units. The example of a Xilinx Virtex II is used to illustrate these concepts and, like the TI DSP used earlier in the course, has been adopted for use in the FPGA laboratory exercise.

One of the major difficulties with the goal of a hands-on real-time laboratory experience with FPGAs is that the students must be able to implement a DSP algorithm on the FPGA. This can be done by requiring either a lab exercise that uses provided configuration files (bitfiles) or to expect the students to have a skills in a hardware design language (HDL). We have chosen a third option: have the students use a C-like language to write an image processing algorithm that exploits the parallelism of an FPGA. The advantage of this approach is that the students can learn to write a parallel program using a quickly-learned language based on the C they have already learned for pre-requisite classes.

The students are thus given a short tutorial on the Handel-C parallel programming language for FPGAs. The tutorial covers the basics of the Handel-C parallel extensions to C, with an emphasis on the concepts of both fine- and course-grained parallel implementations. The difference between pipelined processes and parallel processes are also discussed. Our experience is that students are quickly able to master the Handel-C design of a pipelined image processing application, as discussed in Section 3.6.

The Handel-C tutorial is based on an overhead slide presentation provided by Celoxica and adapted for our use. Interested students are provided the materials to learn more sophisticated design methods using Handel-C if they so desire.

2.3. DSP System Design and Performance

In the final weeks of the course, the performance aspects of DSP system design are discussed. This module of the course is independent of the architecture or the implementation hardware, except for the obvious differences in fixed- and floating-point realizations. This module differs from the previous two modules in the course in that it is a much more theoretical study of system and signal processing performance. The material for this module comes primarily from Chapter 9 in Proakis and Chapter 13 in Ifeachor[4, 5]. Topics included in this module are:

- System realizations, such as Direct Form I and II, linear phase FIR, cascaded and parallel, lattice, and lattice-ladder. The number of computations and memory requirements of each realization are compared.
- Number representations, including IEEE 754 floating-point and two's-compliment fixed-point. Both fractional and integer fixed-point representations are discussed, including the effects of rounding, truncation and multiplication.
- Finite word-length effects in fixed-point realizations. We start with coefficient quantization and move to computational effects such as limit-cycles, the need and implications of scaling, and SNR degradation. These effects are compared in realizations using canonical and cascaded second-order sections.

3. LABORATORY EXERCISES

A major element of the course is the set of laboratory exercises that enable the student to implement DSP algorithms in real-time hardware. The exercises are hands-on and are completed individually by each student. The goal of the exercises is to help the student learn debugging strategies for real-time systems using modern development tools, exploitation of features available in a DSP or FPGA platform, and how to transfer a mathematical algorithm into a programming language. Since both high-level languages (C/C++) and assembly are commonly used in industry development, the students are introduced to the lab hardware by initially programming in C, and then move to mixed C/assembly, and finally code entirely in assembly language.

The lab environment consists of the TI TMS320C6713 DSK and the Celoxica DK Suite used with an RC200 FPGA board. Six of the lab exercises are done using the DSK, and one is completed on the RC 200. Additional support hardware includes a National Instruments multi-channel soft oscilloscope/spectrum analyzer (NI PCI-6259) with LabView and a video camera. The processor hardware kits are checked out by the students, and the lab equipment is available on a 24/7 basis.

The seven laboratory exercises are described in the following sections. With the exception of the FPGA lab (Lab 6), later labs build on the skills learned during previous labs.

3.1. Lab 1 – Signal generation and I/O using dedicated hardware

In this laboratory, the students are familiarized with the TI DSK software tools (Code Composer Studio) 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 24 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 processor Multichannel Buffered Serial Port (McBSP) are done using calls to a provided C library which uses a ping-pong buffering method.

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

1. Be able to use the C compiler, assembler, linker, and debugger.
2. Be able to write a C program that implements a digital sinusoidal signal generator and writes to D/A converter.
3. 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 48 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 'C6713 to compute 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 the McBSP in real-time.
4. Be able to convert filter coefficients from ladder to lattice realizations.

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 IIR second order section (biquad) assembly code 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 biquads. The filter operation is verified as in the previous lab, using a spectrum analyzer to check the frequency response.

The assembly routine implementing the biquad is done in TI "linear assembly." Since the purpose of the class is not to teach assembly programming requiring an in-depth knowledge of the processor pipeline, linear assembly allows the student to program in an assembly-like language that executes at assembly speeds for well-optimized code.

In addition to the typical second order Direct Form II realization, the students are required to implement a lattice-ladder structure in linear assembly 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 pass parameters and output samples to and from C calling routines in registers.

4. Be able to convert filter coefficients from Direct Form I to lattice-ladder realizations.

5. Be able to use the optimization settings in both the C compiler and the linear assembler to increase processing speeds.

3.4. Lab 4 - Adaptive FIR filtering

The purpose of this lab is to teach the student how to use hardware circular buffering to increase the processing speed of a FIR filter. 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 canceller is presented and then the LMS update equations are given. The filter must be a 256 tap FIR filter, and the sampling rate is 48 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 and speakers. 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 (other than DSK board initialization) in assembly.

2. Ability to locate arrays on proper address boundaries so that hardware circular addressing can be used.

3. Ability to place the arrays in on-board RAM to increase performance.

4. Ability to initialize and use hardware circular addressing.

An example of student-written TI 'C6713 "linear assembly" code for the FIR filtering portion of an LMS adaptive filter is given in Figure 1. Note that the code segment illustrates the use of the `AMR` register for circular addressing, and `lddw` double-word loads to increase I/O throughput.

3.5. Lab 5 – Real-time spectral analysis using the FFT

The next 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 block-process the signal and write out the samples while the next block is being processed.

3. Be able to use bit-reversed indexing.

3.6. Lab 6 – Introduction to FPGA processing

At the completion of the previous lab, the lecture portion of the course has progressed through the introduction to Handel-C, and the students are prepared to write a "macro procedure" which implements, using an FPGA, a parallel-pipelined image processing algorithm. The algorithm chosen is a Sobel line enhancement filter. Successful implementation of the pipeline produces a processing rate of one pixel per clock at a minimum of 25 Mpixels/s.

Most of the code which sets up the sliding Sobel kernel in the image buffer is provided to the student. This frees the student to concentrate on the concept of how to do pipelining in an FPGA.

When the student is successful, he/she is able to connect a video camera and monitor to the RC 200 board and observe the output of the image line enhancer in real-time.

The educational outcomes of this lab are:

1. Understanding of the advantages of parallel hardware in an FPGA for improving processing speed.

2. Understand the strengths and weaknesses of DSP processor vs. FPGA implementations for DSP.

3. Be able to understand and debug a parallel-pipelined program.

```
    .title  "adfilt.sa"
    .def    _adfilt
    .sect   ".text"
    .global _adfilt
    .global _eb

_adfilt .proc A4,B4,B3
        .reg inp,d,y,h2:h1,x2:x1,tmp,beta,e,cnt,x0,err
        .reserve B6,B7

    mv      A4,inp          ; Input
    mv      B4,d            ; Desired

    mvk     0x5000,B0       ; set bit 12 & 14
                               ; for regs B6,B7

    mvklh   0x0007,B0       ; set BK0 field to 9
                               ; (blocksize=1024) 4=>32

    mvc     B0,AMR          ; set the AMR reg

    mvk     0,y             ; Init y=0
    mvk     64,cnt          ; Init cnt=256

    mvkl    _eb,err         ; set A7 to storage ptr
    mvkh    _eb,err           ; set A7 to storage ptr
    ldw     *+err[1],beta ; Load beta
    ldw     *+err[2],B6
    ldw     *+err[3],B7
    stw     inp,*--B6       ; move input to mem

loop: .trip 128
    ldw     *B6++,x1        ; Load hist[n],hist[n-1]
    ldw     *B6++,x2

    lddw    *B7++,h2:h1

    mpysp   x1,h1,tmp       ; x[n]*h[n]
    addsp   tmp,y,y         ; y+=x[n]*h[n]

    mpysp   x2,h2,tmp       ; x[n-1]*h[n-1]
    addsp   tmp,y,y         ; y+=x[n-1]*h[n-1]

    sub     cnt,2,cnt       ; cnt--

[cnt]    b      loop

    subsp   d,y,e           ; e=d-y
    stw     e,*err          ; store e to mem

    mvk     64,cnt
    mpysp   e,beta,e        ; e=BETA*e

                    (code for updating filter...)



    stw     B6,*+err[2]     ; store updated pntr

    mv      y,A4            ; move output into A4

    .endproc A4,B4,B3

    b       B3              ;return to address stored in B3
    nop     5
```

Fig. 1. Linear assembly code segment for implementing the FIR portion of an LMS adaptive filter. Note the use of hardware circular buffering (AMR register) and double-word loads (lddw).

3.7. Lab 7 – Finite word-length effects

The final lab in this course 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 an IIR filter similar to the one used in Lab 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 MPYH instruction on the 'C6713, 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 48 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.
4. Understand the effects of word-length on the frequency response of an IIR filter.
5. Understand the advantage of cascaded biquad filters over Direct Form II for large-order filters.

4. DISCUSSION AND CONCLUSIONS

After 12 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 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. This has led to a continual updating of the lab write-up to add new hints and helps.

In addition, we face a challenge to keep our laboratory exercises up-to-date and relevant to the needs of modern DSP employers. This leads to the need to select modern hardware without sacrificing the ability to teach both floating- and fixed-point implementation.

Finally, feedback from the students indicates that the addition of the FPGA experience in the lab is very exciting. They typically rate the FPGA lab as one of the most enjoyable. We are currently investigating how we can add another lab using the FPGA hardware and Handel-C without overloading the lab portion of the course as mentioned above.

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.

5. REFERENCES

[1] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals*. New York, NY, 10017-2394: IEEE Press, 1997.

[2] *TMS320C67x/C67x+ DSP CPU and Instruction Set Reference Guide*, Texas Instruments, Nov. 2006.

[3] *TMS320C6000 DSP Peripherals Overview Reference Guide*, Texas Instruments, Dec. 2007.

[4] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 4th ed. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2007.

[5] E. C. Ifeachor and B. W. Jervis, *Digital Signal Processing: A Practical Approach*, 2nd ed. Pearson Education Limited, 2002.