Soft processors for microcontroller programming education

Charles Goetzman
Computer Science
University of Wisconsin – La Crosse
goetzman.char@uwlax.edu

Jeff Fancher
Electronics
Western Technical College
FancherJ@westerntc.edu

Abstract

This paper will document the development of a microprocessor platform needed for teaching basic assembly and C programming that would integrate well within the lab environment at the technical college level. A variety of platforms were analyzed and will be briefly compared in terms of capability and usability for the educators and students. Particularly the use of soft processors and system-on-chip platforms both from the educator and student's perspective will be explored.

Details will be presented on a developed system using an Altera FPGA platform with a synthesized NIOS II MIPS-like soft processor. An overview will be given of some of the compatible development boards, along with information on the custom interface hardware developed for the lab. The lab coursework produced will also be briefly discussed, followed by results and performance after using the system with students of a variety of programming experience levels.

Background

When developing a course on embedded system or microcontroller programming, there are myriad platforms that can be considered. Along with specific hardware requirements, the development environments and their limitations must be carefully considered, and all the possible configurations can be daunting. There are also challenges involved in selecting an environment to match a variety of student programming experience levels.

Course

Western Technical College offers an introductory microprocessor course as part of its Electrical and Computer Engineering Technology program. The primary goals of the course are to provide introductory programming and hardware/software interface skills along with basic knowledge of microprocessor architecture. The course consists primarily of lab work involving developing for microcontrollers, with lectures providing key programming concepts and system information.

The previous development platform was a 68hc11 system with limited debugging capability and a fairly limited c compiler and standard library. The desire for an updated development environment and gradual attrition of the previous lab hardware led to a complete redesign of the course's platform.

Requirements

The development environment should be free for the students to install, or at least with no significant limitations. Something built around one of the major IDEs (Eclipse, VS, Netbeans) would be preferable. The overall environment should support full debugging including step-by-step execution preferably down to the dis-assembly level.

The hardware needs to be able to survive in the lab environment long-term. Many development boards are simple bare boards without real IO protection or enclosures. Having a reasonable collection of IO (switches, LEDs, etc...) for basic labs along with expandable IO for students to explore their own interface circuits is ideal.

Having a reasonably low cost is advantageous in and of itself, but also below a certain point makes it practical to rent the hardware to students. Due to the nature of almost any lab involving hardware peripherals, simulation has limited value. Giving the students the ability to use the actual hardware at home frees up lab space during the day as well.

If there are other courses involving programming, obviously reusing the same systems is good for both the school (cost) and students (familiarity). The hardware could have uses in even non strictly programming courses as well, such as project courses and digital logic type courses.

Platform rundown

8 and 16 bit flash microcontrollers

This includes most common microcontrollers like the PIC, AVR, MSP, 805x, and 68hcxx families. There are a large variety of development environments, some built on standard base IDEs like Eclipse and Visual Studio, and some totally proprietary. Usually these chips have rather cheap programmers available, but debuggers are often fairly expensive. Many of the more affordable boards are little more than breakouts with no IO protection and limited features like switches or displays or debugger connections. Some of the platforms can be extremely cheap to the point that students could purchase their own as part of the required course materials, however.

The very limited memory and storage of these processors also limits the availability of standard library functions for the C compilers. Any use of floating point math, stdio, or malloc often eats up large amounts of the available memory very quickly. And when these sorts of functions are available they are often limited in various ways.

32bit ARM processors

There are a growing number of more powerful ARM microcontrollers available as well. These overcome most of the limitations of smaller microcontrollers by generally having much larger available memory, full standard libraries, and standard JTAG debuggers supported via GDB.

However, outside of proprietary limited development environments, most of the processors that do have freely available GNU tool chains, they are usually fairly complex to setup, particularly if one wishes to support windows without requiring cygwin. The hardware and peripherals of these microcontrollers are generally complex as well. Even the most simple IO can require several steps of setup and initialization.

Soft processors

All of the main FPGA manufacturers have available synthesizable processor cores. The processor cores themselves are mostly similar to 32bit ARM or MIPS processors and have similar instruction sets and straightforward JTAG debugging support. But the extent and complexity of the peripherals is completely customizable as expected.

Some of these systems at least are built around standard Eclipse/gcc/gdb build and debug chains. And since these companies are in the business of selling FPGAs they already are receiving in effect a per-end-user license, and thus all of the development tools are

completely open and free.

Soft Processor Platform

To meet these requirements, we ultimately chose and developed a FPGA-based development platform based on the Altera NIOS II soft processor. At it's core, the system consists of a fairly straightforward implementation of a NIOS II processor built in Altera's SOPC Builder tool. All of the typical components are present from the CPU core, sdram controller, IO ports, SPI serial master, timers, and pulse width modulators.

Development for the system is performed in the NIOS II EDS application, which is a customized version of the Eclipse IDE using the gcc compiler and the NewLib standard library. There are no limitations on compiler optimizations or code size. The Altera USB Blaster JTAG fully supports GDB so the Eclipse debugger works including Tracing.

For the lab coursework developed, one of the simpler boards was chosen, the Terasic DE1, which has some standard switches, LEDs, seven segment displays, VGA, SD-card, audio, and IO expansion. The board also has 8MB of SDRAM used as the main system memory by the NIOS II processor. The board is relatively affordable and fairly rugged and thus can be rented to students via the school's textbook/materials rental program.

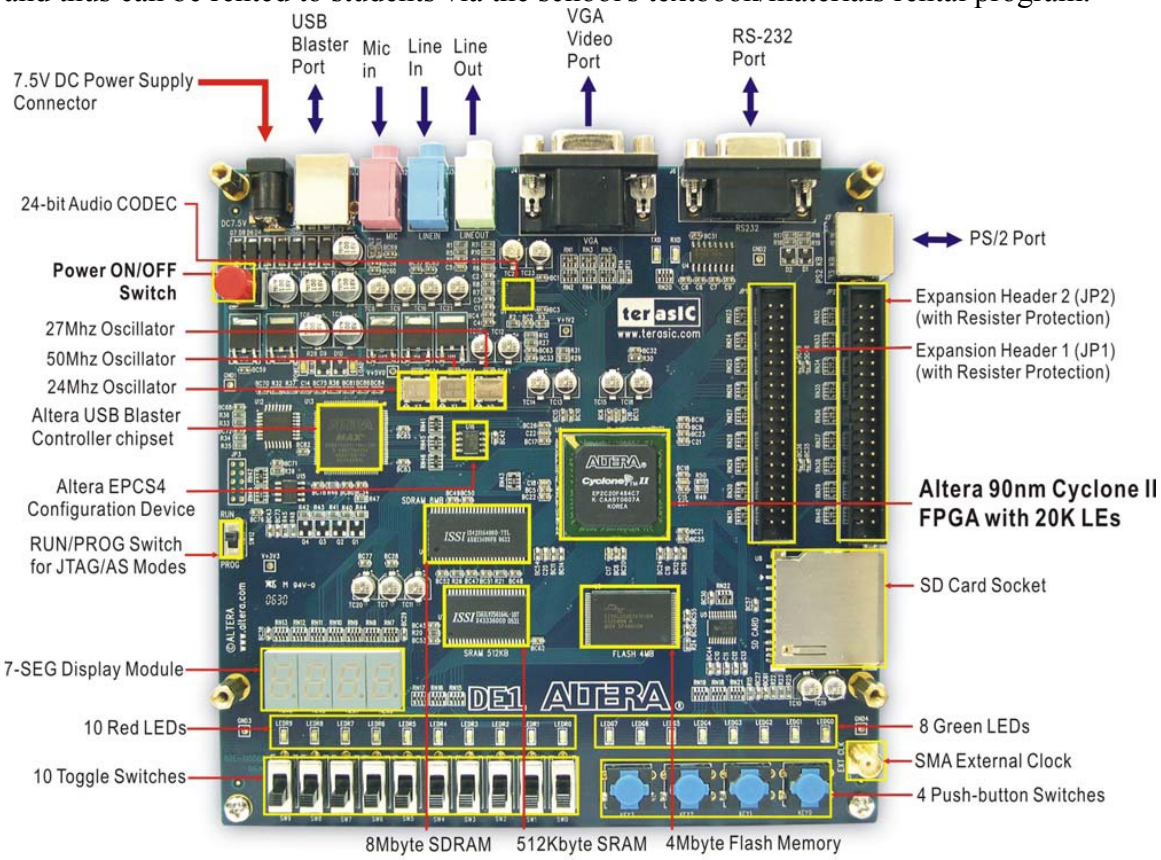

Figure 1 : The Terasic DE1

There are a variety of other boards of varying capability available. Some like the DE0-Nano are about the size of a credit card yet have only IO headers, and some like the DE2 series have much more advanced peripherals like real VGA DACs, Ethernet, and USB master support. Only very small adjustments to the system, mainly around the memory controller and IO mapping is necessary to adapt to these differing platforms.

Circuit design for a number of needed extra interfaces is already complete, including stepper motor drivers, PWM motor drivers, tachometer, and ADC, all fully opto-isolated. After the circuits being tested in class, a printed circuit board design is being developed to implement the interface. Although there is value in having the students build and carefully examine the various interfaces at least once, after that the time savings in having the circuits readily available are significant.

The use of an FPGA also greatly simplified the design of the hardware interface. Different types of labs will necessitate using the same hardware resources in slightly different ways. For instance depending on the intent, a driver employing pulse width modulation may use the PWM signal as an enable when driving a motor for speed control, and as the polarity when driving an RC or LC filter to produce precise voltages. Similar to how most microcontrollers reuse IO pins, we were able to reroute signals as appropriate inside the FPGA to support all the desired modes of operation of the hardware. And of course, future expansion or modification is simple.

Course structure and lab work

The microprocessor course at Western Technical College has an unusual set of requirements because the Electrical and Computer Engineering Technology program offers transfers into both Computer Science and Electrical Engineering programs, so there are competing interests in what materials are presented and emphasized. Combine this with the fact that some students may have had no previous programming experience whatever, and it becomes even more challenging.

The overall learning goals are nothing unusual for an introductory microprocessor course, from architecture, assembly language structure, typical program structures (branches, loops, subroutines), IO, and interrupts. But the Computer Science transfer option wants an emphasis on architecture and assembly as a theoretical backing, while the Electrical Engineering transfer is more focused on practical development and implementation in C.

The course was developed to introduce the processor architecture, memory, and IO structure using assembly language, up through basic branching and looping structures as well as the stack. Then to transition into C for the remainder, when interrupts, more complex flow control, libraries and feedback control loops are discussed.

The format of the course was structured as a 2 hour block of lecture and a 2 hour block of lab per week. The topic was discussed during the lecture, followed by a brief 'mini-lab' to familiarize the students with the topic and go through some examples together. The labs

then usually consisted of applying the programming concept learned to a real world type of hardware/software problem.

Week	Lecture/Lab	Lab Assignments
1	Introduction to Course Handout	Lab # 1 Building a project in NIOS IDE
2	Instructions and Registers	Lab # 2 Math and Logic operations.
3	Memory, Addressing and I/O	Lab # 3 I/O using PWM
4	Looping and Branching.	Lab # 4 Looping
5	Advanced branching	Lab # 5 Stepper motor
6	Pointers and Arrays	Lab # 6 Look-up tables and arrays
7	Stack and Subroutines	Lab # 7 Subroutines
8	Midterm Exam (Lecture)	Midterm Exam (Lab)
9	Introduction to C programming	Lab # 8 Building a C project.
10	If statements and Expressions	Lab # 9 Library routines.
11	While and For loops	Lab # 10 A/D conversion.
12	Functions	Lab # 11 Advanced Debugging & Tracing
13	Interrupts	Lab # 12 Interrupts
14	Program organization	Lab # 13 Feedback control loops
15	Catch-up/Review	Catch-up/Review
16	Final Exam	Final Exam

Table 1 : Course outline

The system developed greatly simplified many of the early labs. Many standard microcontrollers have exceedingly complicated peripherals to support any conceivable mode of operation. For example, most small flash microcontrollers have combined Timer/Counter/PWM modules with a dozen (or more) registers. Whereas with the custom designed soft processor, we were able to use multiple separate simplified peripherals. The basic PWM modules, for example, need only three registers (divider, comparator, and enable/control).

Similarly, we could implement as many separate IO ports as desired for the separate banks of switches, lights, and GPIO pins. We could avoid immediately having to get into the details of bit shifting and manipulation early on, since each IO port had only a single specific function. These topics are of course covered later.

Thus in general we were able to spend more time on core concepts that the students needed to learn, and less time learning specific complex architectural details of a particular system, that has only a small chance of being the same family a student is likely to end up working on in their career (and only if they even go into a field involving embedded development.)

Outcomes

Generally the results for the first class through the course were extremely positive. With few exceptions, the soft processor system enabled labs that covered all desired topics. In particular the simple nature of the peripheral systems allowed the students to focus on specific topics on each lab. This led to very good results with the assembly programming early in the course, as they could focus more clearly on the topic at hand, and the assembly code did not become bloated quickly due to excessive setup and initialization code.

Once they transitioned to C, those students who had experience with any kind of C-syntax language like Java were able to really take off and run with most of the labs, often expanding them. These students usually preferred the C programming in the second half of the course. Additionally, because the Newlib standard library is so extensive, combined with the simple to use debugging window, once the students began the C programming section, it is possible for them to go through and run almost any example programs or tutorials they might find online.

However, those students who had not had any programming experience yet did struggle somewhat with the transition to C, particularly due to the vastly different syntax. They generally found the more explicit programming style of the assembly to be favorable. Since the Eclipse debugger has a dis-assembly view and single-instruction-stepping mode, the students were able to compare the assembly code they had produced in previous labs with the assembly generated by the C compiler. As a final project the students wrote a program that played music, and some chose assembly over C, and thus were able to compare the methods they used and the code generated.

In terms of the hardware, the students found it to be fairly user friendly. The first group of students had used some of the previous hardware platform(s) at the school and so were able to contrast them. The USB connections (the board can be USB powered) made it easy for them to setup the system on their own laptops and develop at home, and more than one expressed interest in purchasing a board for their own ongoing use as the board is only around $150 from distributors like Digi-Key.

There were some problems in the first course through with interrupts functioning properly in all circumstances, but these have since been resolved. There is also a limitation in that permanent programming of the boards (so the program is stored and runs on start up) requires a few extra steps. The hex code generated by the compiler must be combined with the FPGA bit stream into a JTAG Indirect Configuration file (.jic file) that can then be programmed into the boards. This had little impact on the normal lab experience, but would be an issue if the platform was used outside the microprocessor course as a general-purpose microcontroller for a project course or other labs.

Naturally, the boards completely met our desire for a platform that could also be used for a digital logic course. And although outside the normal coursework at a technical college level, those students who are interested can explore the system-on-chip design process and supporting verilog code if so motivated.

Future work

The most immediate goal is to finish the add-on hardware interface board to have it available for the second class to go through the course. A small batch of the first revision boards will need to be manufactured and assembled.

We also have given some consideration to reshaping the course structure to use C in the introductory labs and then transition into assembly language in the second half of the semester. One of the biggest challenges was for students who had little to no programming, after getting used to the extremely explicit nature of assembly, trying to transition to a language that has more abstract concepts of data and control flow can be challenging. Some other programming courses have had success starting with C and then introducing assembly. On the other hand, some concepts like pointers can be much easier to understand when one has knowledge of the processor's addressing modes and how registers can hold and manipulate addresses as any other data.

This is one area that's definitely worth more consideration. Alternatively, we have also given consideration as to whether splitting the course into separate introductory C programming and assembly/architecture courses would be appropriate.
Lastly, the platform as a whole can be expanded from even a fairly simple processor as used in this course to an extremely advanced microprocessor system. Because of this there may be good opportunities for a similar system at the university level as part of an elective in embedded systems programming, or for architecture courses.

Some possible areas for expansion in such a course might include: Real Time operating systems, multiprocessor systems, use of more advanced peripherals such as Ethernet and audio codecs, and Linux kernel and driver development. (The processor can be built with a memory management unit.)

References

Quartus software and NIOS IDE
https://www.altera.com/download/software/quartus-ii-we

NIOS II Developer's Guide
http://www.altera.com/literature/lit-nio2.jsp?ln=devices_processor&l3=Processors-Nios%20II&l4=Literature

Terasic FPGA development boards
http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=13&List=Simple

RTOS
http://www.altera.com/support/examples/nios2/exm-microc-osii-tutorial.html

Processor platforms:
Atmel AVR (http://atmel.com/products/microcontrollers/avr/default.aspx)
Microchip PIC (http://www.microchip.com/pagehandler/en-us/family/8bit/)
TI MSP (http://www.ti.com/msp430/)
ST STM32 (http://www.st.com/web/en/catalog/mmc/FM141/SC1169)
Xilinx Blaze (http://www.xilinx.com/tools/microblaze.htm)
Altera NIOS II (http://www.altera.com/devices/processor/nios2/ni2-index.html)