AC 2008-1341: ON THE USE OF A SOFT-CORE PROCESSOR IN JUNIOR MICROPROCESSORS COURSE

Arlen Planting, Boise State University

Sin Ming Loo, Boise State University

Use of Soft-core Processor in a Junior Level Microprocessors Course

Abstract

Traditionally, microprocessor courses are taught using discrete microprocessors. With the availability of field-programmable gate array and associated soft-core processors, this traditional approach can be changed to provide added educational benefits. The junior level microprocessors course at Boise State University was recently updated using the Altera Nios II soft-core processor. The goal of this course is to teach the basics of microprocessors and peripheral interfacing techniques. Along with replacing the traditional discrete microprocessor with a soft-core processor, the course was modified with the addition of the C programming language. The course used assembly language to teach the features and capabilities of the Nios II processor (instructions, registers and memory) and quickly migrate to the C programming language using a task-oriented approach rather than an exhaustive coverage of the language. Both instructors and students gained valuable experience through this process.

Course Objective

The ECE 332/332L Microprocessors course at Boise State University covers microprocessor architecture, software development tools, and hardware interfacing with emphasis on 16- and 32-bit microprocessor systems. Machine and assembly language programming, instruction set, addressing modes, programming techniques, memory systems, I/O interfacing, and interrupt handling are among the topics studied with practical applications in data acquisition, control, and interfacing. This course was reported to be a favorite of many students, largely because of the interesting devices that could be played with by the end of the course (such as the magnetic card reader). The intent was to retain and potentially enhance this characteristic of the course with the changes implemented.

One impetus to update the Microprocessors course came from experiences teaching advanced computer engineering courses with field-programmable gate arrays (FPGAs). Since at that time the students had not previously been exposed to FPGAs and soft-core processors, valuable time was spent teaching basic concepts that could reasonably be learned in lower level classes. Updating lower level courses such as the Microprocessors course was undertaken to advance the process of providing a foundation for the advanced level curriculum. It was also observed that seasoned hardware engineers from the local computer engineering community were returning to school to improve their software skills. Enhanced software development skills would not only benefit students in advanced digital courses that require a higher level of proficiency in programming, but would also increase the students' future value in the workplace.

Demographics

Since the ECE 332/332L Microprocessors course is a requisite for both electrical and computer engineering (ECE) and computer science students, the course must endeavor to address the disparate interests and needs of students in both disciplines. In addition to those specializing in computer engineering, the ECE group includes students interested primarily in other areas such as integrated circuits, communication and signal processing, control systems, power and energy systems, etc. The primary interest of most computer science students is not small systems. Therefore it is important to attempt to achieve a balance in the course that will adequately teach electrical engineering and computer science students the needed fundamentals of microprocessors, while also providing the computer engineering students a solid foundation for advanced courses.

Decision Process

Options

Several options were initially considered to accomplish the objective of better preparing students for advanced ECE courses. One option was to require all students to take CompSci 253 (Object Oriented Programming with C). The CompSci 253 course addresses typical large computer systems with an operating system that provides low level services (such as memory management). This is a feature not typically found on the small microprocessor systems that ECE students are likely to encounter.

A second alternative was to develop a pre-requisite course that covers the same material as CompSci 253 but with emphasis on small systems. However, the computer science students would not benefit from this since it would cover material not needed in their curriculum. Additionally it would be difficult to find sufficient material to cover without using hardware devices.

The best option appeared to be modifying the existing Microprocessors course. The updated course would expose the students to soft-core processors utilizing FPGAs. This is relevant technology also used by practicing engineers. Introduction of the C programming language was proposed because of its ability to enhance productivity and portability. The course would introduce just enough material from the C programming language that students could work with devices at a low level (minimizing the overlap for the computer science students). This would also give some ECE students their first exposure to the C programming language.

Before updating the Microprocessors course, an experimental course addressing the usage of the C programming language for embedded applications was undertaken to investigate methods of incorporating the C programming language in the electrical engineering curriculum. The experimental course included an accelerated presentation of the C language directed to specific course objectives. When it became apparent that some of the students were struggling with the accelerated approach (basics of the C language in four weeks), the C language instruction was extended for two more weeks. As it turned out, this protracted coverage didn't provide the desired benefits and the better students began to lose interest. However, the exposure to key concepts of the C language did prove to be of value for all the students.

Resource Selection

Development Platform

The use of FPGAs in place of traditional instructional platforms has been an important part of the process of updating the computer engineering curriculum at Boise State University. A FPGA is a semiconductor device containing programmable logic components (logic block), programmable interconnects and memory elements. This programmability allows the user to change functionality an unlimited number of times to fit the desired application. For the Microprocessors course, the FPGA is used to instantiate a soft-core processor.

The Altera DE2 (Figure 1) was selected as the FPGA development board for updating the Microprocessors course. With a complete set of tools, a large selection of standard devices, and the ability to be reconfigured, the Altera DE2 has all the features necessary for prototyping and addressing numerous protocols. The Altera DE2 integrates industry-standard development and debugging tools (Eclipse IDE, GCC compiler and GDB debugger) that the students are likely to encounter in their careers.

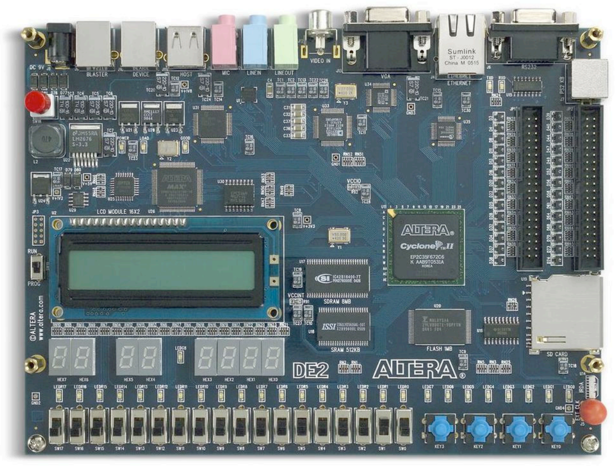

Figure 1. Altera DE2 Development and Education Board

The DE2 comes furnished with many attached devices and also has two expansion headers for almost unlimited add-ons, allowing a large range of future new devices to be attached. The interface logic to connect these devices can be improved or modified for instructional purposes. Therefore, choosing a development board is not a long-term commitment to a particular set of devices. This setup is unlikely to become obsolete quickly because it has device support for USB, audio, VGA, Ethernet, UART, PS2, secure digital, and expansion headers.

Soft-core Processor

A soft-core processor is a microprocessor core that can be wholly implemented using logic synthesis. It can be implemented via different semiconductor devices containing programmable logic. Soft-core processors capitalize on the flexibility of an FPGA to allow the processor to be customized to meet the needs of a target application. Although use of a soft-core processor entails learning associated development tools that wouldn't be required for a discrete processor (and are frequently updated), the advantages for educational purposes far outweigh the disadvantages.

Utilizing a soft-core processor for teaching purposes offers an advantage by providing the capability to expose the students to numerous different hardware configurations in a single course. Dedicated boards require that a connection resource be permanently allocated for specific purposes, thus limiting usage of board. Conversely, a soft-core processor is analogous to a stage where a new prop can be brought in for each new production. Each device/concept can be introduced with a unique processor configuration. For example, to illustrate the concept of cache memory, a processor can be generated without cache and instruction performance and results can be compared to a configuration that includes cache memory.

As new devices become available or time permits the development of instructional materials, they can more easily be integrated into the course curriculum with a soft-core processor. All of the work does not need to be done ahead of time and developed on a dedicated board for months/years into the future. Minor changes to labs can be made each year without requiring major redesign of dedicated boards. The configuration of the soft-core processor can grow or shrink as the needs dictate. Simple configurations can be used at the beginning so students can more easily grasp the big picture; more complex configurations can be generated as their understanding increases. If a project requires multiple UARTs, it is easy to add them.

The Nios II processor was used for software development on the Altera DE2 for this course. Though the soft-core processor was considered the best instructional platform, it is acknowledged that it still needs occasional comparison (via demonstration) to traditional discrete microprocessors.

Since generation of soft-core processor configurations is not a primary component of the Microprocessors course and requires a great deal of time to adequately cover the theory and tools involved, it was decided to leave the topic for an optional demonstration late in the course. It was considered more beneficial, particularly for the non-computer engineering students in the class, to spend the available time on topics other than hardware configuration. Since the demographics of the class do not allow much flexibility for advanced topics relative to hardware, focusing on the basics to provide a good foundation for those wishing to explore hardware configuration issues in a later course was the approach chosen.

Tools

The tools provided by Altera can be categorized into two major groups:

1. Tools to produce hardware configurations (Quartus II, SOPC Builder)
2. Tools to develop software solutions for soft-core processors (Nios II EDS, Altera Debug Client)

The first group of tools involves the generation of soft-core processor configurations which, as noted above, was not a part of the Microprocessors course. These tools were used by the instructors, but were not presented to the students.

For software development, Altera provides an educational development tool (Altera Debug Client) in addition to commercial grade tools (such as Nios II IDE). Altera Debug Client provides assembly of assembler programs with no run time support, and compilation of C programs with minimalist (no ISR or exception handling) run time support. The Altera Debug Client also loads resulting code into the DE2 and establishes a debug session with capabilities to see disassembled code, view and modify registers, view and modify memory, and perform other debug functions (e.g. breakpoints). This is ideal for educational purposes.

The commercial grade tools are tightly coupled with the processor device configuration tool (SOPC Builder), and automatically generate software libraries (HAL) to support most of the devices generated with a particular hardware configuration. This facility is very convenient for advanced users, but hinders the learning process for beginners. For this reason we decided to start with the educational tool (Altera Debug Client) for the first part of course before introducing the more advanced (Nios II IDE) offering. Using Altera Debug Client eliminates the startup code provided by the C runtime and the exception handling in the Nios II IDE, and more importantly, assures that students have to provide the functionality themselves. This provides a better learning environment for assembly language.

The Nios II IDE was used in the course for development using C language. Some modifications were made to deal with the HAL functionality that provided ISR support, in order to ensure that the students understand/know all the layers that exist between their application and the hardware. (The HAL is a layer of software between the physical hardware and the application software running on a computer system, which allows programs to be more easily moved from one system to another. Its purpose is to hide the implementation of a hardware device from the upper layers of software.) Altera's implementation of HAL is automatically included for devices that are integrated with a processor configuration using the SOPC builder, and Altera has done a good job of tying the software libraries generated to the devices supported in a processor configuration.

Altera's Nios II IDE is based on the popular Eclipse IDE framework with Altera-supplied plug-ins. These plug-ins manage the Nios II projects and the make process. If alternate plug-ins could provide minimalist support then the use of Altera Debug Client could possibly be avoided, thus eliminating the need for students to learn two development environments. We have done some investigation into the possibility of user-managed make files for Nios II projects and will do more in the future.

References

Textbooks are available for most traditional microprocessor platforms. However, since the concept of teaching soft-core processors is still evolving, we found no single source text that addressed teaching with the Nios processor. Though there was no unifying document that consolidated the information needed for the updated Microprocessors course, usage of the soft-core processor was considered valuable enough that this did not change our decision. We accepted the challenge of generating our own instructional materials for the processor-specific (assembly and C) portion of the course. The textbook "Embedded System Design - A Unified Hardware/Software Introduction" by Frank Vahid and Tony Givargis was selected as the text for the lecture portion of the course.

Course Approach

The Microprocessors course actually consists of two separate parts, the lecture (ECE 332) and the lab (ECE 332L). Though ECE 332 and ECE 332L are co-requisites, each part is individually graded and assigned credits (3 credits for the lecture, and 1 credit for the lab). Lecture classes are taught twice a week in two 1 hour-15 minute sessions, and the lab portion consists of a 3-hour session.

The updated Microprocessors course as taught at Boise State University in the Fall 2007 semester introduced both assembly and C languages, and utilized a soft-core processor instantiated on an FPGA. The labs developed for the updated Microprocessors Lab course are summarized in Table 1. The first three labs provided the students hands-on experience with microprocessors utilizing the assembly language as covered in the first five weeks of the lecture series. The remaining labs dealt primarily with the C language, with the exception of a portion of the ISR lab.

Table 1: Updated 2007 Microprocessors Lab Course Outline

Week	Topics/Assignment
1-2	Familiarization with DE2, Nios II, assembly programming, and Debug Client; introduction to memory. (assignment: bubble sort routine)
3	Parallel Input/Output (PIO): Interfacing to LEDs and switches (assignment: create subroutines and macros)
4	Seven-segment display. Develop display system to interface to switches, LEDs, buttons and seven-segment.
5	Switch from Debug Client to Eclipse IDE: Nios II tutorial
6	C programming language and interfacing to buttons, switches and LEDs
7	LCD interface routines using C
8	Exploration of interrupt service routines (ISRs)
9	ISRs with assembly only; ISRs with C and minimum HAL
10	Introduction of hardware abstraction layer (HAL) for Nios II
11	PWM and DC motor with H-bridge
12-16	Final Project

Numerous references, including textbooks, vendor-supplied handbooks and tutorials, and data sheets were utilized in the course. Reference manuals and tutorials supplied by Altera for the DE2, the Nios II processor, Nios II software developer and Altera Debug Client were used for the course. "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie (K&R) was the primary reference for the C language portion.

For the assembly language portion of the course, Altera Debug Client was utilized in lieu of the standard commercial grade development tools available from Altera. The Nios II IDE development tool was used for development in the C language. However, since one of the learning objectives for the Microprocessors course is interfacing to low level devices and handling of interrupts, Altera's implementation of HAL was disabled for exception processing (ISRs).

Assembly language programming concepts were presented with a mixture of devices to help keep interest in the labs. Introduction of devices began with memory, followed by PIO (LEDs, switches, buttons, and the seven segment display). This took about 4 weeks and then the focus of the course moved to the C programming language. Based on the belief that it is not necessary to teach the entire C language to significantly enhance software skills beyond what is achieved with assembly language alone, a subset of the C language was introduced after the assembly language portion of the course. Much less time was spent presenting the basic concepts of C language programming than had been spent in the experimental course.

A goal-oriented approach was used to present the key foundational concepts of software interfaces in both languages, in order to produce a greater level of proficiency more quickly and efficiently than could be achieved with an exhaustive coverage of either language. With both the assembly and C languages, basic configurations were introduced at the beginning so that students learned to write code as early as possible. Simple example programs were provided in tutorials to promote this learning process.

Synergy of teaching assembly and C together

Both assembly and C can be presented in the same course when taught in the proper balance using a goal-oriented approach. Both languages can be learned in about the same time that teaching C alone would take. The assembly language was taught first in the course to provide a foundational understanding of processors and platforms that would accelerate the process of teaching C. The C language was added to provide a higher level view of the same processor concepts, further reinforcing the knowledge provided by learning assembly.

The goal-oriented approach utilized involved teaching a directed subset of C from a hardware perspective. The versatility of the C language allows it to be taught at various abstraction levels, beginning as a relatively low-level language and advancing to higher-level concepts as the students gain in understanding. C programming was taught from a hardware-centric perspective using practical examples. Object oriented programming principles were included by example. Topics usually considered as advanced techniques and traditionally presented at the end of a C language course– such as pointers, structures, unions and bit structures – were presented earlier in the course.

The manipulation of bits is generally the realm of hardware devices. The process of bit twiddling using techniques such as bit shifting and masking has traditionally been done in assembly language, and moving that code to C does not yield any benefits. However with the combined usage of bit structures and unions, this process is reduced to fairly straightforward code. Thus, the introduction of the constructs of pointers, structures and unions can reduce the tedium of dealing with the signals of connected hardware devices. Since bit structures can be platform-dependent, their usage is best restricted to lower platform dependent layers.

Teaching C *in addition to* assembly provides advantages that would not be provided by simply replacing assembly language with C. In either language, working at the device level requires becoming familiar with the processor and the address space. The concept of pointers must also be learned in either case (pointers in assembly languages may not be recognized as such in the same context as C). Pointers are the most difficult concept to learn in C. Teaching the concepts of pointers in assembly first, observing the instructions involved, and then translating that knowledge to implementation in C had its benefits. Once learned in assembly, the concept of pointers is easier to understand in C (the only difference is syntactic differences). Pointers are the primary reason that C can replace assembly language for device level code.

Other synergies between the assembly and C languages were observed in relation to understanding registers, processor architecture, and processor address space. In the C language, the introduction of the *register* keyword is difficult to understand relative to what usage it could have. After using assembly, it is easier to understand how it can effectively be used. Doing low (device) level microprocessor development in C is difficult to do without a good understanding of the processor architecture and the processors address space. This includes the program, data, stack and devices. In this view it can be argued that understanding the assembler for a processor before trying to do work with C is a definite advantage. This is why we have chosen to overlap the instruction of both the assembly and C languages.

Outcome and Future Direction

The Fall 2007 EE332/332L course was completed by 29 students, about two-thirds of whom were ECE majors. The students worked in teams to complete all of the lab assignments, including the final lab project.

Final lab projects were undertaken to consolidate and demonstrate the knowledge gained in the lecture and lab portions of the course. The students chose their own teams, which were allowed to propose their own projects and proceed upon approval by the instructor. Teams that did not create their own project were assigned one by the instructor.

For the final project, there were nine teams ranging from two to six students. All final projects were developed in the C language. The projects were:

- DE2 version of Space Invaders. On-board hardware utilized in the implementation included buttons (user input), VGA, LCDs (score display), timers and alarms (movement of aliens and weapons), and the JTAG UART (output for testing purposes).
- Client/Server with IRDA. This project utilized two FPGAs, a keyboard, IR transceiver, and LCD display.
- Motor Speed Detector and Regulator, utilizing PWM and an infrared emitter and detector sensor.
- Audio to LED Display.
- LCD Scrolling Marquee.
- DE2 version of Pong Game.
- Ping Pong. This project utilized VGA, sound and keyboard.
- Etch-A-Sketch.
- Bomb Squad. This project utilized two DE2s, keyboard, wireless modules, motors (remote vehicle), audio and LED display.

Each project was provided with a Nios II processor configuration (ptf and sof files). Since each project had different requirements and needs, the use of a soft processor allowed the instructor to create different configurations for each team. The teams were required to perform a demonstration of the product for the instructor, and produce a final project report describing their project. All projects were completed within the time frame provided with little help from the instructor.

Feedback was solicited from students completing the Fall 2007 Microprocessors course. The majority of the students rated the course positively overall, with a quarter of the students crediting the FPGA and soft processor for their positive response. Another quarter of the students specifically mentioned the value of learning the C language in addition to assembly. Negative comments focused on the amount of work required in the lab, particularly for just one credit, and the need for clearer overall course direction and expectations. Providing labs in advance to allow more time for preparation, and using a textbook that more closely related to the course progression were also mentioned.

Some of the computer engineering students expressed the opinion that students with a CS background (particularly those previously exposed to C in other classes) had an advantage in the course. However, this was not borne out by observations of the actual performance of the students in the course. The CS students did not necessarily have a better grasp of the C language at the level utilized in the Microprocessors course. With the exception of an operating systems course, the CS curriculum does not generally utilize the C language for low level purposes.

Several students commented that the C language was presented too quickly and that C should instead be a pre-requisite for the course. The abbreviated time spent presenting the basics of C did result in some confusion that necessitated readdressing some of the concepts (especially pointers, structures and unions). The material from Chapter 4 of K&R takes considerable time to cover, and does not progress concepts rapidly enough to allow introduction of new and more interesting devices with each lab.

A number of students felt overwhelmed and expressed their opinion that there was too much information for a single course. The sheer number of documents and corresponding volume of material that the students were expected to deal with was certainly a factor contributing to the students' sense of being overwhelmed. The handbooks and tutorials furnished by the vendor tended to be all-encompassing and more appropriately targeted to advanced courses. Use of these materials required the students to sort through a large volume of material beyond their understanding and the scope of the course, in order to extract a relatively small amount of information relevant to a junior level Microprocessors course.

Based on this experience, refinements are being made to the Microprocessors course that will be taught in the Spring 2008 semester. The following steps are being taken to improve the course:

- Develop tutorials for vendor products that are more appropriately targeted to the scope and objectives of this course, thereby reducing the volume of extraneous material and resulting confusion.
- Move some material out of lab and into lecture handouts, and utilize homework and quizzes to reinforce understanding and increase student accountability for learning. This will adjust workload-credit percentages. Lecture materials will be classified to emphasize

practical (lab-oriented) materials versus text (abstract) materials, which will allow introduction of some concepts in a different order than in the text in order to provide the necessary background for the labs to proceed.

- Revise the order of presentation of the C programming language from that outlined in K&R, in order to facilitate the transition from assembly language to C and continue introducing new devices in each lab. The concepts of pointers will be taught first, followed by structures, unions and bit structures at the beginning of the C language portion of the course.
- Adjust amount of time spent presenting the key concepts of the C programming language (somewhat more than 2007 Microprocessors course, but less than experimental course).
- Redesign labs to simplify future modification and maximize reusability. The concept is to divide the labs into two parts: one part that is instructional in nature, the other that represents the creative endeavor required of the students. The instructional portion can remain relatively unchanged from semester to semester, requiring updating only to accommodate changes in the hardware and tools used. The creative portion can be changed every semester to insure that students are exposed to new projects.
- Extend the use of the educational development tool Altera Debug Client) to continue challenging students, and delay introduction of more advanced tools (Nios II IDE) until later in the course after ISRs have been introduced.

Table 2 reflects the proposed modifications to the Microprocessor lab.

Table 2: Microprocessors Lab 2008 Outline

Week	**Topics/Assignment**
1	Familiarization with DE2, Nios II, and Debug Client (simplified tutorial)
2	Introduction to memory. (assignment: bubble sort routine)
3	Exploration of address space beyond memory, concept of memory mapped I/O. PIO devices: Interfacing to LEDs and switches
4	More advanced PIO (seven-segment). Integration of concepts from previous labs to implement display system using switches, LEDs, buttons and seven segment. Organization of code into modules and directories.
5	C language tutorial. Redo seven segment display in C, continuing use of Debug Client
6	LCD interface routines in C language, continuing use of Debug Client
7	Exceptions: return to assembly language to explore issues of exception processing
8	Hardware abstraction layer (HAL), introduction to Nios II IDE and related HAL facilities
9	HAL interrupts (abstraction of interrupts provided by Nios-supplied HAL routines)
10	*Spring break*
11	PWM and DC motor and H-bridge: use of PIO core to control direction and speed of DC motor with PWM
12-16	Final Project

Bibliography

S.M. Loo, “On the Use of a Soft Processor Core in Computer Engineering Education,” *ASEE* 2006.

http://www.terasic.com.tw

J. Kairus, J. Forsten, M. Tommiska, and J. Skytta, “Bridging the Gap Between Future Software and Hardware Engineers: A Case Study Using the NIOS Softcore Processor,” *33rd ASEE/IEEE Frontiers in Education Conference*, Boulder, CO, November 5-8, 2003.