Experiences with the Blackfin Architecture for Embedded Systems Education

Diana Franklin John Seng

Dept. of Computer Science
California Polytechnic State University
San Luis Obispo, CA 93407
{franklin,jseng}@csc.calpoly.edu

Abstract

In the course of a major curriculum change at California Polytechnic State University, the embedded processing course was redesigned. During this process, the course had the opportunity to purchase new hardware. Each choice has different tradeoffs in cost, performance, development environment, and documentation.

We first present our goals in the class and how we came to choose the Blackfin architecture using Analog Devices' EZ-KIT Lite development board. We then give an overview of the architecture along with the implementation of an expansion board developed to interface with the EZ-KIT Lite board.

We then present our experiences with this setup in the hopes that others who might be thinking of a similar curricular change can learn from our successes and failures. We outline the strengths and weaknesses of the Blackfin architecture as an education platform. We then discuss our experiences and present the support materials we developed to accompany the course, including lecture material and laboratories. Finally, we discuss our future directions for our uses with the board.

1. Introduction

Designing the curriculum for an embedded processing course is especially difficult in today's schools because of the many conflicting goals in curricular design. The ideal would be cheap, flexible, powerful hardware coupled with an industrial-strength, intuitive, feature-rich development environment, augmented with a textbook that is targeted towards students rather than a manual targeted at professionals. If we take a step back and look at the entire curriculum, we would also like a processor that could be used for a wider array of classes, such as digital signal processing, as well as student projects.

Unfortunately, such a bundle of technology, educational materials, and cost do not coincide. At California Polytechnic State University, San Luis Obispo, the Blackfin processor. It satisfies several of the above goals, mainly that it is cheap, general and powerful hardware, coupled with a good development environment, but it was not without disadvantages. Our students used the manuals, augmented by lecture slides.

In this paper, we explore the tradeoffs that are involved in designing a single class, CPE 316, Embedded Systems, at California Polytechnic State University, San Luis Obispo. We describe our design and how it relates to those tradeoffs. Finaly, we augmented the original hardware and developed a detailed set of lecture slides that follow the Blackfin, which currently has no textbook written for it. We provide the class materials that we developed on-line at http://www.csc.calpoly.edu/ franklin/316/Bundle.tgz

We begin by analyzing our curricular goals for the embedded systems class in Section 3. We continue in Section 4 by describing the Blackfin architecture, our architecture of choice, and the development environment provided. Section 5 presents the expansion board design and the flexibility it gives to the labs. Section 7 and Section 8 gives a brief summary of our lectures and labs from several instantiations of the class. We give ideas for future development and conclude in Section 10.

2. Related Work

Embedded processing has become increasingly important, and with its rise in industrial significance, the best way to teach the concepts has been studied by several educators.

Many groups have looked at high-level approaches to improving embedded processing education in the curriculum. Michigan State University proposed an approach to integrate embedded processing into the whole curriculum rather than a single course [1]. A full curriculum targeted towards embedded processing, including design from math classes and engineering classes on up, has also been proposed [3]. They stress that high-level principles, not specific information commercial companies might want, should be emphasized.

We take on many of the practical matters in designing an embedded processing course. We assume that the core topics have already been decided, and it is our job to convey this information in a way that fits well with the rest of the curriculum, is up-to-date, is not too costly, and fulfills as Many educational goals as possible in a major that has little room for additional requirements.

Course	Integration	Financial
textbook	unlike MIPS	inexpensive boards
intuitive software	parallelism	multiple courses
breadboard access		DSP

Table 1. Summary of goals

3. Goals

As with any course development, there were disparate goals in designing this course. There were three underlying sets of goals. First, we had the normal goals that anyone does with an embedded processing course, that of conveying the information for the course in the most painless, efficient manner. Second, we had issues with integrating this course with the rest of the curriculum. Finally, we had financial considerations to minimize the amount of hardware necessary to purchase. These goals are summarized in Table 1.

Within the embedded processing course, we had several goals. The hardware needed to be easy to use, with a development environment that was intuitive and quick for the students to pick up. Cal Poly is on the quarter systems, so the students can not waste much time learning new environments. In order to allow control of interesting devices, it needs a mechanism for students to connect their own breadboard to the processor. Finally, we wanted a textbook for the concepts in the course. We wanted one of two things - a textbook that is not tied to any single processor, coupled with manuals or a textbook that was coupled with our textbook. The former is possibly more realistic for the workplace, although the latter is easier on the students.

No course is in isolation, so there are higher-level goals to consider. Prior to this, the major language is MIPS because of its use in the Hennessy and Patterson architecture book. Students should have exposure to a variety of languages, so an assembly language that illustrates a new set of features is useful. Finally, the students have not yet been exposed to parallel processing, so a language that allows parallel instructions is desirable.

The financial considerations are listed last, but in this economy in a public school, they often become the overriding factor. We needed boards that were either donated or inexpensive. In order to amortize the cost of the boards, they should be used for multiple classes. To this end, the processor should be powerful and capable of digital signal processing tasks.

In the end, we were able to satisfy almost every goal except for the textbook. In the rest of the paper, we present how we satisfy the goals through the use of the EZ-Kit Lite Blackfin board and special hardware attached to it. For a texbook, we used a combination of detailed lecture slides and helpful laboratories. We were not satisfied enough with the general textbooks we found to require the students to purchase them. Because this was a senior level course, we expected that this was a more gentle introduction to the resources that will be available on the job.

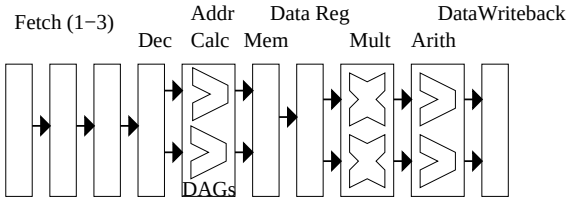

Figure 1. The 10-stage Blackfin Pipeline.

4. Blackfin Architecture

The Blackfin is a hybrid microcontroller and digital signal processor. We used the EZ-KIT Lite, which was obtained at an educational discount from Analog Devices. We now present the interesting details about the Blackfin environment we had, split up into architecture, assembly language, software development environment, and EZ-KIT Lite board.

4.1 Architecture

The Blackfin is an in-order, multi-issue processor. The pipeline has two data paths throughout. The processing core consists of a 10-stage pipeline. The pipeline is depicted in Figure 1.

Instruction fetch requires three stages, with a decode stage fourth. It fetches 64 bits each cycle, though serial instructions require only 16 or 32 bits. It only executes 64 bits in a single cycle in the presence of a 3-wide parallel instruction.

Stages five and six are for memory operations and branches address calculations. It employs two Data Address Generators (DAGs) for address calculations. Once the branch address is calculated, it uses static branch prediction to go to the predicated destination.

The Blackfin reads the data register file in cycle seven, and then performs computations in cycle eight and nine. For computation, it performs multiplication first and then has an alu for accumulation or any other arithmetic operation. It also includes special-purpose video units. Data results are written in cycle ten.

There are two register files - 8 32-bit data registers and 8 pointer registers. It also has several special-purpose registers for looping and memory address calculations. In addition, there are two 40-bit accumulators, one associated with each multiply/ALU pair.

The Blackfin has three caches - two data and one instruction cache. There is also a very small instruction buffer in the fetch unit that can hold short loops. In each cycle, you can perform a load from all three caches. You may not perform two loads to the same data cache in the same cycle.

The architecture of the Blackfin itself presented an excellent opportunity to reinforce the ideas taught in the computer architecture course. The pipeline was still in order, but it had more pipeline stages and the stages were performed in a different order than the MIPS processor. The students were also

able to learn about static branch prediction, which was not emphasized in the previous course. Finally, the presence of dual data caches allows students to think consciously about when their data is accessed in order to place data such that you can access both caches in the same cycle.

4.2 Assembly Language Features

The Blackfin ISA has several unique features beyond the simple MIPS instruction set. The main differences are the address calculation features, control features, variable data widths, and parallel processing.

The DAG allows for a very rich set of addressing modes. In general, one can access a memory location at a constant offset from a register index and increment the index in a single instruction. Furthermore, it allows circular addressing with a stride, automatically wrapping the pointer around when it reaches the end of the buffer. It also has bit-reverse addressing specifically designed for the FFT algorithm.

In order to maintain high performance with a 9-stage pipeline with, it needs branching support. The Blackfin provides two major mechanisms to help branches. First, it provides static branch prediction. Any branch can be labeled to predict taken. Unfortunately, this only saves four out of eight stall cycles. The address is not calculated until cycle four, so for loops with a known number of iterations, the Blackfin provides a zero-overhead loop mechanism in the fetch unit. It can keep track of two nested loops at once. The entire loop is buffered in the unit, along with the counter and the beginning and end program counters. This automatically provides the proper instruction with no stalls until the loop is complete.

The Blackfin provides support for 16-bit operations as well as 32-bit operations. You can either perform a single 16-bit operation on each ALU or have each 16-bit half of a 32-bit number be treated as a separate 16-bit value for the purposes of arithmetic operations. This allows one to perform four 16-bit operations in a single cycle when employing both ALUs.

Finally, as referred to above, the Blackfin allows limited parallelism. It may perform two 16-bit and one 32-bit operation at once, drawn from a list of parallelizable operations. Only one store may be performed each cycle, though one can perform two loads. There are two DAGs, so address offsets and updates may also be performed in parallel.

This instruction set satisfied all of the educational goals of the assembly language. The advanced branching instructions allowed for an excellent tie-in of core architectural material to the course, and the parallel instructions provide a unique opportunity. This was especially important because even correctly predicted branches as well as unconditional jumps had a 4-cycle penalty. The ability to control branches in the assembly language and think about the performance ramification makes the knowledge more concrete.

4.3 Software

The software environment needed to be intuitive and easy to pick up, especially in our quarter system. We use the Analog Devices' Visual DSP++ as an integrated development environment for the class. Visual DSP++ is designed to be used with the EZ-KIT lite, a processor simulator, or with a JTAG interface. This program allows programming the board in Blackfin assembly or C and provides an overall interface which is highly similar to other integrated development environments.

The only problem with this software is a combination of hardware problems and the license server. Occasionally, it gets into a state in which the student can no longer control the hardware. If they close the program, the license does not always return to the license server right away. Upon attempting to restart the program, the license server will say it is out of licenses. This requires a license server restart.

4.4 EZ-KIT Lite Board

The EZ-KIT Lite Board provides I/O opportunities for students with the Blackfin chip. They provide flexibility in augmenting their design by having flash memory that can be used to configure different input and output pins.

The most basic functions that are fun and easy to use are the LEDs and the pushbuttons. The sample codes that come with the board are simple for the instructor to understand.

The board also includes more advanced features like audio/video and bus protocols. The video interface was used in a lab described in Section /refsec:labs.

The main problem with the board is that it has a set of switches on it that, if changed, cause the board to act in odd ways. At Cal Poly, the labs are open to allow senior project students to use hardware for their projects, but they are not monitored at all times. Students will flip the sometimes flip the switches, and it is difficult to tell.

Unfortunately, there was no good access for connecting a student breadboard. Section 5 will describe the expansion board designed to give students access to several input and output pins on the EZ-KIT Lite board.

4.5 Documentation

The Blackfin architecture has a manual as well as a separate Instruction Set Architecture manual. In addition, the EZ-KIT Lite Board has a manual. These manuals are all electronic. Students may request hard-copies as well, though they are very large and heavy.

The Blackfin architecture and ISA manuals are very well indexed and easy to navigate through Acrobat Reader. The EZ-KIT Lite is a little more difficult to utilize efficiently. We

found that the students were more comfortable with the physical versions of the books and had not had much experience with electronic manuals. In retrospect, I wish I had done a half a lecture on how to navigate the manuals effectively.

4.6 Discussion

The Blackfin 533x on the EZ-KIT Lite board satisfied our hardware goals. It had an intuitive environment, though not bug-free, it was inexpensive, it had an assembly language sufficiently different from MIPS, allowed for parallel execution, and had the functionality for digital signal processing. The only thing it lacked was a simple interface to a student breadboard.

Our experience with this hardware was mostly positive. When problems occurred, though, it was very difficult to track them down. It could be the students' software, the hardware switches, the connection to the development environment, or a bad state. When restarted, sometimes the license server would then fail.

To alleviate this, students should be counseled early in the class to save working versions of their code to determine whether a problem is with their code or the board.

5. Expansion Board Design

Although the Analog Devices' EZ-KIT lite board is highly integrated and provides excellent performance, the board is not designed to be readily used in an educational environment. Several of the board pins are connected to other chips and are not available for use through on-board pin headers. Unfortunately, the board does not provide easy access to input/output pins. What the board does provide is a 3-socket expansion interface intended to be used with other Analog Devices' expansion cards. Each socket is a 90-pin connector with a fine pitch spacing. We use this interface to connect a custom expansion board for use in a class lab environment.

Our expansion board contains simple circuitry to buffer some of the input/outputs pins on the board. One fact to note when using the I/O pins of the Blackfin is that the I/O pins on the Blackfin processor use 3.3V interface circuitry. Connecting 5V circuits directly to the I/O pins would damage the blackfin. Instead, we used voltage level conversion buffers to allow 5V circuitry to be used during the labs.

The expansion board design provides a modest number of digital inputs and digital outputs. The design allows software to control 8 digital outputs and 8 digital inputs. Should more inputs and/or more outputs be required, an SPI I/O port expander would be good for that purpose. Also, a CD4094, would work well as an output expander because of the shift-and-store inteface it provides.

A 24-pin ribbon cable is used to connect the students' breadboard with the Analog Devices' board. On one end of the cable is a polarized connector which connects with the expansion board and on the other end is a 24-pin DIP socket which plugs directly into a breadboard.

6. Textbook

Currently, no textbook exists for that targets the Blackfin architecture. We considered a more general textbook, such as Computers as Components [4]. Although this was useful to use as instructors, and we incorporated some of the publicly available on-line slides into the lectures, it was at such a high level that we made it a recommended textbook, not a required textbook.

This meant that are lecture notes were the only resource the students had beyond the manuals. Our lectures slides are a combination of high-level, general material, followed by specific information for the Blackfin architecture.

7. Lectures

The lecture slides were a combination of theoretical material and Blackfin-specific implementation. The figures for the Blackfin-specific material were obtained from the hardware and ISA manuals [2].

Because there is no textbook on Blackfin, the lecture slides were the only reference the students had in a more educational format tying the relevant information together. We are releasing the slides so that they may be used as a building block for someone to tailor their own slides if they wish. They are by no means complete and will continue to be developed as the class is taught more often.

7.1 Lecture Topics

We have created a set of lectures that cover the core embedded processing subjects as well as additional special topics that are related to architecture and embedded processing in general. The core topics are:

- Memory-Mapped I/O / Polling
- Interrupts
- Timers
- Ports / Buses
- DMA and Power
- analog / digital conversion

We also added several topics, ranging from architectural lectures to tie the chip back to concepts introduced in the architecture classes to pure C and assembly programming techniques.

- Blackfin Overview / ISA describes the overall architecture as well as giving examples on branching mechanisms and loading and storing.

- Blackfin Pipeline Gives details on what each pipeline stage performs. Timing diagrams of instruction sequences and their stall cycles.

- Blackfin Calling Convention, functions vs. handlers Generic function call convention in the context of Blackfin and why handlers do not get to follow the rules of calling convention.

- Static Branch Prediction Details on the zero-overhead-loop, static branch prediction, conditional instructions. Includes timing diagrams and statistical performance problems. Relates the branch penalties when operations occur in the pipeline.

- Parallel Processing Statically scheduled parallel programming, Blackfin parallel instructions, loop unrolling, software pipelining.

- C for Assembly Programs C keywords that range from necessary to useful when programming with devices. Memory regions and scope in C. The keywords volatile, register, static, inline. Utilizing two data banks. Blackfin-specific keywords restrict, making easily-recognizable circular buffers, inline assembly. Interfacing C functions with assembly functions. regions, scope,

- Optimizing Code Introduces the idea of profiling, Amdahl's Law, and test input sets. Presents several optimization techniques like DMA, data locality, and some simple examples of branch removal.

7.2 Discussion

Several of the additional topics were taught in the second instantiation. This led to some different observances in the lab work for the course.

We found that the first time this course was taught, before the C for Assembly lecture was included, students strongly preferred using C in the laboratories. After the addition of the C lecture, students were much more comfortable using C, and more than half of the students used C when they were given a choice.

Before the calling convention lecture, students had very little idea of how, from a register point of view, the handler should be written. Some students were reserving registers to be used as communication between the main loop and the ISR, whereas others were destroying random registers without realizing that this would affect the registers used in the main loop. This greatly enhanced the understanding of both the unpredictability of when the ISR is called and the importance of register usage conventions.

For the rest of the extra lectures, they are very much bonus material intended to reinforce concepts learned in either assembly language courses or architecture courses. An embedded processing course is the ideal place to do this, since this is sometimes the first time students have needed to program in a meaningful way at this level. In previous courses, they often felt the assembly language was just an educational task with no real purpose. Once they see the usefulness, one needs only to bring in a performance-critical problem in order to expand the focus of the course. This gives the opportunity to teach about profiling and high-level code optimizations all the way down to branch prediction, code scheduling, and pipelining. It can serve as a great culmination of all of the software and hardware skills the students have learned.

8. Labs

Our labs were designed with a few goals in mind. First, we wanted to target the skill sets of polling, interrupts, and control. Second, we wanted to make the labs interesting so that by the end of the quarter, the students could imagine themselves building a robot if called upon to do so. Finally, we needed to fit everything in a 10-week course. The labs below are not from a single instantiation of the class, but chosen from various instantiations of the class. The full text of the labs can be perused at: and downloaded from:

8.1 Polling

The original polling labs were fairly uninteresting, only requiring the students to respond to button presses by changing patterns on the LEDs.

A proposed future lab would create a Simon Says game where the LEDs would light up in a certain sequence, and the player would need to repeat that sequence with the buttons. The computer would keep generating faster and longer sequences until the player could no longer get the sequence correct.

8.2 Interrupts

Implement a ping-pong game, where the LEDs present the ball, and the buttons are the paddles. A player can lose by either pressing the button at the wrong time or not pressing the button when the ball is there. At the end, display a message that indicates both who won and why they won. As the game continues, the ball needs to accelerate.

This lab served several purposes. It was fun for the students, required thought as to how to detect all the ways to lose,

and allowed for some flexibility in design by having them decide how to display the loss. Several students even implemented extra functionality by allowing a game reset with one of the other buttons. In one instantiation of the course, this was the most successful lab.

8.3 Nested Interrupts

The nested interrupts assignment was a part of the interrupts lab. They were to display morse code depending on what button was pressed, but allow interruption of displaying the different patterns depending on which other button was pressed. They were to implementing the displaying of the pattern in the interrupt service routine, not in the main program.

8.4 Timers

For this lab, the students built a dimmer. The light's brightness was controlled by the amount of time the light was turned on. Timers controlled the light turning on and off. When one button is pressed, the light gets dimmer, and another causes the light to get brighter.

The students enjoyed this lab very much. The biggest mistake was to change how often the light turned on and off without ever turning the light on for a longer period than it was turned off.

8.5 Advanced labs

In various instantiations of the class, the last lab involved the students receiving input from external devices, performing some operation and producing output for an external device. These devices could be hooked up to the breadboard.

Servo Lab We had a standard hobby servo that is controlled by a 1-2ms pulse with a period of 20 ms. If the pulse width is 1ms, it is turned all the way to the left. At 2ms, it is turned all the way to the right. You can place it anywhere in between by adjusting the width between 1-2ms. The period must stay constant at 20ms.

The servo was controlled by the buttons. There were two instantiations - two buttons set them to far left and far right, while the two middle buttons made the servo rotate slowly to the left or right. In the other version, all four buttons determined four positions for the servo to point.

Potentiometer A potentiometer dial, when rotated, adjust the power it is sending between 0 and 5 volts. This is then connected to an ADC0831 and read in by the students.

The ADC0831 interface was the most complex the students encountered. They needed to transmit a chip select signal along with a clock to the ADC0831 and then sample the incoming bit 8 times in order to obtain the 8-bit value for the volt.

Students did not realize how precise the timing needed to be about putting the chip select down before beginning the clock, and then waiting a cycle before beginning the sample.

The potentiometer was used to control the LEDs. The LEDs could either display the 8-bit number in binary, or it could look more like a voltmeter with the number of lights growing from one side or another.

The potentiometer and servo can be combined ot have the potentiometer control the servo. This involves more coordination for the students, but they thoroughly enjoyed getting the hardware to work. This lab was a highpoint for many of the students.

8.6 Discussion

There are many ways to design the labs. In our quarter-system environment, we felt the need to streamline the labs so that the students could learn the most concepts in the shortest amount of time. This led to tradeoffs in how the labs were structured as well as to how much information was provided to them.

When designing the labs, we had a trade-off between small problems that targeted specific skills and large labs that would take fewer different files. Due to a combination of the development environment and the fact that they were initially coding in assembly, the overhead with beginning a new program was quite large. In retrospect, it is important that different parts merely build on each other and do not require a new codebase. What were listed above are the core projects, although the actual labs often include some smaller, simpler parts before building to the full lab. The intent of the smaller parts was to allow for more partial credit if students could not get the whole thing working. In the future, teaching the students about how to break down large projects in order to test them thoroughly would have been better than cutting the projects up into different parts that did not directly build on each other.

There were also differences in how we implemented the labs. The first instantiation of the course provided students with only the manuals, requiring them to begin from scratch. The second instantiation of the course provided sample code (often the code similar to that shipped with the board) so that they could use that as a baseline and modify it for the specific assignment. In order to try to ensure the students took the time to understand the given code, a set of questions was asked about the sample code and turned in. This definitely made it quicker for students who could learn from sample code to finish the projects. Several groups that understood the concepts completed early labs in very little time. Struggling students resorted to some method of random code replacement,

not truly understanding the sample code and often not making the changes to it in the right places. It is clear that for strong students, the sample code method removed much of the tedium that would have been involved and took nothing away from the learning. For the struggling students, however, it is unclear which was better. With no sample code, they do not know what to generate on their own, so it would take much more time to solve the labs. On the other hand, if they solve the early lab, that would give them a more solid foundation to solve later labs. With sample code, they could more easily fool themselves into thinking they were not so lost.

9. Future Work

Since the course is still in its first year, it will continue to be developed in the coming years. In the future, we will be augmenting our slides with material and improving the laboratory projects.

For the lectures, the more general material was not integrated seamlessly into the Blackfin-specific details. More work will be done in the following year with obtaining support materials for the students and integrating them into the lecture slides.

In addition, more hardware components can give new and interesting laboratory assignments. There are a variety of labs that could be added for a course that is a semester long. This would open up the possibility of an open-ended project for the last month of the course. In addition, we did not have time to touch upon code optimization in the laboratories. We could give a task and have the students learn how to profile code, time their code with on-board timers, and have a contest as to which group had the fastest solution. The students were very excited about such a prospect.

10. Conclusion

Embedded processing courses will always have a difficult time keeping up with technology because students work at the assembly level. Textbooks are hard-pressed to keep up with the new hardware offerings, and schools face many pressures when choosing a development platform.

We give analysis on what problems were faced in designing our embedded processing course. We found a hardware / software environment that serves most of the goals set out, and we have augmented the available materials with our own. Our materials are now publicly available. The course was largely successful, with just a few changes needed in the material to present in order to make up for the lack of a textbook. We hope others who choose the same setup will be able to learn from our contributions of materials and experiences.

References

[1] B. Chang, D. Rover, and M. Mutka. A multi-pronged approach to bringing embedded systems into undergraduate education. In *ASEE*, 1998.

[2] A. Devices. *Blackfi n Processor Family Manuals*. ADI, 2005.

[3] S. Guangfan, W. Peidong, L. Jinbao, and W. Kaizhu. A curiculum design and consideration for the embedded systems. In *ICITA204*, 2004.

[4] W. Wolf. *Computers as Components*. Morgan Kaufmann, 2001.