8-1-2005

A framework for teaching real–time digital signal processing with field–programmable gate arrays

Tyson S. Hall
Southern Adventist University, tyson@southern.edu

David V. Anderson
Georgia Institute of Technology - Main Campus, dva@ece.gatech.edu

A Framework for Teaching Real-Time Digital Signal Processing With Field-Programmable Gate Arrays

Tyson S. Hall, *Member, IEEE,* and David V. Anderson, *Senior Member, IEEE*

***Abstract*—Many curricula include separate classes in both digital signal processing (DSP) theory and very high-speed integtrated circuit hardware description language (VHDL) modeling; however, there are few opportunities given to students to combine these two skills into a working knowledge of DSP hardware design. A pedagogical framework has been developed whereby students can leverage their previous knowledge of DSP theory and VHDL hardware design techniques to design, simulate, synthesize, and test digital signal processing systems. The synthesized hardware is implemented on field-programmable gate arrays (FPGAs), which provide a fast and cost-effective way of prototyping hardware systems in a laboratory environment. This framework allows students to expand their previous knowledge into a more complete understanding of the entire design process from specification and simulation through synthesis and verification. Students are exposed to different aspects of signal processing design, including finite precision, parallel implementation, and implementation cost tradeoffs.**

***Index Terms*—Digital signal processing (DSP), DSP chip design, DSP hardware, field-programmable gate array (FPGA), fixed-point hardware.**

I. Motivation

THE digital signal processing (DSP) market and industry are undergoing a slow, but steady transformation. More and more advanced signal processing systems are finding their way into embedded systems. These systems typically have limited processing resources and stringent power limitations. Often DSP systems for these devices must be implemented on fixed-point processors and a small amount of custom or reconfigurable hardware. The addition of custom hardware requires DSP engineers to leave the comfort of floating-point arithmetic, highly optimized DSP processors, and even MATLAB and enter the world of DSP hardware design.

Unfortunately, students graduating from most DSP programs are left unequipped to deal with the challenges of DSP hardware design and hardware/software co-design. Many curricula include separate classes in both DSP theory and hardware modeling; however, there are few opportunities given to students to combine these two skills into a working knowledge of DSP hardware design [1], [2]. Students often struggle to bridge this gap between the theory and the hardware implementation of DSP systems [3]. This paper presents a pedagogical framework whereby students can leverage their previous knowledge of DSP theory and very high-speed integtrated circuit hardware description language (VHDL) hardware design techniques to design, simulate, synthesize, and test digital signal processing systems [4]. In addition, a course is described that uses this framework to teach DSP hardware design.

TABLE I
Real-Time DSP Technologies and Their Pedagogical Usefulness

Function	FPGA	DSP μP	MATLAB
Fixed–point Arithmetic/Operations	•	•	○
Floating–point Arithmetic/Operations	○	○/•[1]	•
Filter/Transform Implementations	○	•	•
Explore Design Trade–offs *(power, area, complexity, throughput, etc.)*	•	○	–
HW/SW Trade–offs	•	–	–
Parallel Processing	•	–	–
Data Flow/Buffering	•	•	–
Peripheral I/O	•	○	–
System Analysis	○[2]	○[2]	•

[1] There are both fixed–point and floating–point DSP microprocessors.

[2] System analysis/testing is possible with third–party MATLAB interfaces.

Key:

– = No or very limited support

○ = Possible

• = Efficient, well–suited to technology

Many courses and textbooks have been developed for teaching real-time DSP concepts using DSP microprocessors [5]–[12]. The program presented herein is not designed to replace or displace DSP microprocessor-based programs; the goals of the program are different. Programs that rely on DSP microprocessors as their primary implementation medium tend to emphasize software programming rather than hardware design. By using field-programmable gate arrays (FPGAs) as the core technology, students are given the opportunity to design custom hardware implementations [13] and investigate concepts, such as massively parallel algorithms. In addition, FPGAs can synthesize microprocessor cores, allowing students to investigate the tradeoffs between hardware and software implementations.

As illustrated in Table I, FPGAs provide a very versatile platform for teaching real-time DSP implementations. While they are not optimal for every function, FPGAs do provide the most flexibility, which proves invaluable in a classroom environment.

Manuscript received March 16, 2004; revised May 25, 2005.

T. S. Hall is with the School of Computing, Southern Adventist University, Collegedale, TN 37315-0370 USA (e-mail: tyson@southern.edu).

D. V. Anderson is with the Georgia Institute of Technology, Atlanta, GA 30332-0250 USA (e-mail: dva@ece.gatech.edu).

Digital Object Identifier 10.1109/TE.2005.853069

Students can use a single system to explore hardware and software designs and to make various design tradeoffs to optimize their systems for power, area/size, complexity, throughput, latency, etc.

II. Course Logistics

This course is a senior-level technical elective sponsored by the Digital Signal Processing technical interest group within the School of Electrical and Computer Engineering (ECE) at the Georgia Institute of Technology (Georgia Tech), Atlanta, and it is denoted ECE 4273 DSP Chip Design. ECE 4273 is a three semester-hour course with two hours of lecture and a three-hour laboratory session each week. The lecture and laboratory material are closely coordinated so that topics covered in lecture are reinforced in a hands-on laboratory experiment within, at most, a two-week time period.

Encouraging the interaction between students specializing in DSP and those majoring in computer engineering is one of the goals of ECE 4273, and this course is designed for students from both backgrounds. Ideally, an even mix of students from each major will be enrolled, and laboratory project teams can be formed by combining students with different specialties. However, in practice, the majority of students enrolling in this course at Georgia Tech have been DSP students.

Undergraduate students taking this course are expected to be familiar with MATLAB, digital filter design, basic transforms, FPGAs, and VHDL. These assumptions are valid given the enforced prerequisites (a senior-level fundamentals of DSP course) and the required core curriculum for electrical and computer engineering majors at Georgia Tech, which includes laboratories and classroom lecture on FPGAs and VHDL in the digital design and computer architecture sequences [14], [15].

ECE 4273 has also proved to be popular among first-year DSP and computer engineering graduate students. To accommodate those students who have not had the prerequisites, laboratory assignments and tutorials on the computer-aided design (CAD) tool software, FPGAs, VHDL, and MATLAB are provided during the first few weeks of the course. Many of the undergraduate students have also found these resources useful to refresh their understanding of the basic concepts.

ECE 4273 has been taught with this course format for two semesters; the first semester was fall 2002. The high-level objective for this course is to familiarize students with hardware integrated circuit (IC) implementations of DSP algorithms. More specifically, at the end of this course, students are expected to be able to synthesize digital logic and fixed-point signal processing systems using VHDL, be able to design hardware filters using distributed arithmetic, to optimize a hardware filter given realistic design constraints using a variety of filter design techniques, and be familiar with parallel processing structures, classic DSP microprocessor architectures, and hardware/software co-design. At the completion of this course, students filled out a course survey. With such a brief history, substantial assessment data is not available; however, Table II contains a summary of the pertinent student evaluation data gathered to date.

After the first offering of this course, the student survey showed that the students felt the course objectives had been adequately addressed; however, they ranked course organization and planning as the weakest component of the course. To address this issue, the instructors made three primary improvements to the course. First, new computers, FPGA boards, and CAD tools were purchased for the laboratory. This upgraded equipment increased student efficiency in the laboratory by making compilation times faster and by providing FPGA boards and tools that were more similar to those tools that students were using in other courses. Second, lectures were supplemented with more readings and handouts to provide additional sources from which students could learn. Finally, laboratory assignments were reorganized to balance the amount of time each project required. From the survey data in Table II, these improvements were rated positively by the students. Not only did the students' perception of the course organization and planning increase, but they also felt that the course better met the course objectives.

TABLE II
Student Evaluation Data: Interpolated Medians

Description	Fall 2002[2]	Fall 2003[3]
Course Organization & Planning	3.9	4.5
Covered Course Objectives	4.2	4.8
Exams Covered Course Objectives	4.6	5.0
Learned Current Problems in DSP Implementations	4.0	4.8
Opportunity to Assess Design (Eng. Process)	4.9	5.0
Built on Previous Course Work	4.6	4.8

[1] Scores are based on a scale of 1 to 5 with 1 representing "Strongly disagree " and 5 representing "Strongly agree."

[2] The survey was completed by 7 out of 12 students.

[3] The survey was completed by 4 out of 11 students.

In addition, this course has been found to introduce students to new opportunities for research. Graduate students who have taken this course and stayed at Georgia Tech (thus making their progress easier to track) have successfully integrated this material into their research projects. Their research has expanded to include applied experiments and hardware implementations of their innovations. At least three published papers [16]–[18] and two patent disclosures can be directly attributed to their participation in ECE 4273.

III. Topical Coverage

Since the nature of this course is a convergence of DSP and computer engineering (CMPE), lecture material is pulled from both of these disciplines. The course schedule typically consists of one week of lectures on DSP theory, optimization techniques, etc., followed by one week of implementation-related lectures. The laboratory projects then provide students with an opportunity to combine these two subjects into a working knowledge of DSP hardware design.

A. DSP Material Covered

The early lectures on DSP are designed to introduce students to the basic concepts needed in the rest of the course. Subjects

covered include an introduction to real-time DSP systems, a discussion and comparison of different number formats, and an in-depth look at fixed-point arithmetic and the effects of quantization. The lectures during midsemester focus on filtering as one of the core DSP building blocks found in signal processing systems. These topics include finite-impulse response (FIR) filter structures, filter design optimizations, the canonical signed digit format, sums-of-powers-of-two optimizations, subexpression sharing, and infinite-impulse response (IIR) filters. Toward the end of the semester, topics are varied from semester to semester to emphasize the material needed for the final capstone laboratory project. In the past, the last few weeks of lectures have covered topics, such as floating-point arithmetic, adaptive filtering, DSP microcontroller architectures, two-dimensional (2-D) filtering, and hardware/software co-design.

B. CMPE Material Covered

As discussed previously, the class census has been dominated by students specializing in DSP. For this reason, lectures on FPGAs, VHDL, and hardware implementations have had to start from a very basic point and progress quickly to the DSP implementation issues that this course should address. Thus, the early lectures focus on introducing and refreshing students on these subjects. Topics covered include basic FPGA concepts and architectures, introduction to VHDL for synthesis, and a review of state machine design with an emphasis on VHDL implementations. Lectures during midsemester correspond with the DSP material being covered. Subjects covered include FIR filter implementations, hardware versus software fixed-point multipliers, and distributed arithmetic architectures. The semester ends with lectures on soft-core microprocessors, hardware implementation of a floating-point arithmetic unit, and implementations of basic adaptive-filter structures. In addition, commercial DSP microprocessor architectures are investigated and used in discussions of implementation costs and tradeoffs.

C. Laboratory Projects

The laboratory projects are completed by students in teams of two. The semester starts with tutorials and basic projects introducing the CAD tools (Altera's Quartus II software), VHDL, and MATLAB. Projects are then assigned that explore basic implementation concepts, such as buffering, fixed-point arithmetic, and quantization effects through the design and synthesis of low-order FIR filters. Later laboratory assignments cover subjects, such as IIR filters, filter optimization techniques, and "multiplierless" implementations (e.g., distributed arithmetic). Most laboratories are designed to take one three-hour laboratory period to complete. However, this time constraint requires some of the subjects (such as FIR filtering) to be broken up into multiple units with each unit being addressed in a laboratory period.

Each laboratory has multiple components. A prelaboratory exercise is given to the students a week in advance, and it includes a problem analysis or theoretical study that ties the lectures to the laboratory assignment. During laboratories, students implement a hardware design in VHDL for their DSP system. Finally, the students test their design and analyze its performance relative to expected performance using the MATLAB–FPGA interface.

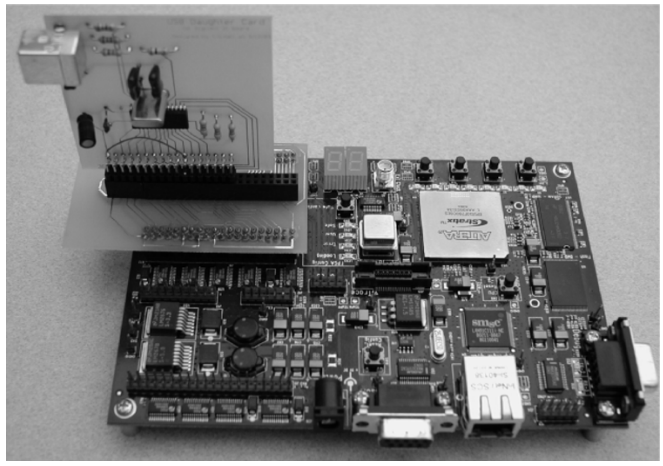


Fig. 1. FPGA board that comes with the Altera Nios Development Kit–Stratix Edition is shown with a custom USB daughter card attached to it. The USB daughter card is built around the Phillips PDIUSBD12 chip with implements the physical-level interface of the USB protocol.

For the final capstone laboratory project, students are allowed to form larger teams (up to four members), and they are given four weeks to complete their projects. Students are given the choice of proposing their own design project or implementing the provided one. In either case, the final project is required to include a soft-core processor synthesized on the FPGA and custom DSP module (such as a floating-point unit or optimized hardware fast Fourier transform module). These requirements allow the students to investigate hardware/software design tradeoffs and attempt to achieve an optimal partitioning of their system.

IV. Lab Infrastructure

Having a straightforward prototyping system is very important to the success of this course. This section describes a platform designed by the authors to facilitate the experiments done by students in the laboratory exercises.

At the highest level, an environment is needed that contains both theoretical analysis and hardware implementation capabilities. Currently, these two environments exist separately in the forms of the MATLAB software and commercial FPGAs. However, a convenient interface does not exist between these two environments that meets the needs of the course presented herein. (While there are commercial interface programs for high-level programming of FPGA devices, these tend to hide many of the implementation details that this course is trying to teach.) To alleviate this problem, a plug-and-play Universal Serial Bus (USB) connection was created between the PC and the FPGA [19].

The FPGA board selected for this course is a development board (Nios Development Kit–Stratix Edition) provided by the Altera Corporation. A custom daughter card was then developed for this board to provide the USB capabilities. Fig. 1 shows the Altera Nios Development board with the custom USB daughter card attached.

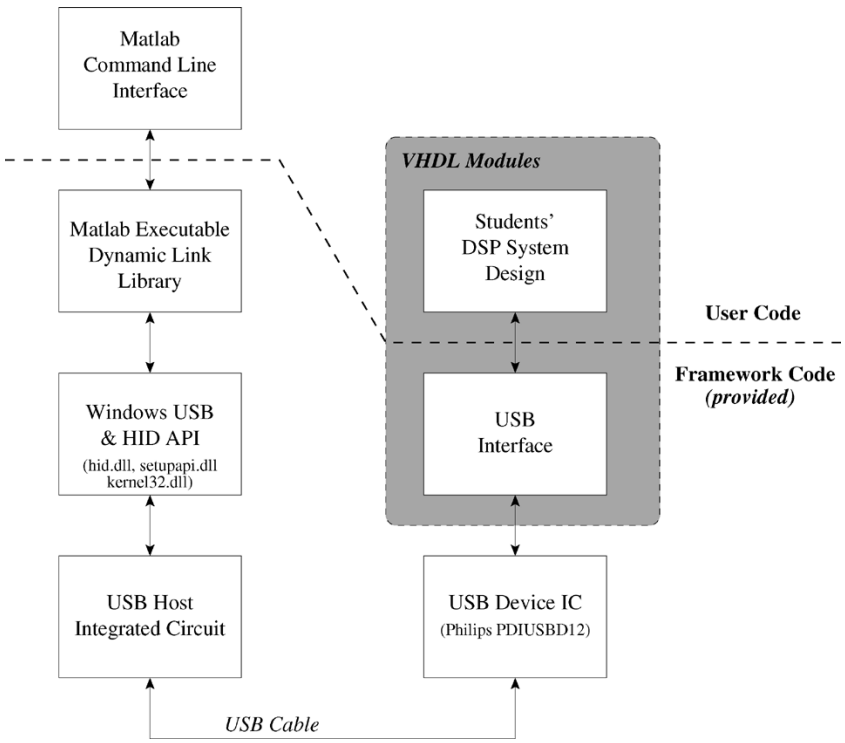


Fig. 2. This diagram illustrates the data path for the MATLAB–FPGA interface. On the PC side, only the MATLAB executable module had to be created since most of the USB protocol is already implemented in Windows. On the FPGA side, however, much of the USB and HID protocols had to be implemented in synthesized hardware on the FPGA.

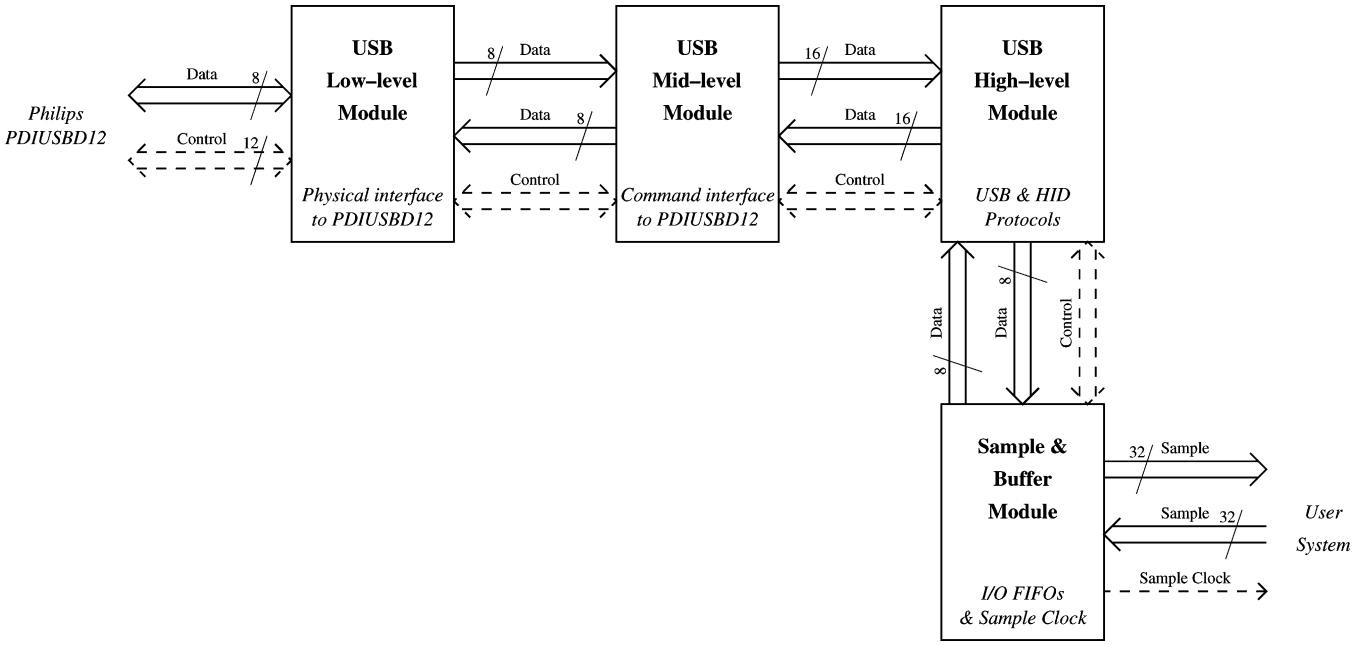


Fig. 3. USB interface on the FPGA is comprised of four basic blocks. The low-level USB module communicates directly with the PDIUSBD12 chip. The midlevel USB module implements the command interface protocol for the PDIUSBD12 chip. The high-level USB module implements the USB and HID protocols, and the sample and buffer module buffers the USB packets and provides a sample-level interface to the students' code, which is designated as user system here.

The prototyping system requires a similar USB protocol stack to be implemented on both the PC and the FPGA, as shown in Fig. 2. On the PC side, most of the USB protocol is already implemented in Windows. On the FPGA side, however, much of the USB and human interface device (HID) protocols had to be synthesized on the FPGA.

A. PC Side

On the PC side, a toolbox of MATLAB functions provides the ability to send and receive arbitrary waveforms; generate, send, and receive sinusoidal waveforms; and measure the frequency response of the hardware system. Supporting the public interface for this toolbox is a MATLAB executable (MEX) dynamic

```
COMPONENT usb_clk IS
  PORT( inclk0  : IN  STD_LOGIC;
        c0      : OUT STD_LOGIC;
        c1      : OUT STD_LOGIC;
        locked  : OUT STD_LOGIC
      );
END COMPONENT usb_clk;
COMPONENT usb_interface IS
  PORT( reset       : IN    STD_LOGIC;
        clock_45MHz : IN    STD_LOGIC;
        clock_90Mhz : IN    STD_LOGIC;
        vbus        : IN    STD_LOGIC;
        dmreq_n     : IN    STD_LOGIC;
        int_n       : IN    STD_LOGIC;
        suspend     : IN    STD_LOGIC;
        sample_in   : IN    STD_LOGIC_VECTOR(31 DOWNTO 0);
        usb_data    : INOUT STD_LOGIC_VECTOR( 7 DOWNTO 0);
        sample_out  : OUT   STD_LOGIC_VECTOR(31 DOWNTO 0);
        sample_clk  : OUT   STD_LOGIC;
        ale, a0     : OUT   STD_LOGIC;
        reset_n     : OUT   STD_LOGIC;
        dmack_n     : OUT   STD_LOGIC;
        rd_n, wr_n  : OUT   STD_LOGIC;
        a0, cs_n    : OUT   STD_LOGIC
      );
END COMPONENT usb_interface;
COMPONENT filter IS
  PORT( reset   : IN  STD_LOGIC;
        clock   : IN  STD_LOGIC;
        x       : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
        y       : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
      );
END COMPONENT filter;
```

Fig. 4. Declarations of the three components here are needed to implement the FPGA interface. The logic needed to implement the students' system is located in the "filter" component. The four modules shown in Fig. 3 are encapsulated within the usb_interface component.

link library (DLL) that handles communication with the Windows USB and HID application programming interface (API) [20], [21].

The FPGA implements a USB device that is natively supported by Microsoft Windows XP. Specifically, the FPGA board is registered as a generic HID device. When it is connected to the USB port of a PC, it will be automatically recognized, and the appropriate Windows drivers will be loaded (i.e., it is a plug-and-play device).

Once the USB device is connected and the drivers loaded, MATLAB can communicate with the FPGA board via the custom USB toolbox created for this course. The core of this toolbox is found in three main functions: sendcos, sendwave, and freq_response.

1) Sendcos: The sendcos function sends a sinusoidal waveform to the FPGA and returns the output from the FPGA. This function takes the frequency, duration of the waveform, scaling factor (resolution), and sampling frequency as input arguments. Internally, sendcos generates a cosine waveform of the specified frequency and length with an amplitude of 1.

TABLE III
FILTER COEFFICIENTS

Coefcient	Decimal Value	Binary Value
b_0	0.25	0.010
b_1	0.125	0.001
b_2	0.125	0.001
b_3	0.25	0.010

TABLE IV
DISTRIBUTED ARITHMETIC LOOKUP TABLE

Address	Expression	DA Lookup Table Decimal	DA Lookup Table Binary
0000	0	0	0.000
0001	b_3	0.250	0.010
0010	b_2	0.125	0.001
0011	$b_2 + b_3$	0.375	0.011
0100	b_1	0.125	0.001
0101	$b_1 + b_3$	0.375	0.011
0110	$b_1 + b_2$	0.250	0.010
0111	$b_1 + b_2 + b_3$	0.500	0.100
1000	b_0	0.250	0.010
1001	$b_0 + b_3$	0.500	0.100
1010	$b_0 + b_2$	0.375	0.011
1011	$b_0 + b_2 + b_3$	0.625	0.101
1100	$b_0 + b_1$	0.375	0.011
1101	$b_0 + b_1 + b_3$	0.625	0.101
1110	$b_0 + b_1 + b_2$	0.500	0.100
1111	$b_0 + b_1 + b_2 + b_3$	0.750	0.110

2) Sendwave: The sendwave function sends any arbitrary waveform to the FPGA and returns the output from the FPGA. This function takes the input waveform and scaling factor (resolution) as input arguments. The samples of the input waveform are expected to be $1 > x > -1$.

3) Freq_Response: The freq_response function uses the sendcos function to send a range of sinusoidal waveforms to the FPGA and returns the magnitudes of the respective outputs. In this way, an experimental frequency response magnitude vector is generated. If the input frequency vector contains equidistant values of the form $\text{freq}(i+1) = \text{freq}(i) + \text{constant}$, then the returned vector represents the magnitudes of the discrete frequency response $X[k]$.

B. FPGA Side

On the FPGA side, VHDL modules have been created that implement the USB and HID protocols, receive packets of data from the USB connection, rebuild the individual samples (currently supports up to 32-b samples), issue the sample and a sample clock pulse to the students' DSP system, receive the output sample, encapsulate multiple samples into data packets, and transmit them back to the PC through the USB connection. The FPGA communicates with the USB bus via a custom

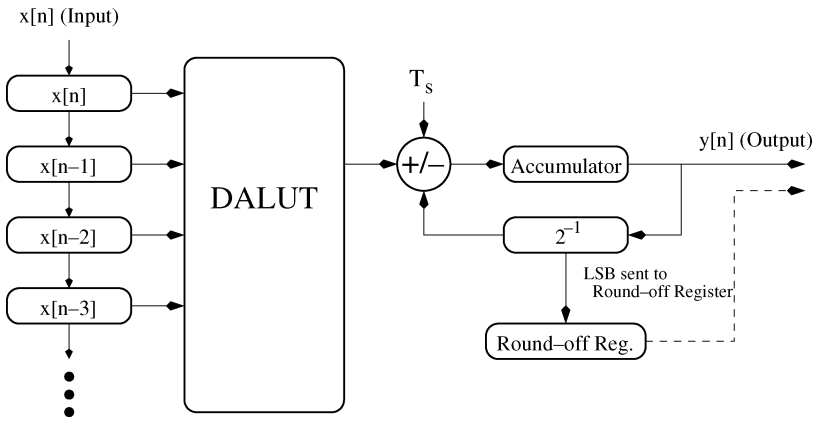


Fig. 5. Overall structure of the serial distributed arithmetic implementation of an FIR filter is presented here. Note that the address input to the DALUT is formed from delayed samples of the input signal only.

daughter card built for the FPGA board. The daughter card contains a Phillips PDIUSBD12 chip that handles the physical communication link with the USB connection. For interrupt-based transfers, this chip has a maximum throughput of 512 Kb/s. By providing all of this functionality up front, students can concentrate on implementing and optimizing the DSP systems and not the testing infrastructure.

There are four drop-in VHDL modules that provide the USB functionality to the FPGA. As shown in Fig. 3, the low-level USB module communicates directly with the Phillips PDIUSBD12 integrated chip, which handles the physical-level USB connection to the USB port on the PC. The midlevel USB module translates the commands from the high-level module into multiple command and data transfers with the PDIUSBD12 chip. The high-level USB module implements the USB and HID protocols. The bulk of this module is involved with the automatic enumeration of the USB device when it is plugged into a PC as required by the USB specification. Finally, the sample and buffer module stores the incoming and outgoing data in first-in, first-out (FIFO) buffers, provides the conversion between bytes and samples, and generates the sample clock, which can be used by the students' code to read the next sample.

The top-level VHDL project file typically contains the declarations and routing logic for at least three components: the top-level USB interface (containing the four modules shown in Fig. 3), clock phase-locked loop (PLL) (to generate the 45- and 90-MHz clocks), and the student's DSP system. These top-level VHDL component declarations are shown in Fig. 4. Except for adding the components and respective signal routing code for these modules to a top-level VHDL design file, the students need only be concerned with the data and control signals presented at the interface to the sample and buffer module.

The interface to the filter code is comprised of two data buses and five control signals. The data buses, sample_in and sample_out, are both 32 bits long, but only those bits that are used need to be routed to the students' component. The remaining bits of sample_out can be left unattached, and the remaining bits of sample_in should be tied to the sign bit of the data coming from the students' component.

The control signal sample_clock provides a clock that advances a single period each time a new data sample is output from the usb_interface component. This signal can be used to clock the delay registers or otherwise control the data flow for the filter component.

In addition, the control signal sample_reset provides a reset that can be triggered by the on-board reset signal or a reset command from the MATLAB–FPGA toolbox. All filter delay registers should be cleared on reset to allow accurate system analysis for successive sine wave inputs.

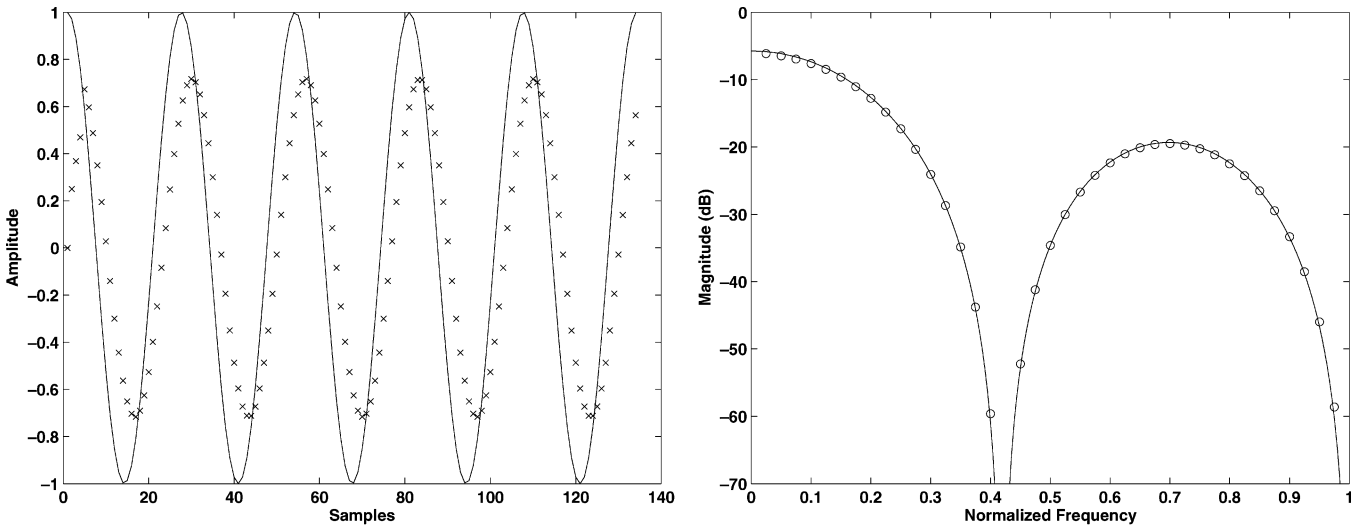
V. Distributed Arithmetic Laboratory

To illustrate the concepts covered in this course and the usefulness of the MATLAB–FPGA interface, one of the laboratory assignments will be examined in this section. The distributed arithmetic (DA) laboratory steps students through the building of FIR and IIR filters without the use of hardware multipliers. To achieve the convolution operations, a sequence of table lookups, shifts, and additions is used in bit-serial arithmetic fashion [22].

In the prelaboratory assignment (students are expected to complete this sequence before they arrive in the laboratory), students are asked to design an FIR filter with a given set of constraints. Students use MATLAB's Filter Design and Analysis Tool (a part of the Signal Processing Toolbox and Filter Design Toolbox) to complete this portion of the project. This nontrivial exercise is required since the filter must meet specifications even though only a limited number of bits may be used for the coefficients. Typically, students are asked to design a seventh- or eighth-order filter; however, to conserve space in the tables, a third-order filter with the coefficients in Table III will be illustrated here.

Next, the students generate the fixed-point binary values of the filter coefficients (Table III) and create the distributed arithmetic lookup table by adding the appropriate filter coefficients together as shown in Table IV.

The students then write the VHDL description of the hardware shown in Fig. 5, where the DA lookup table is the one shown in Table IV. The students' component is compiled with the USB interface discussed in the previous section and downloaded to the FPGA board. The MATLAB toolbox discussed previously is used to send sample inputs to the DA component and receive the outputs from the DA component. Fig. 6 shows the experimental data collected from the DA filter implemented on the FPGA. Also shown is the ideal frequency response generated from MATLAB.



(a) (b)

Fig. 6. (a) This figure illustrates the output of the DA filter for a single cosine waveform. The MATLAB toolbox of functions presented in this paper provides the interface to the FPGA. In this case, the input (denoted by the solid line) is a 300-Hz sine wave with an 8-kHz sampling frequency. As expected, the output (denoted by the x's) is a scaled and shifted version of the input. (b) Frequency response for the DA filter implemented on an FPGA can be generated by plotting the output magnitudes from input sine waves. This MATLAB toolbox encapsulates this functionality into a single function whose output is similar to the freqz() function found in MATLAB's signal processing toolbox. The theoretical frequency response (generated by MATLAB's freqz() function) is shown with a solid line. The frequency response measured from the FPGA implementation of the DA filter is shown with the circles.

VI. Conclusion

The proliferation of embedded devices, the advances in field-programmable gate arrays (FPGAs), the advent of structured application-specific integrated circuits (ASICs), and many other factors suggest that students preparing to enter industry specializing in the fields of signal processing or computer architecture need to have a knowledge of hardware digital signal processing (DSP) implementations. A course for teaching the concepts needed when designing embedded signal processing systems has been described. In addition, a framework that students can use to implement custom DSP hardware on FPGAs and easily test the performance using a MATLAB front end has been provided. All of the materials developed for this course—including the very high-speed integrated circuit hardware description language (VHDL) components, MATLAB toolbox, and daughter card PCB files—have been distributed through the authors' Website [23].

References

[1] L. S. DeBrunner and V. DeBrunner, "The case for teaching DSP algorithms in conjunction with implementations," presented at the IEEE Signal Processing Society's 2nd Signal Processing Education Workshop, Pine Mountain, GA, Sep. 2002.

[2] A. J. S. Ferreira and F. J. O. Restivo, "Grasping the potential of digital signal processing through real-time DSP laboratory experiments," presented at the IEEE Signal Processing Society's 2nd Signal Processing Education Workshop, Pine Mountain, GA, Sep. 2002.

[3] C. H. G. Wright, T. B. Welch, D. M. Etter, and M. G. Morrow, "Teaching DSP: Bridging the gap from theory to real-time hardware," *Comput. Educ. J.*, vol. 13, no. 3, pp. 14–26, Jul.–Sep. 2003.

[4] T. S. Hall and D. V. Anderson, "From algorithms to gates: Developing a pedagogical framework for DSP hardware design," presented at the IEEE Signal Processing Society's 2nd Signal Processing Education Workshop, Pine Mountain, GA, Sep. 2002.

[5] R. Chassaing and D. W. Horning, *Digital Signal Processing with C and the TMS320C25*. New York: Wiley Interscience, 1990.

[6] R. Chassaing, *Digital Signal Processing with C and the TMS320C30*. New York: WIley Interscience, 1992.

[7] ——, *Digital Signal Processing: Laboratory Experiments Using C and the TMS320C31 DSK*. New York: Wiley Interscience, 2002.

[8] W. T. Padgett, "An undergraduate fixed point DSP course," presented at the IEEE Signal Processing Society's 2nd Signal Processing Education Workshop, Pine Mountain, GA, Sep. 2002.

[9] J. Vieira, A. Tome, and J. Rodrigues, "Providing an environment to teach DSP algorithms," presented at the IEEE Signal Processing Society's 2nd Signal Processing Education Workshop, Pine Mountain, GA, Sep. 2002.

[10] A. J. Kornecki, "Real-time systems course in undergraduate CS/CE programs," *IEEE Trans. Educ.,* CD-ROM Supplement,, vol. 40, no. 4, p. 9, Nov. 1997.

[11] C. H. G. Wright, T. B. Welch, D. M. Etter, and M. G. Morrow, "Teaching hardware-based DSP: Theory to practice," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, vol. 4, Orlando, FL, May 2002, pp. 4148–4151.

[12] W. S. Gan, "Teaching and learning the hows and whys of real-time digital signal processing," *IEEE Trans. Educ.*, vol. 45, no. 4, pp. 336–343, Nov. 2002.

[13] S. L. Wood, "Signal processing and architecture in the lower division electrical engineering core," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, vol. 5, May 2001, pp. 2713–2716.

[14] K. Newman, J. O. Hamblen, and T. S. Hall, "An introductory digital design course using a low–cost autonomous robot," *IEEE Trans. Educ.*, vol. 45, no. 3, pp. 289–296, Aug. 2002.

[15] J. O. Hamblen, "Rapid prototyping using field-programmable logic devices," *IEEE Micro*, vol. 20, no. 3, pp. 29–37, May/Jun. 2000.

[16] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, "A novel high performance distributed arithmetic adaptive filter implementation on an FPGA," presented at the IEEE Int. Conf. Acoustics, Speech, Signal Processing, Montreal, QC, Canada, May 2004.

[17] D. Allred, V. Krishnan, W. Huang, and H. Yoo, "Implementation of an LMS adaptive filter on an FPGA employing multiplexed multiplier architecture," in *Proc. Asilomar Conf. Signals, Systems, Computers*, Pacific Grove, CA, Nov. 2003, pp. 918–921.

[18] W. Huang, V. Krishnan, D. Allred, and H. Yoo, "Design analysis of a distributed arithmetic adaptive FIR filter on an FPGA," in *Proc. Asilomar Conf. Signals, Systems, Computers*, Pacific Grove, CA, Nov. 2003, pp. 926–930.

[19] T. S. Hall and D. V. Anderson, "Merging theory and implementation: A framework for teaching DSP hardware design," presented at the Amer. Soc. Engineering Education Annu. Conf, Salt Lake City, UT, Jun. 2004.

[20] J. Axelson, *USB Complete*, 2nd ed. Madison, WI: Lakeview Research, 2001.

[21] C. Peacock. (2004) USB in a Nutshell. Beyond Logic. [Online]. Available: http://www.beyondlogic.org/

[22] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Mag.*, vol. 6, no. 3, pp. 4–19, Jul. 1989.

[23] T. S. Hall. (2004) FPGA Resource Page. School of Electrical and Computer Engineering at Georgia Tech. [Online]. Available: http://www.ece.gatech.edu/academic/courses/fpga/

Tyson S. Hall (S'98–M'05) received the B.S.C.M.P.E., M.S.E.C.E., and Ph.D. degrees in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, in 1999, 2001, and 2004, respectively.

He is currently an Assistant Professor in the School of Computing at Southern Adventist University, Collegedale, TN. His research interests include rapid prototyping of mixed-signal systems, cooperative analog–digital signal processing, reconfigurable computing, and embedded systems education.

David V. Anderson (S'91–M'00–SM'03) was born in Utah and raised in La Grande, OR. He received the B.S. degree (*magna cum laude*) in electrical engineering and the M.S. degree from Brigham Young University, Provo, UT, in 1993 in 1994, respectively, and the Ph.D. degree from the Georgia Institute of Technology, Atlanta, in 1999.

His research involved the development of a digital hearing aid algorithm that has now been made into an acclaimed commercial product. At present, he is an Associate Professor in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. His research interests include audition and psycho-acoustics, signal processing in the context of human auditory characteristics, and the real-time application of such techniques. He is actively involved in the development and promotion of computer-enhanced education.