

Integrating Hardware and Software Concepts in a Microprocessor-Based System Design Lab

Steven K. Reinhardt
EECS Department
The University of Michigan
1301 Beal Ave.
Ann Arbor, MI 48109-2122
stever@eecs.umich.edu

Abstract

The EECS 373 “Design of Microprocessor-based Systems” course at the University of Michigan ties hardware and software together by providing a modern platform on which students simultaneously develop both hardware and software components of simple systems. Our semi-custom target platform combines a highly integrated 32-bit embedded Motorola PowerPC processor and two high-density Xilinx FPGAs, allowing plenty of headroom for follow-on student projects and future course expansion. Versions of the development tools we employ in the lab are also available, with simulation capabilities, on public workstations and students’ personal PCs, enabling students to work outside the lab and maximizing the leverage of finite lab space in the face of growing enrollments.

1. Introduction

Many (if not most) undergraduate computer science/engineering programs fork very early into “hardware” and “software” tracks. As a result, opportunities to combine hardware and software concepts in significant ways are limited. However, as microprocessor-based systems become ever more widespread, the ability to view a hardware/software system as an integrated whole is a crucial part of an engineering education. At the University of Michigan, we have developed a lab course on microprocessor-based system design (EECS 373) that addresses this need. The key feature of EECS 373 is the integration of a modern, 32-bit microprocessor platform with FPGA devices, allowing simultaneous design of hardware and software components in a real-world environment.

The official ABET objectives for the course are that students will:

- learn how the hardware and software components of a microprocessor-based system work together to implement system-level features;
- learn both hardware and software aspects of integrating digital devices (such as memory and I/O interfaces) into microprocessor-based systems;
- learn the operating principles of, and gain hands-on experience with, common microprocessor peripherals such as UARTs, timers, and analog-to-digital and digital-to-analog converters;
- gain practical experience in applied digital logic design and assembly-language programming; and
- be exposed to the tools and techniques used by practicing engineers to design, implement, and debug microprocessor-based systems.

This paper outlines the history and motivation behind the current course, describes the lab infrastructure and current lab sequence, and concludes with a look at possible future directions.

2. History and Goals

EECS 373 at Michigan was originated in 1990 by Prof. Janice Jenkins. In its original incarnation, students constructed simple systems on a solderless breadboard combining an Intel 80186, SRAM, EPROM, an 8251 UART, an 8255 parallel I/O chip, and an analog-to-digital converter. Although the functions of the system were very limited, students generally enjoyed the course and derived much satisfaction and experience from putting the system together and making it work. However, the lab component of the course had very little software design content and no hardware design content. A wiring diagram was provided for each lab; students simply wired up their breadboards as indicated

and wrote a small program to exercise the relevant hardware component(s).

By the fall of 1997, it was clear that a wholesale update of the lab was necessary. The Intel in-circuit emulators for the 80186 on which the students relied were no longer being manufactured, and as these emulators broke, we were forced to reduce the number of available lab stations. This situation was particularly painful in the face of steadily increasing enrollments. We also had a desire to increase the design content of the course, but the infrastructure was not capable of supporting this change. The noise and reliability problems inherent to solderless breadboards caused great frustration in merely interfacing RAM and ROM dependably enough to boot the processor. Developing and debugging more significant logic circuits was out of the question.

Our priorities for the new lab infrastructure were to support:

1. the ability to teach concepts that relate to both embedded and general-purpose systems
2. labs that focused on system design (hardware and software) rather than debugging of physical connections
3. a “real world” (relevant and non-artificial) environment that would engage students without being overwhelming
4. a development environment that could be extended outside the lab, so that a limited number of lab stations could be leveraged to support a large number of students

To satisfy goals 1 and 3, we wanted a 32-bit processor that had a reasonable market presence in both the embedded and general-purpose markets. This consideration eliminated any of the 8- or 16-bit microcontrollers (such as the Motorola 68HC11 or Intel 8051) commonly used in undergraduate labs. As EECS 373 is typically the first course in which students do a significant amount of assembly-language programming, we also wanted a processor with a fairly regular instruction set so that learning the ISA would not be an obstacle. We had found in the previous semesters that the irregularities of the Intel x86 architecture often created difficulty for students with no assembly language background, or whose only exposure to assembly was the use of DLX in the undergraduate computer organization course.

We chose the PowerPC architecture because it has a presence in the general-purpose market in the Apple Macintosh product line, as well as a significant and

growing share of the embedded market. In particular, the PowerPC architecture has also been adopted by Ford, a major local employer, for many of its automotive embedded applications. As a result, embedded-system development tools for PowerPC devices are widely available.

To satisfy goal 3, we required development and debugging technology that allowed students to focus on design correctness rather than physical implementation idiosyncrasies. FPGAs were the obvious choice to replace breadboards for hardware implementation. For software debugging, we sought a processor that supported on-chip emulation (aka background debug mode, or BDM), where the CPU itself has the ability to act as an “in-circuit emulator” under the control of a debugger running on another machine. This technology provides the power of in-circuit emulation without the expense or physical interconnection issues.

To satisfy goal 4 (enabling students to work outside the lab), we had originally hoped to develop a target platform that would be inexpensive enough to provide to each lab group. However, we found that the development tools we selected (Xilinx Foundation and SDS SingleStep) both provided simulation environments, allowing students to develop and even test both hardware and software components outside of the lab. Both of these tools run under Microsoft Windows, so they were easily deployed on the numerous public Windows NT workstations supported by the College of Engineering. Students who wish to work on their personal PCs can take advantage of a free demo version of SDS SingleStep (which is functional enough to support nearly all of the course needs) and an inexpensive student version of Xilinx Foundation. By making these development and simulation tools widely available, we felt comfortable supporting a large class (up to 100 students per term) on a limited number of physical lab stations, allowing us to focus more resources on each individual hardware platform.

3. Lab infrastructure

At the time of lab development, we were unable to find a commercially available platform that combined a 32-bit processor with FPGAs on a single board. The only readily available platforms that integrated FPGAs and processors were focused on FPGA development, using only a simple microcontroller (e.g. an Intel 8051) for the CPU. We thus reluctantly concluded that some custom development would be necessary. To minimize

the amount of effort, we added a custom FPGA-based expansion board to an existing processor development kit.

Our specific target platform is the Motorola MPC823ADS application development system, which uses the Motorola MPC823 embedded PowerPC processor. The MPC823 is a highly integrated device, including the usual embedded controller peripherals (timers, interrupt controller, memory controller, etc.) along with a communications coprocessor, video controller, Ethernet, USB, and PCMCIA interfaces. It is used in the Kodak DC260 and DC280 digital cameras. The MPC823ADS system consists of two boards, a motherboard that supports the entire MPC8xx family and a daughterboard with the MPC823 itself. The motherboard provides 4 MB of DRAM and 2 MB of flash memory, plus buffering logic and physical connectors for most of the MPC823's I/O interfaces. The daughtercard includes a video encoder chip, video connectors, USB connectors, and high-density Mictor connectors for HP logic analyzer probes, in addition to the processor itself. Our custom expansion board sandwiches between the motherboard and daughterboard, with connectors on both sides. Figure 1 presents a block diagram of the target platform, and Figure 2 is a photograph of an assembled system.

The expansion board holds the FPGAs and several simple I/O devices used in the basic lab sequence, including a two-digit seven-segment LED display, a 10-element LED bar graph display, DIP and pushbutton switches, A/D and D/A converters, and a 64Kx8 SRAM chip. One of the FPGAs interfaces the processor bus to the simple peripherals, while the other connects the bus to a 40-pin header for further expansion.

Because we wanted to connect the FPGAs to the full processor bus as well as peripheral devices, we were primarily constrained by pinout rather than density. We selected Xilinx XC4010E FPGAs because they were available in 191-pin ceramic PGA packages. Although higher-pinout devices were available in surface-mount packages, we wanted to stay with socketable devices with the expectation that student use would require occasional replacement of burned-out chips.

Students use the Xilinx Foundation tools to develop their hardware designs. Currently we use schematic capture for design entry, though HDL synthesis is also available. The Xilinx tools also include a logic simulator that allows students to debug their hardware designs outside of the lab. For in-lab debugging, we

have HP/Agilent 16600A 136-channel logic analyzers that connect to the processor bus for a full display of processor activity. Additional analyzer channels can connect to test pins on the expansion boards; students can route internal FPGA signals to the test pins to aid in debugging their designs.

Software development is done using SDS SingleStep, an application that runs on a Microsoft Windows PC and provides an integrated graphical debugger, editor, C compiler, assembler, and linker. SingleStep communicates with the MPC823 processor via its BDM (debugging) port to provide full visibility of CPU and memory state as well as single-step and breakpoint control over execution.

In addition to the target platform and the HP logic analyzer, each lab station has a 333 MHz Intel Pentium II PC running Windows NT to run the Xilinx and SDS tools and a basic oscilloscope and analog function generator for use with the A/D and D/A converters.

4. Lab sequence

The current lab sequence starts with two introductory labs, one hardware and one software. In addition to familiarizing students with the development and debugging tools, these labs provide time to present necessary background material in lecture. The real design labs start with the third lab exercise. These labs build on each other to the point where students have designed and constructed the software and glue logic for a system with polled serial I/O and multiple interrupt sources. The sequence is capped with a simple mini-project that allows the students to explore or expand on facets of the system that are not covered in the standard labs. As time progresses, we plan to incorporate the more successful and exciting mini-projects from previous semesters into updated standard labs in later semesters.

Lab 1: Introduction to Xilinx FPGAs and the EECS 373 Expansion Board

This lab serves to introduce the students to the digital hardware side of the lab equipment, including the Xilinx design tools. Students implement a simple circuit that takes a binary value input via DIP switches and displays it in hexadecimal on a seven-segment display. The processor is not used in this lab. The lab is intentionally simple so that students can complete it in their first lab session without additional preparation.

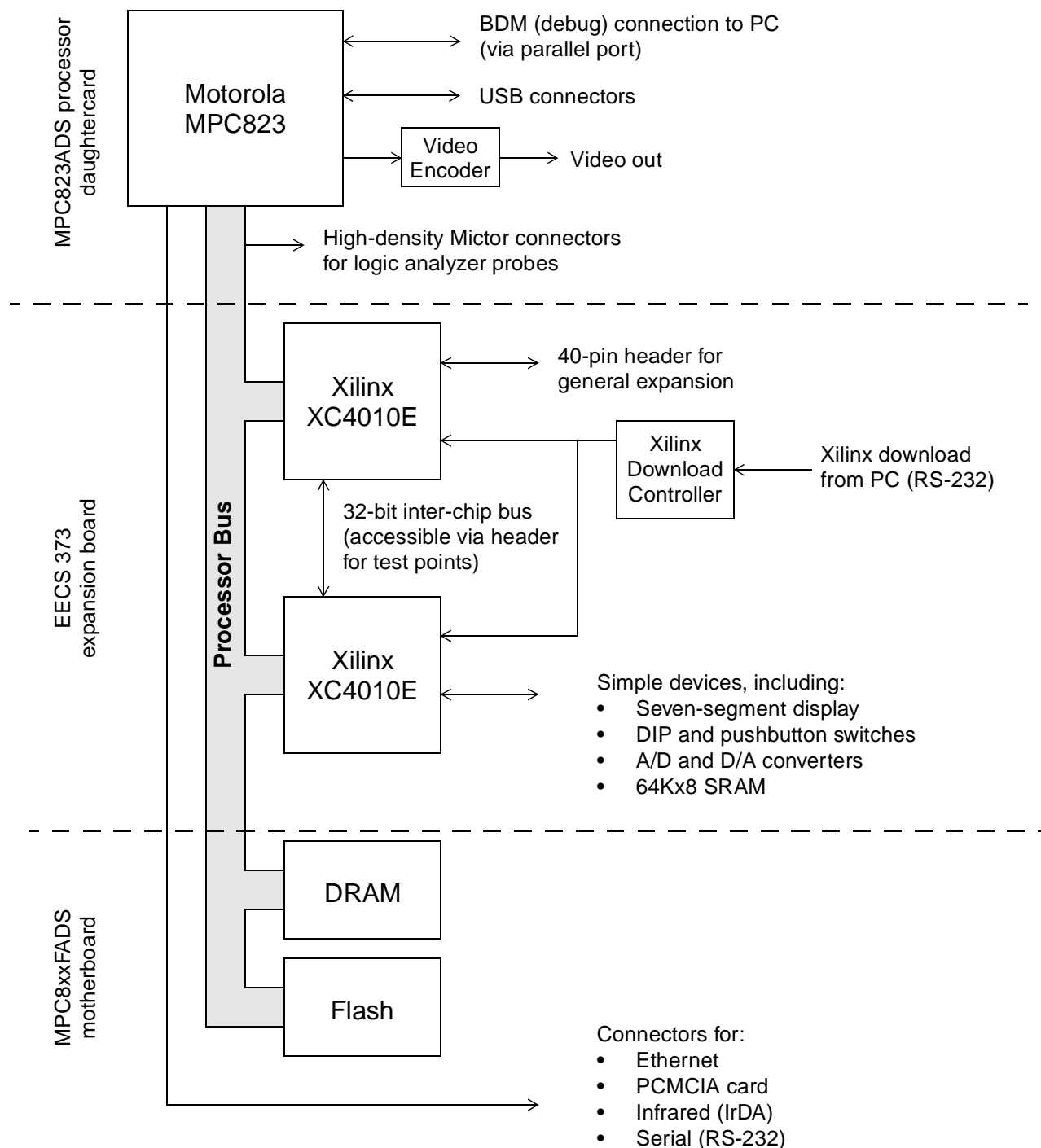


Figure 1. Block diagram of target platform.

Lab 2: Introduction to PowerPC Assembly Language and the SingleStep Simulator

The second lab introduces the students to the software tool chain. Before coming to lab, the students are

exposed to the assembler, linker, and debugger by observing and modifying a simple program using the SDS CPU simulation environment outside the lab. In the lab session itself, we present the students with another program that contains both syntactic and logi-

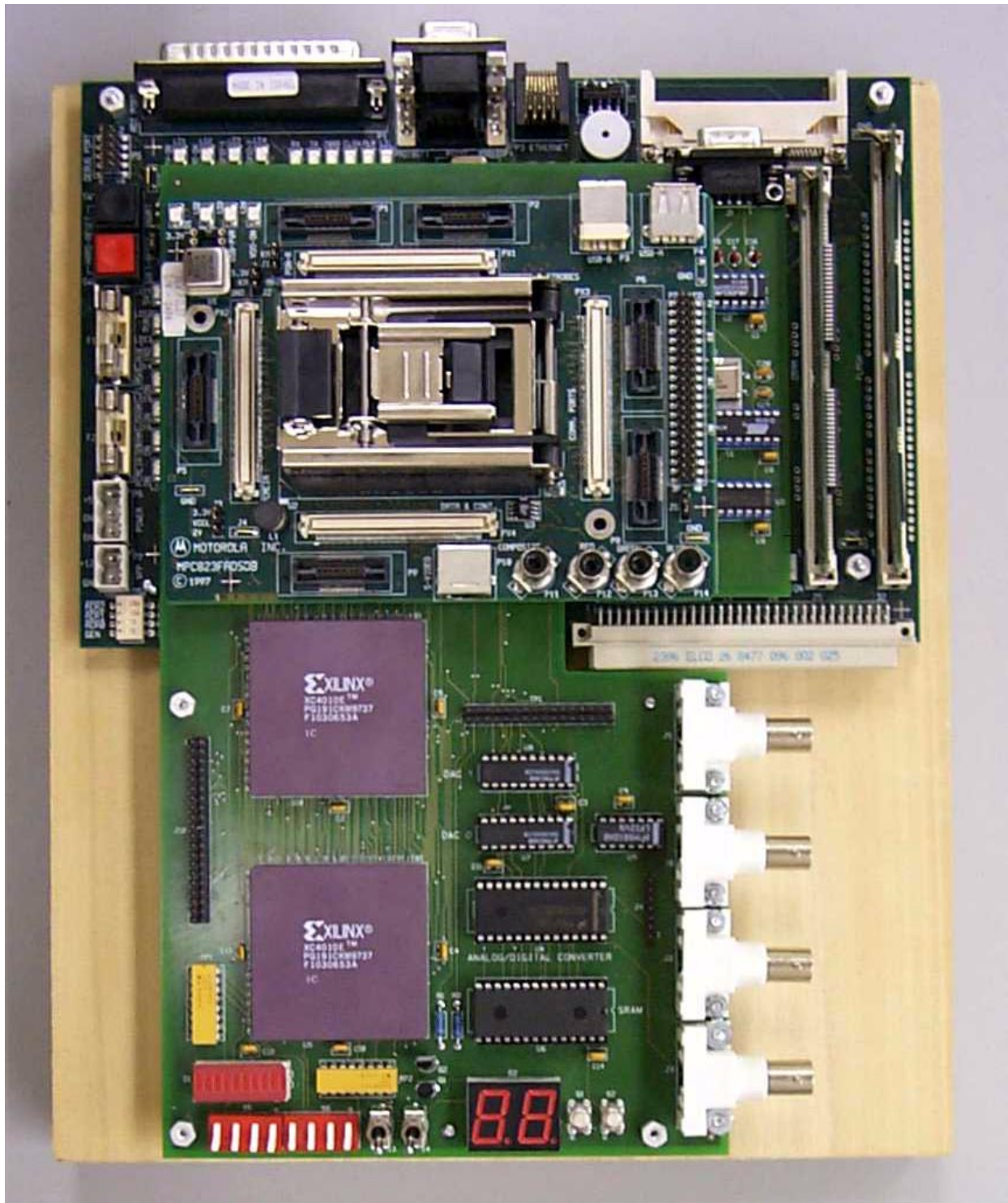


Figure 2. Photograph of target platform. The processor daughtercard is at the top (with the MPC823 in the large ZIF socket). The motherboard is underneath at the top. The expansion board is sandwiched in the middle, but extends below the daughtercard to expose the FPGAs and I/O devices.

cal errors. The students must use the tools to debug this program, solidifying their familiarity.

Lab 3: Basic Bus Interfacing for I/O Devices

The first “real” lab involves designing the glue logic to interface simple input and output devices (switches and LEDs) on the expansion board to the MPC823 bus. The software component involves polling the switches and writing to the LED registers to implement a simple specification (e.g., one pushbutton initializes the display based on DIP switch positions, while another pushbutton complements the displayed value).

Lab 4: Bus Interfacing for Byte-Addressable Memory

This lab reinforces the concepts of transfer size and alignment by having the students build a 256-byte (32-word), 32-bit wide, byte-addressable memory on the Xilinx device. This memory is constructed from 32x8 RAM modules available in the standard Xilinx library. The students’ designs must handle byte, halfword, and word reads and writes. The simple I/O from lab 3 is carried over as well, and the software component is enhanced to allow the use of the DIP switches, pushbuttons, and LED displays to read and write locations in the memory (a la the front-panel switches of yesteryear).

Lab 5: Simple Serial Communication

In this lab, the students develop a simple calculator program that communicates via an RS-232 serial link. Unlike labs 3 and 4, which emphasized hardware, lab 5 is purely a software lab. However, it serves several useful purposes:

- The program is complex enough that students are forced to decompose it into functions. This reinforces assembly-language procedure calling, a prerequisite for managing interrupts in the succeeding lab.
- We introduce the concept of a *device driver* as a module that hides implementation-specific device details behind a standard procedural interface. This concept came of necessity, since the simulated UART model available with the processor simulator is completely different from the actual serial interface available on the MPC823. We turned necessity into a virtue by defining a serial device-driver interface and

having the students implement the driver for the simulated UART. We then provide the (significantly more complex) driver for the MPC823 serial port.

- The terminal-handling program provides the basis for richer interaction with the target platform than switches and LEDs allow. We take advantage of this in lab 7, where the program is enhanced with commands to control an interrupt-driven LED display.

Lab 6: Basic Interrupts

In lab 6, students use interrupts to add a simple timer function to their lab 5 system. The seven-segment display indicates elapsed seconds (using periodic interrupts from the MPC823’s real-time clock), while the pushbuttons generate interrupts to start/stop and clear the timer. In addition to writing the appropriate interrupt service routines, the students must design the hardware to generate a shared level-sensitive interrupt from the pushbuttons, including the capability to poll and reset the interrupt status via software. Meanwhile, the polling-based serial calculator from lab 5 must continue to work unmodified. To catch stack overruns more quickly, the students use their 256-byte memory from lab 4 for the stack.

Lab 7: Timers

Lab 7 builds on the lab 6 system, using timer interrupts to generate a “chaser” display on the expansion board’s bar-graph LEDs. Unlike lab 6’s timer interrupt, which is a simple once-per-second interrupt provided by the MPC823, lab 7 uses the MPC823’s programmable interval timers. The “calculator” program is expanded to support two more terminal commands that modify the frequency and direction of the chaser display, respectively. This lab introduces students to programmable timers, including the derivation of timer configuration settings to provide interrupts at various frequencies, and also provides additional experience with interrupts.

Lab 8: Analog-to-Digital Conversion

The final structured lab exercise has the students interface the simple analog-to-digital converter found on the expansion board (a National Semiconductor ADC0808) to the CPU. The students must decipher the appropriate timing constraints from the ADC data sheet, and convert these into a hardware bus interface.

This task is not entirely trivial as the CPU bus runs at 20 MHz and many of the minimum input pulse widths of the ADC are measured in microseconds. The software component of this lab uses the pushbuttons to trigger data collection at multiple sample rates. The students set up the programmable timers from lab 7 to initiate individual sample conversions at the desired rate, while the end-of-conversion signal from the ADC triggers a separate interrupt to read and store the sample. Students save their digitized samples of various-frequency sine waves and do some simple post-lab analysis of the recorded signal using a spreadsheet or Matlab.

Lab 9: Mini-Project

The final exercise of the semester allows the students to explore the capabilities of the lab system by defining their own (mini-)project. In recent semesters, students have experimented with the video controller, infrared port, and ethernet port, and have interfaced various other sensors and output devices as well. Our goal is to let the students build on projects and knowledge gained from previous semesters, leading to a library of interesting software and hardware components which students can integrate to build interesting systems.

5. Conclusion and future work

By combining a highly integrated 32-bit embedded processor with high-density FPGAs, we have developed a flexible, powerful lab platform with plenty of headroom for follow-on student projects and future course expansion. FPGAs allow students to design interface circuits for a 32-bit bus without the wiring complexity, noise, or reliability issues of the breadboards they replace. Forcing the students to design interacting hardware and software components gives a tangible appreciation for the way in which these components cooperate. Students also appreciate the opportunity to work with state-of-the-art components that have real-world significance and compare with what many of them will find in industry; the challenge is for the instructor to avoid overwhelming the student with the complexity that is inherent in these modern devices. We leverage our investment in a limited number of physical lab stations by making versions of the development tools with simulation capabilities widely available outside the lab.

Although the basic lab infrastructure is stable, we are continually seeking to improve and enhance the curriculum to take fuller advantage of this equipment. The greatest challenge to improving this course lies in avoiding the temptation to throw more material at the students than most can absorb in a single semester. Possibilities that we have considered include:

- expanding the range of hardware devices that can be interfaced
- interfacing assembly with C or C++ to allow more significant software efforts
- letting students burn their code into the development system's on-board flash memory to create a stand-alone bootable system
- providing TCP/IP support over the ethernet port (eventually building up to an embedded web server)

Fitting more topics into the current course would require raising the level of detail (e.g., by providing Xilinx macros for selected bus interface circuits) and/or replacing some existing labs (e.g., the timer or ADC labs). We can also provide further opportunities to interested students by offering additional courses using the same or similar lab equipment. For example, Prof. Jim Freudenberg is developing an embedded control systems course with EECS 373 as the only prerequisite. Motivated students are also encouraged to do independent-study design projects for course credit.

For more information, including detailed lab descriptions and expansion board schematics, see <http://www.eecs.umich.edu/courses/eecs373>.

Acknowledgments

Steve Raasch designed and managed the construction of the EECS 373 expansion boards. Steve Raasch, Nathan Binkert, Erik Hallnor, and Chau Doan also played crucial roles as teaching assistants for the first offering of the revised course. Ghassan Shahine and Trevor Mudge taught the course subsequent to the revision and contributed many improvements.

The new lab was made possible by generous support from Software Development Systems, Inc. (SDS, now a part of Wind River Systems), Hewlett-Packard/Agilent Technologies, Intel, and Motorola. We particularly acknowledge the support of Manny Hatz, Mike O'Donnell, and Bill Wasserman.