Use of Discrete and Soft Processors in Introductory Microprocessors and Embedded Systems Curriculum

Sin Ming Loo
Electrical and Computer Engineering Department
Boise State University
Boise, Idaho 83725
208-426-5679
smloo@boisestate.edu

C. Arlen Planting
Electrical and Computer Engineering Department
Boise State University
Boise, Idaho 83725
208-426-4826
clarenceplanting@boisestate.edu

ABSTRACT
This paper describes a sequence of two courses, starting with the teaching of introductory microprocessor concepts and extending to advanced embedded system programming. The introductory microprocessor course is taught using a soft processor with a field-programmable gate array as the development platform, a combination which allows the course to undergo continual improvement without being limited by fixed hardware. The second course builds on the foundation of the first course, with an emphasis on working with advanced devices, building complete embedded systems, and developing embedded programming skills with different targets. This paper describes the experiences gained from the first course, and the detailed plan for the second course. This paper also describes which tools to include and which to leave out in the learning process for this process to be most effective from both the students' and instructor's perspective.

Categories and Subject Descriptors
K.3 [Computers and Education]: Computers and Education - General

General Terms
Experimentation

Keywords
Microprocessors, Soft Processor, Field-Programmable Gate Array, Curriculum

1. INTRODUCTION
Most microprocessors courses have traditionally been taught using a discrete microprocessor, such as Motorola 6800, Intel x86, ARM, or IBM PowerPC series [1]. The x86 platform has historically been the one utilized in the microprocessors course at Boise State University (BSU). The introductory microprocessors course at BSU taught only assembly language programming with little emphasis on other language skills.

The advent of field programmable gate arrays (FPGAs) and access to more powerful embedded processors has made it possible for students to tackle much larger projects than in the past. Increasingly sophisticated projects involving robotics, digital radio communications, MP3 players, video interfacing, and various sensors are more meaningful and exciting to the students, but also require a higher level of proficiency in programming. In our experience most electrical engineering students have learned to design hardware well, but lack the software skills to adequately demonstrate the functionality of that hardware. These skills would not only benefit students in advanced digital courses, but would also increase the students' future value in the workplace.

To address these issues, there has been an ongoing effort at BSU since 2004 to update the computer engineering courses. An integral part of every stage in updating BSU's core computer engineering courses involves the use of FPGAs in place of traditional development boards, taking advantage of the fact that the functionality of an FPGA can be changed without requiring physical changes to the board itself.

The endeavor started in the sophomore Digital Systems (EE230) course, with the major change being the introduction of a low-cost FPGA in place of the prototyping board with discrete components. The updated course has been very well received, and provides students an early exposure to reconfigurable hardware concepts. This sets the stage for the introduction of a soft-core processor in the microprocessors course.

Individuals familiar with FPGAs and soft core processors might assume that this approach would necessarily include teaching the entire suite of processor configuration tools, which could be overwhelming for both students and instructors. Key to the success of the microprocessors course update was the strategic decision to expose the students to only the processor program development tools, with the instructors responsible for usage of processor configuration generation tools. This enabled the students to concentrate on learning how to use a microprocessor rather than how to configure it.

Our next updating effort was the junior microprocessors course, with major updates including introduction of the C programming language, stressing the use of structures, unions, and pointers; use of a soft core microprocessor, and a sizable FPGA as the development target. The updated microprocessors course has also

Conference'08, October 23-24, 2008, Atlanta, GA, USA.

been very well received. Students indicated the heavy workload was worthwhile. This is the last course in our electrical engineering curriculum where students will interact with processors unless they select a system-level design project involving processors in their capstone senior design sequence.

Students have inquired which course they should take if they want to study advanced topics in microprocessors systems or embedded system design. Our curriculum has elective computer engineering courses in hardware description, system testing, and hardware/software co-design, but none as the natural follow-up to the new microprocessors course.

The remainder of this paper describes the experiences gained from teaching the first course (microprocessors) in the two-course sequence, and the plan for the second course (embedded and portable computing). Section 2 outlines BSU's electrical engineering curriculum and pre-update computer engineering courses. Sections 3 and 4 discuss the goals and resource selection for updating the course sequence. Section 5 provides details of the re-designed microprocessors course, including selection of which tools will be presented to the students. Section 6 presents our plan for the subsequent course. A summary and conclusions are presented in Sections 7 and 8.

2. EXISTING SITUATION

The Department of Electrical and Computer Engineering at Boise State University (BSU) offers an ABET-accredited Bachelor of Science in electrical engineering with computer engineering as an option. BSU also has an ABET-accredited Computer Science program, but does not have a separate computer engineering program. One of the core courses offered at Boise State University for students specializing in computer engineering is the Microprocessors course. The students take Microprocessors after they have taken Introduction to Computer Science (basic software skills and object oriented programming with Java) and Digital Systems (sophomore digital logic course).

The Microprocessors course at Boise State University covers microprocessor architecture, software development tools, and low-level hardware interfacing with emphasis on 16-bit and 32-bit microprocessor systems. Machine and assembly language programming, instruction set, addressing modes, programming techniques, memory systems, I/O interfacing, and handling of interrupts 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 (such as the magnetic card reader) that could be played with by the end of the course. The intent was to retain and potentially enhance this characteristic of the course with the changes implemented.

Since the microprocessors course (lecture and lab) 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. Most computer science students are more interested in hardware with an operating system. 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.

A course titled "Embedded and Portable Computing Systems", specifically addressing embedded design with the PIC microcontroller, has been offered at BSU. This was a primarily hardware-oriented senior/graduate level course utilizing assembly language only. Students taking this class received no C programming instruction.

3. GOALS

The goals of the updated two-course sequence were to more effectively teach the basics of microprocessor programming using updated technology, and to build on the foundation gained in the Microprocessors course to expand what is covered in the Embedded Systems course.

It was decided that the updated Microprocessors course would involve:

- A RISC microprocessor (MIPS-like)
- Simple memory-mapped devices (LEDs, switches, buttons)
- Initial use of assembly language to understand processors
- Transitioning the knowledge of microprocessors from assembly language to the C language [2]
- Coverage of topics such as polling, time management (delays vs. timer), interrupts and interrupt service routines (ISRs)
- Advanced devices, such as character LCD, pulse width modulated (PWM) DC motors and A/D conversion

The new follow-on Embedded Systems course would include:

- Advanced time management issues and usage of state machine construct in order to manage time
- Introduction of microcontrollers (specifics of memory and device and how they relate to programming)
- How coding can affect the ease of transferring code to other platforms (retargetability)
- Advanced devices (I^2C, SPI, USB, UART) from hardware and software perspectives
- Sensors and component interfacing (wiring)
- Use of test equipment to aid system development and debugging

The selection of resources to accomplish these goals is discussed in the following section.

4. SELECTION OF RESOURCES

4.1 Development Board and Tools

The two-course sequence was updated to utilize a soft processor instantiated on an FPGA. A board that had previously been used in graduate level courses at BSU - the Altera DE2 (shown in Figure 1) with Nios II processor - had most of the desired features, including:

- Classic RISC architecture closely approximating MIPS
- Variety of memory types (FPGA on-chip memory, SRAM, SDRAM, and Flash)
- Numerous attached devices plus two expansion headers for future add-ons (device support for USB, audio, VGA, Ethernet, UART, PS2, secure digital, and expansion headers)

In addition, the following are available from Altera for use with the development board:

- Free software integrating industry-standard development and debugging tools (e.g. GNU, Eclipse IDE, GCC compiler and GDB debugger) that students are likely to encounter in their careers
- Instruction set simulator (allowing work to be done at home) provided free with software tools from Altera

Best of all, the board is reconfigurable, allowing a different configuration for each lab and final project. A soft processor is very expandable with new interfacing hardware written in hardware description language. This feature allows the instructor to quickly create different configurations in order to easily meet different needs of various projects, and various courses. In addition, it gives an opportunity to demonstrate hardware/software co-design concepts in subsequent courses.

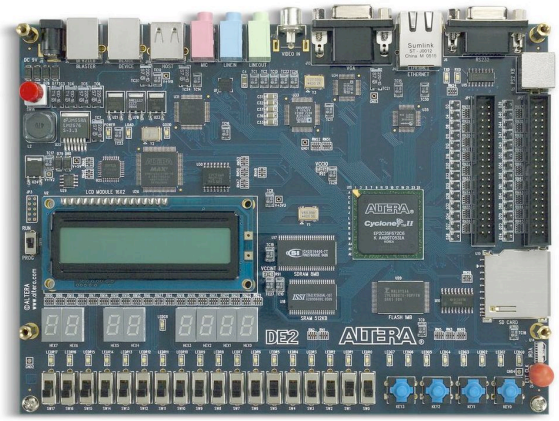

Figure 1. Altera DE2 Development and Education Board [3]

Since a soft processor is a microprocessor core that can be wholly implemented using logic synthesis, this provides the capability to expose the students to numerous different hardware configurations in a single course. When covering topics such as unimplemented instruction exceptions or the effects of working with or without cache, real instances can be demonstrated rather than only discussed from a theoretical standpoint. For example, a processor configuration with the hardware multiplier unintentionally omitted provided an object lesson in what is and is not supported in the hardware and how the processor handles unexpected behavior. (The usage of the div assembly instruction will result in an exception and send the program counter to exception address (0x20).)

Dedicated boards require that a connection resource be permanently allocated for specific purposes, thus limiting usage of the board. Conversely, a soft processor is analogous to a theater where a new stage set can be brought in for each new production. Each device/concept can be introduced with a unique processor configuration. For example, the concept of cache memory can be illustrated by using different configurations to generate a processor with cache and one without cache and comparing instruction performance and results.

As new devices become available or new instructional materials are developed, they can more easily be integrated into the course curriculum with a soft processor. If other devices in addition to those available on the DE 2 board are desired, that device or sensor can be put on an external board and installed on one of the IDE-like connectors. Any necessary hardware interface can be done in HDL (Verilog or VHDL) and this HDL design will directly connect to the Avalon bus which provides quick and easy access to the Nios II processor.

Unlike a discrete processor, all of the work does not need to be done weeks ahead of time and developed on a dedicated board for continued use 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 processor can grow or shrink as 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. Developing custom configurations for the course final projects is quick and easy.

In evaluating the use of FPGAs for the two-course sequence, several apparent disadvantages were addressed. Cost was one consideration. The general perception is that FPGA-based solutions are more expensive than utilizing a discrete board. However, this is not necessarily accurate when one factors in the savings provided by the reconfigurable aspect vs. add-ons necessary with a discrete board. (The Altera DE2 lists for $495, or $269 student price; BSU received a special academic discount to reduce the price to ~$150). The FPGA proved cost-effective when taking into account not only the initial cost of the discrete board, but the total cost with add-ons to provide comparable functionality.

Other apparent disadvantages had to do with the processor configuration, including the need for 1) extra instruction to understand reconfigurable aspects (higher abstraction level, need to instantiate every time), and 2) the creation of processor configurations. In the lower level courses, creation of processor configurations was done by the instructor. Though this requires additional effort on the part of the instructor, the capability to change the functionality an unlimited number of times to fit the desired application provides a distinct instructional advantage. Also, once a configuration is done it is reusable for future semesters.

Though the soft processor was considered the best instructional platform, it is acknowledged that it still needs occasional comparison (via demonstration) to traditional discrete microprocessors.

4.2 C Programming Language

Since the C language is the choice for implementation of today's dominant operating systems [4], including Windows and Linux [5], it seemed prudent to follow suit with this precedent. Though C++ has more features than C, these features are not particularly useful in teaching microprocessors and do not offset the fact that it is more difficult to learn than C. In addition, many small microcontrollers (such as the Microchip PIC) are not supported by C++ compilers.

5. MICROPROCESSORS COURSE WITH SOFT PROCESSOR

This course needed revamping to become more representative of what practicing computer engineers deal with on a daily basis,

including embedded systems without operating systems. The ultimate objectives are to update the course using a modern development environment with modern debugging capabilities, to teach the basics of microprocessor programming (assembly plus C), and to have the students practice these skills with realistic laboratory assignments and projects.

The Microprocessors course is a one-semester course consisting 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 once a week. The rest of this section contains details of our course update and implementation, including educational methodology, course details, and outcomes.

5.1 Approach

The primary changes in the updated Microprocessors course involved selection of a MIPS-like processor implementation (RISC architecture vs. the CISC platform previously used), teaching the C programming language in addition to assembly, and utilizing a modern development platform and tools.

Although CISC x86 is the most prevalent microprocessor architecture, this doesn't necessarily equate to the best educational platform for teaching basic microprocessor concepts. The instructions for CISC platforms are variable in length, which complicates learning compared to the fixed length instructions for RISC platforms. As many of the instructions available with the CISC are not applicable to the basic microprocessors course, the significantly smaller set of instructions provided for a RISC platform was considered more appropriate. In addition, the students will also use RISC when they take the Computer Architecture course.

Introduction of the C programming language to the Microprocessors course was proposed because of its ability to enhance productivity and portability with minimal overhead. The course would introduce just enough material from the C programming language that students could work with devices at a low level. This would minimize the overlap for the computer science students, and 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 (taught in Spring 2007) to investigate methods of incorporating the C 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.

5.2 Resources Utilized

5.2.1 Course References

Textbooks are available for most traditional microprocessor platforms. However, since the concept of teaching soft 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 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.

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. The textbook "Embedded System Design - A Unified Hardware/Software Introduction" [6] was selected as the text for the lecture portion of the course. This text has been dropped in our latest offering of the course, and replaced with inhouse-generated materials. "The C Programming Language" (K&R) [7] was the primary reference for the C language portion.

5.2.2 Development Platform and Tools

A microprocessor was considered a more generalized platform better suited for teaching microprocessor basics than microcontrollers, which have a wider variety of implementations. (Microcontrollers will be addressed in the Embedded Systems course.). In addition, the Nios II processor has an architecture which, when coupled with memory-mapped I/O, simplifies understanding of the system's address space. The Harvard architecture typically found on microcontrollers requires special C qualifiers to identify data residing in different memory spaces.

The tools provided by Altera include tools to produce hardware configurations, and tools to develop software solutions for soft processors. The tools involving the generation of soft processor configurations 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 (Nios II IDE). Altera Debug Client provides assembly of assembly language programs with no run time support, and compilation of C programs with minimalist run time support (no interrupt service routine or exception handling). 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 advantageous for educational purposes.

Altera Debug Client is ideal for first time exposure to working with embedded systems, but lacks the facilities for advanced development and debugging. It also lacks simulator capabilities that would allow its use for homework assignments, providing students the freedom for exploring microprocessor concepts outside of the lab.

The Nios II IDE automatically generates software libraries (Hardware Abstraction Language, or HAL) to support most of the devices generated with a particular hardware configuration. While very convenient for advanced users, this hinders the learning process for beginners. For this reason we decided to start

with Altera Debug Client for the first part of the course before introducing the Nios II IDE. Using 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 development tool was used in the course 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 (interrupt service routines, or ISRs). (Altera's Nios II IDE is based on the popular Eclipse IDE framework with Altera-supplied plug-ins that 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 altogether, and students wouldn't need to learn two development environments.)

5.2.3 Limiting Scope of Tool Usage

Unless decisions are made to limit which tools are taught, the number and complexity of tools required for teaching the microprocessor course utilizing a soft-core processor with an FPGA as target platform could quickly overwhelm the students and instructors. As shown in Table 1, the instructor will use all four major development tools while the students will use just two. It is critical not to overload the students with extra tools that will make the learning process much more complicated that it needs to be.

Table 1. Tools usage

Development Tool	Used by Instructor	Used by Student
Altera Quartus II	Yes	No*
SOPC Builder	Yes	No
Altera Debug Client	Yes	Yes
Altera Nios II IDE	Yes	Yes

*only to download *sof* file

The Altera Quartus development suite is a software tool for designing and debugging FPGA designs. The input can be schematics capture or HDL (Verilog or VHDL). Though this tool is usually used in digital design or higher level FPGA system design courses, the students in the Microprocessors course do not need to use this tool. However, the FPGA will still need to be programmed. The instructor will use SOPC Builder to configure the Nios II processor, and will synthesize it using Quartus II to generate the *sof* file for student use in the lab.

Through experience, it has been found that starting from a simple Nios II configuration with Nios II core, JTAG_UART, and SRAM before other devices are added will be most fruitful.

The reason why we do not have our Microprocessors students use the processor configuration generation tools (Altera Quartus II and SOPC Builder) is because they should concentrate on learning how to use a microprocessor rather than how to configure one! (However, the students will be exposed to those tools briefly at the end of the Microprocessors course, and students in the second course of the sequence will learn how to use the tools to configure a microprocessor.)

As for Altera Debug Client and Altera Nios II IDE, these tools will be used extensively by the instructor and the students. The Debug Client is used first because it is simple (less complex than the industrial strength tool), it is good for assembly or C programming (individually but not combined), and it supports hardware debugging. However, it does not provide simulator support (which is more appropriate for homework assignments). For this reason, the transition to Nios II IDE is made as quickly as possibly.

Once the students are comfortable with the basic development concepts, Nios II IDE is brought in for the course. It has features that any practicing engineer would use in the field.

5.3 Course Details

5.3.1 Course Approach

A goal-oriented approach was used to present key foundational concepts in both languages, in order to produce a greater level of proficiency more quickly 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 (in the first lab assignment). Simple example programs were provided in tutorials to promote the learning process.

An exhaustive presentation of the C programming language was not the goal of this course. Presenting problems and solutions with assembly language and then re-solving those problems utilizing the C language provides an alternate method of instruction that can be termed as goal-oriented. This approach can greatly reduce the amount of time and effort for both students and instructors.

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 parallel I/O (PIO) such as 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 our belief that it isn't necessary to teach the entire C language to significantly enhance software skills beyond those 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 embedded systems programming course.

A course outline is presented in Table 2.

The labs developed for the updated Microprocessors Lab course are summarized in Table 3. 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, with the exception of a portion of the ISR lab, dealt primarily with the C language.

Table 2: Updated Microprocessors Course Outline

Week	Lecture Topics
1-3	Nios II Processor System Architecture and Programming Memory, Registers, Program counter Assembly Instructions, Memory organization, Addressing modes Assembler Directives, Instruction Set Reference, Instruction encoding/decoding
4-6	The C Programming Language: K&R Chapters1-4 C Program structure, Pointers, Structures, Unions, Bit structures C access to devices Cache bypass in C Inline assembly
6-7	Exceptions
7-8	Hardware abstraction layer (HAL)
8-9	Devices: Timers, counter, watchdog timers UART, PWM
10-15	Keypad, Keyboard, Analog to digital, Real time clock, LCD controller, Memory controllers Performance measurement, ISR performance, Simple bus, Communication protocols

Table 3: Updated Microprocessors Lab Outline

Week	Topics/Assignment
1	Familiarization with DE2, Nios II, and Debug Client (simplified tutorial)
2	Introduction to memory; develop 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. 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	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

5.3.2 Synergy of teaching assembly and C together

Exposure to assembly is required for the Computer Architecture course. However, it was our belief that the addition of the C programming language would provide additional benefits. Both assembly and C can be presented in the same course when taught in the proper balance using a goal-oriented approach. 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. Assembly language is the best way to understand and learn the foundations of microprocessors, since it is the primary interface to the processor. 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 early in the course.

Bit manipulation is one concept that can benefit from the introduction of the C language. 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 made it easier to understand the concept of pointers in C. Once pointers have been learned in assembly, the only differences that need to be learned in C are syntactic. Pointers are the primary reason that C can replace assembly language for device level code.

The following four figures demonstrate the object oriented aspects of the microprocessors course. Figure 2 shows the top level of a simple general-purpose input/output (gpio) program and the abstractions with layering utilized. Figure 3 is a structure definition for a memory mapped I/O based device and includes a definition for a cache override. Figure 4 shows the methods for data encapsulation with accessor methods (get_RUN and get_POS). Figure 5 shows usage of a show variable to match the state of the output only port so that individual signal may be updated independently. Mutator methods (show_RUN and show_POS) are provided.

```c
// file: shadow.c

#include "switches.h"
#include "ledr.h"
#include "types.h"

int main()
{
    bits POS = 0;
    bits RUN = 0;

    LEDR_Init();

    while (1)
    {
        show_RUN( RUN = get_RUN() );
        show_POS( POS = get_POS() );
    }

    return 0;
}
```

Figure 2. Simple gpio Programming with Layering

```c
#ifndef PIO_H_
#define PIO_H_

#include "types.h"

#define NOCACHE 0x80000000

typedef struct pio_regs {
    word data;
    word direction;
    word interruptmask;
    word edgecapture;
} PIO_REGS;

#endif /*PIO_H_*/
```

Figure 3. Structure Definition Memory Mapped I/O Device

```c
// file: switches.c

#include "system.h"
#include "PIO.h"
#include "switches.h"

static volatile PIO_REGS *SW =
         (PIO_REGS *)(SWITCHES_BASE | NOCACHE);

static  union {
            word    data;
            struct {
                bits POS    : 3;
                bits fill_1 : 14;
                bits RUN    : 1;
                bits unused : 14;
            } bits;
        } SH_SW;

bits get_RUN ( void )
{
    SH_SW.data = SW->data;

    return SH_SW.bits.RUN;
}

bits get_POS ( void )
{
    SH_SW.data = SW->data;

    return SH_SW.bits.POS;
```

Figure 4. Data Encapsulation with Accessor Methods

```c
// file: ledr.c

#include "system.h"
#include "PIO.h"
#include "ledr.h"

static volatile PIO_REGS *LEDR =
          (PIO_REGS *)(LEDR_BASE | NOCACHE);

static  union {
            word    data;
            struct {
                bits fill_1 : 6;
                bits POS    : 3;
                bits fill_2 : 5;
                bits RUN    : 1;
                bits fill_3 : 3;
                bits unused : 14;
            } bits;
        } SH_LEDR;

void LEDR_Init ( void )
{
    SH_LEDR.data = 0;
    LEDR->data = 0;
}

void show_RUN ( bits RUN )
{
    SH_LEDR.bits.RUN = RUN;

    LEDR->data = SH_LEDR.data;
}

void show_POS ( bits POS )
{
    SH_LEDR.bits.POS = POS;

    LEDR->data = SH_LEDR.data;
}
```

Figure 5. Show Variable

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.

It should be noted that the intent was not to write C code as translated assembly, which is hard to read and maintain and offers little benefit over assembly code. By effectively utilizing the facilities of the C language, many assembly language routines can be reduced to very small and elegant solutions. Writing code at the lowest level to access devices is generally very tedious in either language, but by providing appropriate abstractions this code can be isolated in layers to allow the higher level more freedom to solve problems with less consideration for hardware details. This also provides for easier retargeting to other platforms.

5.4 Course Outcome

Final lab projects are undertaken to consolidate and demonstrate the knowledge gained in the lecture and lab portions of the course. For the final project, the students choose their own teams ranging from two to six students (depending on the complexity of the project). The teams are allowed to propose their own projects and proceed upon approval by the instructor. Teams that do not create their own project are assigned one by the instructor. All final projects are developed in the C language.

Each project is provided with a Nios II processor configuration (*sof* and *ptf* files). Since each project has different requirements and needs, the use of a soft processor allows the instructor to create different configurations for each team. The teams are required to perform a demonstration of the product for the instructor, and produce a final project report describing their project.

The final projects successfully demonstrated the students' grasp of the knowledge presented in the course. A wide range of devices has been utilized in the final projects, including:

- Interface to student-built joystick to accompany VGA-based game
- Interface to Super Nintendo game controller for game project
- Two-way infrared communications
- irDA device controller to DE2 interface
- 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 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 utilized VGA, sound and keyboard,
- Etch-A-Sketch, and
- Bomb Squad -- utilized two DE2s, keyboard, wireless modules, motors (remote vehicle), audio and LED display.

All projects have been completed within the time frame provided with little help from the instructor. Based on student evaluations of the course, the course update was considered successful.

6. EMBEDDED SYSTEMS COURSE WITH SOFT AND DISCRETE PROCESSORS

6.1 Objectives

The planned second course in the two-course sequence will build upon the introductory (microprocessors) course with advanced topics. The goal is to provide students with equally strong software and hardware backgrounds, such that they can develop systems that run reliably and efficiently.

The first objective is to incorporate both soft and discrete processors in this course. One might ask, if the path of progression is heading toward the use of soft processors in FPGAs, wouldn't the use of a discrete processor be taking a step backwards? That's a reasonable question to ask and one that has no absolute right or wrong answer. Discrete processors typically offer high performance and often exclusive special capabilities that can't be totally replaced or matched by a soft processor counterpart. The other reason to include discrete processors is that it is beneficial for the students to be exposed to another processor in order to learn to write code on one platform that is appropriately layered to port easily to another platform.

The second objective of this course is to teach embedded systems programming considerations, and object oriented programming with C. The course will include teaching layered, modular programming concepts and selected object oriented programming principles applicable to embedded systems [8, 9]. It will also provide exposure to abstraction interfaces of varying quality so the students will gain the finesse to recognize and create an effective hardware abstraction interface. The course will selectively implement object oriented programming principles applicable to small embedded systems.

The final aim is to bring the first two objectives together by providing opportunities to practice using real-world devices. Sensors with different communications interfaces will be brought in as programming assignments and to be used in projects. Sensors that are interfaced using current, voltage and serial protocols such as UART, SPI and I^2C will be part of the curriculum. Since a soft processor is part of the curriculum, hardware/software codesign concepts can also be introduced. If the schedule allows, a small project may be assigned to explore pure software implementation, pure hardware implementation, and an implementation taking advantage of synergism between hardware and software.

6.2 Implementation

The partitioning of time spent on teaching hardware versus software has been given much consideration in the two-course sequence. Software concepts can be categorized as 1) language skills, 2) device algorithms, and 3) operating system issues. As shown in Table 4, not only does the percentage of time spent on software differ between the two courses but the software topic emphasis also varies significantly. Compared to the Microprocessors course, the focus of the software concepts presented in the Embedded Systems course is less about language skills and more related to device interfacing and operating system issues.

The new Embedded Systems course will begin with a few of the more advanced concepts of the C programming language not specifically covered in the Microprocessors course, including object oriented programming in C, layering, race conditions, and cooperative and pre-emptive multiprocessing. Those topics will be introduced in conjunction with a case study of UART (further described below). Foundational skills developed in the first half of the course will then be employed in the second half to work with I^2C, SPI and USB platforms.

In object oriented languages such as Java, data and code are tightly coupled. Conversely, in a structured or procedural language such as C, data and code are uncoupled. To use C in object oriented programming necessitates that data and code be loosely associated (by means of establishing coding and naming conventions) to approximate an object oriented language. Making this transition requires an understanding of the concepts of data abstraction and encapsulation, and C. Achieving the goal of

encapsulation and data hiding is accomplished by concentrating on the appropriate usage of the static key word, and providing functions to emulate the functionality of accessor and mutator methods. (The interpretation of "object oriented" concepts for this course doesn't address concepts such as polymorphism, inheritance, etc.)

Table 4. Hardware/Software Breakdown in Two-Course Sequence

	Microprocessors Course	Embedded Systems Course
Software		
Language Skills	50%	5%
Device Interfacing	10%	30%
Operating System	5%	20%
Hardware	35%	45%

Use of the Altera DE2 for prototyping purposes will continue in this course. Students will tackle similar projects implemented on different target platforms, followed by a review of issues found with each platform. Students will select the platform for their final project early in the course.

This course will target small embedded processors without operating systems, requiring students to develop the code for the services an operating system would normally provide. Concepts typically covered in an operating systems course that are applicable to embedded systems will also be addressed, including time management, solutions to address concurrency issues (race conditions [10]), and communications protocols. Since these issues would typically be handled by an operating system in the case of general purpose computers, different approaches are necessary for small embedded systems [11-16].

Advanced time management issues and usage of state machine construct in order to manage time will be addressed. This approach allows for multiple threads of execution to be accomplished in a cooperative manner. (To allow for accurate measurement of time, this will be used in conjunction with an external crystal and an interrupt service routine.) A cursory introduction to pre-emptive context switching (pre-emptive multi-tasking) will also be included.

A case study of serial communications (UART) will also be presented in the course, incorporating RX/TX, interrupt handling, operating system concepts (issues of concurrency with ISRs), data structures (circular buffers), network protocols in SLIP, and low level device access. Additionally, the solution to the problems encountered in the case study can be structured utilizing layering techniques which go hand-in-hand with encapsulation methods.

The serial communications case study will show a classic device algorithm and provide insight into the measures taken to ensure reliable usage of a device. When tackling the later devices (such as I^2C) the students will brainstorm to come up with possible approaches, then experiment with methods for a short period of time, and regroup to compare results. Once the routines are completed, the class will again review and evaluate results.

The course will be project-oriented, with all projects developed on the Altera DE2 development board and retargeted to other platforms. Code for small embedded systems written on one platform with the intent of porting to another is generally more appropriately layered, which results in well-written code that is inherently retargetable.

The initial project will be development of an ultra-light menu system for embedded applications. This project is intended to reacquaint the students with the Altera DE2 and tools used in the Microprocessors course, and test their understanding and skills using pointers, structures and unions. This should be a very small efficient menu, intended not for dealing with everyday processes but for infrequent updating of configuration items or as a debugging tool. The routine will be a passive component, ready to be used but with no continuous impact on system performance.

The second project will demonstrate a stepwise refinement approach to designing a large project by starting with smaller pieces that are designed and tested separately before being integrated into a final solution. The intent is to illustrate that many pieces of code can coexist and run cooperatively as a whole without needing an operating system to manage the pieces. Components will include those from the Microprocessors lab projects (the majority implemented with minimal amounts of code), the ultra-light menu developed in the first project, and an Altera alarm abstraction that will be provided.

The third project will involve retargeting the comprehensive project developed above to various other platforms. The remaining projects will include a real-time clock (I^2C), other I^2C devices, and SPI devices. The final project will involve a platform containing a USB device.

7. SUMMARY

To date, the updated Microprocessors course has been taught three times and the course has been refined based on experiences in each preceding semester. For example, the coverage of C programming language concepts has been reduced and the order of presentation revised to facilitate the transition from assembly language to C (pointers, structures, unions and bit structures were taught at the beginning of the C language portion of the course). Tutorials for vendor products were modified to reduce the volume of material beyond the scope and objectives of this course that students were required to sort through. The use of homework and quizzes was increased to reinforce understanding and increase student accountability for learning.

Some material was moved out of the lab portion and into lecture handouts, which adjusted workload to better match the credits for the lecture/lab portions of the course. Lecture materials were classified to emphasize practical (lab-oriented) materials versus text (abstract) materials, allowing introduction of some concepts in a different order than in the text in order to provide the necessary background for the labs to proceed.

The labs were also redesigned 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.

In the most recent offering of the course, smaller more frequent homework assignments and labs have been successfully used to cover the same material in a manner that seems to be less overwhelming to the students. Course handouts and examples have been increased to help students more thoroughly grasp the concepts. Quizzes and tests have been updated to better reflect the increased expectation of student understanding.

The upcoming version of the Microprocessors course will also include a refresher section on number system concepts and an early test of programming skills.

With the updated Microprocessors course as a foundation, less time will need to be spent on the C programming language in the embedded systems course. This will leave more time in the new Embedded Systems course for specifically tackling embedded programming for devices and protocols such as the UART, I^2C, and SPI.

8. CONCLUSION

A soft processor instantiated on an FPGA with classic RISC architecture was used to provide a modern development environment in the updated Microprocessors course at Boise State University. Industry-standard development and debugging tools (Eclipse IDE, GCC compiler and GDB debugger) that the students are likely to encounter in their careers were also incorporated in the course. A combination of assembly and C language was used to teach the basics of microprocessor programming, and the students learned to practice these skills with realistic laboratory assignments and projects. The planned Embedded Systems course will provide the natural follow-up to the updated microprocessors course.

The update process for the computer engineering courses at Boise State University has been fruitful for students and instructors. Students get to learn modern design techniques with up-to-date tools, beginning with the introductory Microprocessors course and continuing into the Embedded Systems course.

9. REFERENCES

[1] S. M. Loo, "On the Use of a Soft Processor Core in Computer Engineering Education," *Proceedings of 2006 ASEE Annual Conference*, Chicago, IL, June 18-21, 2006.

[2] G. Skelton, "Introducing Software Engineering to Computer Engineering Students," *Proceedings of the 2006 Southeast Conference,* 0-4244-0169-0/062006 IEEE.

[3] http://www.altera.com/education/univ/materials/boards/unv-de2-board.html, Visited: December 1, 2008

[4] G. J. Nutt, 2003. *Operating Systems,* 3rd ed. USA:Addison-Wesley.

[5] A. Silberschatz, P.B. Galvin, and G. Gagne, 2005. *Operating System Concepts,* 7th ed. Hoboken, NJ: John Wiley and Sons, Inc.

[6] F. Vahid and T. Givargis, 2002. *Embedded System Design – A Unified Hardware/Software Introduction,* Hoboken, NJ: John Wiley and Sons, Inc.

[7] B.W. Kernighan and D.M. Ritchie, 1988. *The C Programming Language,* 2nd ed. Upper Saddle River, NJ: Prentice Hall.

[8] M. Curreri, "Object-Oriented C: Creating Foundation Classes Part 1," Available: http://www.embedded.com, *Embedded Systems Design, 9/10/03*.

[9] C. Cantrell, "Embedded Object-Oriented Programming," *Circuit Cellar,* Issue 187, Feb. 2006, pp. 52-59.

[10] D.P. Reed and R.K. Kanodia, "Synchronization with Eventcounts and Sequencers," *Communications of the ACM,* vol. 22, no. 2, Feb. 1979.

[11] K.G. Ricks, W.A. Stapleton, and D.J. Jackson, "An Embedded Systems Course and Course Sequence," *Proceedings of 2005 Workshop on Computer Architecture Education,* Madison, WI June 5, 2005.

[12] D.J. Jackson and P. Caspi, "Embedded Systems Education: Future Directions, Initiatives, and Cooperation," *ACM SIGBED Review,* Volume 2, Issue 4, October 2005.

[13] F. Vahid, "Embedded System Design: UCR's Undergraduate Three-Course Sequence," *2003 IEEE International Conference on Microelectronic Systems Education,* Anaheim, CA, June 1-2, 2003.

[14] J. Conrad, "Introducing Students to the Concept of Embedded Systems," *International Conference on Engineering Education,* Gainesville, FL, October 16-21, 2004.

[15] T.S. Hall, J. Bruckner, and R.L. Halterman, "A Novel Approach to an Embedded Systems Curriculum," *36th ASEE/IEEE Frontiers in Education Conference*, San Diego, CA, October 28-31, 2006.

[16] A. Striegel and D.T. Rover, "Enhancing Student Learning in an Introductory Embedded Systems Laboratory," *32nd ASEE/IEEE Frontiers in Education Conference*, Boston, MA, November 6-9, 2002.