Developing Multidisciplinary Laboratories with the Cypress PSoC

Project Proposal

Stephen Hammack
sch7373@cs.rit.edu

Chairperson: Jim Vallino

Rochester Institute of Technology
Department of Computer Science
102 Lomb Memorial Drive
Rochester, NY 14623-5608

Table of Contents

1. Problem

There aren't many courses at RIT that combine both Computer Engineering and Computer Science. Because of that, it is hard to be multidisciplinary in computer hardware and software if you are interested in both. The goal of this project is to create educational labs using a hardware platform that combines software and hardware for multidisciplinary learning. These labs will combine programming software and configuring hardware to develop example real-world applications, leading to a culminating project.

2. Definitions

Here are the definitions of terms used in this proposal that may be unfamiliar.

- ADC – Analog to Digital converter: reads an analog input voltage and converts it to a digital value
- ARM – a CPU architecture widely used in embedded hardware, such as microcontrollers and cell phones
- ASIC – Application-Specific Integrated Circuit: an integrated circuit designed for a specific purpose, rather than general use, such as a digital camera image processor. The digital logic is hard-wired in an ASIC, unlike FPGAs.
- Auxiliary hardware – any device that is not part of the PSoC used for input and output (e.g. LEDs, servos, pushbuttons, etc.)
- Clock signal – an electrical signal that oscillates between a high and low state at a specified rate (Hz). It is used to synchronize hardware components
- DAC – Digital to analog converter: takes a digital value and converts it to a corresponding analog output voltage
- HDL – Hardware Description Language: a language used for formally describing an electronic circuit, such as a microprocessor or the configuration of an FPGA. Verilog is a commonly used modern HDL, which uses a syntax inspired by C.
- FPGA – Field Programmable Gate Array: a commonly used type of programmable computer hardware that contains many logic gates, along with other components such as RAM and analog hardware. An HDL is used to configure the connections between the gates for the desired application, which allows for flexible digital hardware design without needing to fabricate an ASIC.
- IC – Integrated Circuit: a set of circuits embedded into a single chip, rather than made of multiple discrete components
- Pin – an input or output connection from the PSoC, used to interface with auxiliary hardware
- Potentiometer – a resistor that can be adjusted to vary its resistance
- PSoC – Programmable System-on-Chip: a SoC by Cypress Semiconductor that combines a CPU and programmable digital and analog components. It essentially combines the flexibility of a CPU for complex software and an FPGA for flexible hardware interfaces. The PSoC manufacturer provides design tools for configuring the hardware at a high level including software interfaces to the hardware, which makes it easy to use.

- PWM – Pulse-width Modulation: an electrical signal that utilizes square waves as pulses to control power or transmit information. Typically, the square wave signal has a fixed period and varying duty cycle, the ratio of the square wave's pulse duration to its period. Varying the duty cycle provides the control or information transfer. One application is to quickly flicker an LED such that it appears dimmer than it would with constant power. Another application is for a motor controller to measure the duty cycle and translate it to the speed and direction of the motor connected to it.
- SoC – System-on-Chip: an IC that combines all computer components onto a single chip. These often also include digital and analog I/O capabilities

3. Background

The goal of this project is to create a set of laboratory exercises to teach Computer Science students about computer hardware and how to interface with it from software. Learning about computer hardware and interacting with it from software would be a focus of a course using any of the labs that this project will develop. The labs need to be approachable to Computer Science students. Allowing software to interface with the hardware, rather than having everything done in hardware, provides a familiar ground for Computer Science. A Computer Science student will also likely find algorithms easier to write in software than hardware.

There are many published papers with Computer Science courses that utilize computer hardware. One example paper focuses on hardware design with FPGAs [1]. Because of the high configurability of FPGAs, they are good tools for teaching a wide range of computer hardware design skills. Within limitations set by the FPGA being used, it can be used to interface with and process almost anything. This feature is the focus of another paper that utilizes FPGAs to teach concepts of reconfigurable computing [2]. However, FPGAs are configured with an HDL, which requires knowledge of computer hardware design. Computer Science students generally do not have the skill set necessary for using HDL [3], so working with low level hardware on an FPGA would be difficult. This project is more focused on learning high level computer hardware concepts, while HDL is low level, so HDL will not be used. Fortunately, there are various ways to abstract the configuration of an FPGA, which will be taken advantage of in these labs.

There are many real-world applications where software running on a CPU interacts with hardware. Systems that are classified as real-time or embedded systems almost always have software control of external hardware elements connected to the physical world. An example is software called firmware that makes calculations and runs the hardware components inside a larger hardware device, such as a wireless card. Another example is a control system for a robot, which interacts on larger scale hardware such as sensors, joysticks, and motors. Because of these applications, these labs will be designed to use software that interacts with hardware. While firmware generally is not taught in Computer Science courses due to how closely it is tied to its hardware, if an FPGA and CPU are used in combination for these labs, teaching firmware concepts becomes easier. There are courses that already use this combination for teaching hardware/software design, though they seem to be more Computer Engineering oriented [4]. The configurability of a PSoC allows early labs to limit how complex the hardware is while students are learning firmware concepts, and increase in complexity as the labs progress. Some labs will also use hardware, such as a servo, that may be used in a larger system like a robot.

The fact that robots in Computer Science programs can motivate students to learn [5] will be taken advantage of when it can.

In summary, the purpose of these labs is to teach high level computer hardware design and interactions between it and software. The hardware aspects need to be approachable to Computer Science students while still teaching real world concepts. This leads to the chosen development platform for the course, the PSoC by Cypress, which abstracts the hardware design and communication between the hardware and software.

4. Functional Specification of the PSoC

The Programmable System-on-Chip, or PSoC, by Cypress [6] is a device that combines an embedded ARM CPU with programmable input/output hardware, making it adaptable to many physical applications. This flexibility makes it ideal for implementing labs with a broad variety of topics and real-world applications, because there are many different types of I/O available to the PSoC and they can be configured as needed. Additionally, just like an FPGA, things like digital logic in hardware are possible, allowing such concepts to be taught. Unlike an FPGA, since hardware is configured using components in a diagram, low level computer hardware design and HDL are not required. Additionally, hardware components come with software APIs, so the communication between the hardware and software is abstracted.

In particular, the CY8CKIT-050 PSoC 5LP Development Kit (Figure 1) will be used because the PSoC 5LP has the most capabilities and the development kit has some auxiliary hardware on-board and easily accessible pins for additional auxiliary hardware. Refer to the lower half of Figure 1 to see what is available. Note that the prototyping area allows custom I/O hardware to be installed for use. The expansion ports D and E allow for I/O hardware by Cypress, such as more CapSense (capacitive touch) sliders and buttons, and may also be used for additional custom I/O.

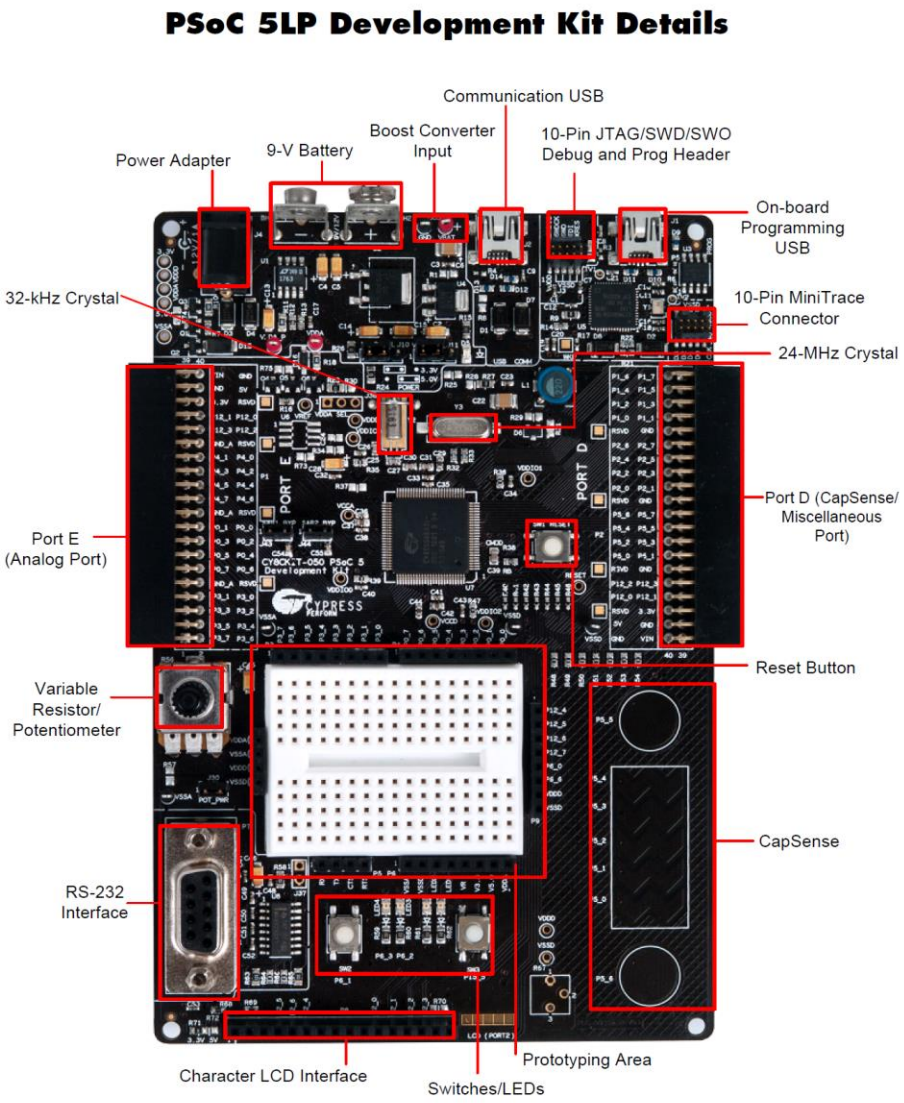

Figure 1: Diagram of the PSoC 5LP Development Kit [7]

5. Overview of the Labs

Because the PSoC has two programmable parts, hardware and software, the labs will be described in these two portions. Hardware design will be discussed first in the lab assignment because the software interfaces with it.

Each lab should take about two hours to complete, to accommodate for any issues a student may face when understanding the lab. Each lab will follow the design specification described in the next major section. Additionally, there will be guidelines for the final project and at least one example final project.

It is suggested that these labs be done in a physical lab environment with professors or TAs who have read the professor materials for the lab. Holding a physical lab will allow students to ask questions, since the target audience for a course that uses these labs includes students who don't know about computer hardware. Additionally, demonstrating that the lab works would be easier to do directly to the professor.

This is the suggested schedule for the labs over a fifteen week semester:

- Week 1 – No labs, time used for syllabus and starting material
- Week 2 – Lab 0 to introduce developing with the PSoC
- Weeks 3-12 – One lab per week, total of ten labs
- Weeks 13-15 – Final project

The following sections describe lab material to be created and outline the three initial lab exercises.

5.1. Setup and General Usage Notes

Before using the PSoC development kit, there is some set-up required, such as installing the development software. There are also useful things to know about the development kit, such as the built-in auxiliary hardware on the board. Additionally, there are some important guidelines to follow to avoid damaging the development kit. For example, some pins are better used for analog signals, while others are better for digital. Setup and general usage notes that cover these issues will be created. Some of the more important warnings will also be highlighted where appropriate in the labs as reminders.

5.2. Lab 0: Introduction to PSoC and PSoC Creator

This lab will introduce the basics of using the PSoC and the IDE: PSoC Creator. The way the IDE is used to program the PSoC is different than normal for several reasons. A schematic design view is used to arrange and wire together hardware components. A graphical configuration interface is used to assign pins, interrupts, and other things. Based on the schematic design, C code is generated to interface the CPU with the hardware. Then the student writes software to perform any necessary software tasks. The code is then compiled and loaded onto the PSoC, along with the hardware configuration.

The student will implement the following using the IDE, following instructions from the lab manual.

5.2.1. Hardware

To keep this lab simple, the only hardware will be an output pin for an LED, an input pin from a push button, and an LCD panel. The output pin will be one of the ones connected to an LED on the development board. The input pin will be one of the ones connected to a push button on the development board. The LCD panel will be plugged in to the port at the bottom of the development kit.

5.2.2. Software

To keep this lab simple, the software will continuously read the input pin and turn the LED on if the button is depressed or off if the button isn't depressed. The student code will all run in the main function, and use functions that the development environment generates to interface to the pushbutton and LED. Additionally, the software will count how many times the button is pressed and print the count to the LCD.

5.3. Lab 1: Basic Analog and Digital I/O

The purpose of this lab is to show how to input and output analog and digital signals, and do processing without CPU intervention. In this lab, the pushbutton will light an LED entirely in hardware. Lab 0 did the same thing with CPU intervention, however, using the CPU to continuously poll the pushbutton is inefficient, complicates the ability to perform other tasks at the same time, and does not guarantee how quickly the CPU can respond. Using only hardware to handle the same task fixes these problems.

The second part of this lab deals with reading an analog value. The student will use an ADC to measure the voltage on a potentiometer, and print the value to the LCD display using the CPU.

The student will implement the following using the IDE, following instructions from the lab manual.

5.3.1. Hardware

An ADC will be configured to read the voltage on a potentiometer. The LCD display will be used to display the value read by the ADC. Additionally, the PSoC will be configured to have a pushbutton light up an LED while the button is depressed. The development board includes all the auxiliary hardware without additional wiring: the LEDs, the pushbuttons, the potentiometer, and the LCD display (plugs in to a dedicated port).

5.3.2. Software

The software will continuously perform the following operations: wait for a new ADC reading, use the value returned from the ADC and previously returned values to calculate a moving average, which is necessary to reduce noise, and print the calculated value to the LCD display.

The LED control is completely implemented in hardware, so the software is not involved.

5.4. Lab 2: Counters, Flip-Flops, Debouncers, and ISRs

The purpose of this lab is to introduce interrupts and three hardware components: counters, flip-flops, and debouncers. Interrupts allow the CPU to respond to a hardware event without continuously checking for the event. Counters have a value that increments on a stimulus and can produce a signal depending on the value. Flip-flops store a value that can be used later. In particular, this lab uses a toggle flip-flop. Debouncers filter inputs, such as pushbuttons.

The lab also optionally introduces sleep modes. Putting the PSoC into a sleep mode while the CPU is not doing anything reduces power usage, which is very important in many applications.

The product of this lab is a numeric value on the LCD screen that increments every second, a value on the screen that increments every time a pushbutton is depressed, and an LED that toggles between on and off when the same pushbutton is depressed.

The student will implement the following using the IDE, following instructions from the lab manual.

5.4.1. Hardware

Synchronous circuitry plays a part in this lab. Essentially, a synchronous circuit waits for a clock signal before reading its input and changing its internal states and output. This property allows hardware components to work together, as described below. Synchronous circuitry will be discussed in more detail in a future lab.

A counter is a component that holds a value that counts up or down on a stimulus. The counter can also generate a signal when the value meets a certain condition. In this case, the counter is configured to count up at every clock pulse, where the clock is running at 1 kHz. Additionally, it is configured to signal an interrupt when the count reaches 1000, or after one second passes. It is also configured to reset itself when this happens so it will continually generate an interrupt every second.

A debouncer is a component that filters an input, to ensure that a state change is stable before reflecting it in the output. The input is sampled when a clock signal goes high, where the clock should be between 10 and 200 Hz. In other words, switches like pushbuttons often provide an unstable signal while it is transitioning between pressed and depressed states, and the debouncer does not change its output until the input is stable. This filter is necessary to ensure that a button press is only registered once. Additionally, the debouncer can send a pulse that lasts for one clock period when the state goes positive, goes negative, or both.

A flip-flop is a component designed to hold a bit for future use. In this case, a toggle flip-flop is used with the debounced pushbutton as the toggle input. The toggle flip-flop will switch its state from 0 to 1 and vice versa when the toggle input is high and the clock goes high.

For this lab, a pushbutton is connected to a debouncer, which is configured to generate a pulse when the button is depressed, or goes into a negative state. The pulse is connected to an interrupt and a toggle flip-flop. The flip-flop uses the same clock as the debouncer so the two are synchronized. The output of the flip-flop is connected to an LED output pin. This combination of components allows the pushbutton to toggle the LED on and off.

5.4.2. Software

Interrupt service routines (ISRs) are used so a CPU can quickly respond to hardware events without needing to spend time polling the hardware to see if something happened. Utilizing these frees up CPU time to do other jobs or enter a low power state, increasing power efficiency while remaining responsive. It also allows the CPU to more easily respond to multiple event sources.

In this lab, ISRs are used to update the LCD display with the count of the counter and pushbutton events. Real applications could use the same interrupts to run a periodic task or efficiently respond to user input.

Optional: since the only thing the CPU is doing is responding to interrupts, the main function does nothing after initializing the system other than looping infinitely, since exiting the main function shuts down the system. A sleep mode can be used to save energy when some functionality is not needed, which is important in situations where there is a limited power supply. This optional part of the lab will use a sleep mode that turns off the CPU and leaves the hardware running with the interrupts active.

5.5. Labs 3-10

Here is a list of potential labs for Labs 3-10. They will follow a structure similar to the first three labs described, building upon previous concepts when appropriate. The concepts covered also get more complex.

- Lab 3 – Analog I/O (more detailed ADC, DAC, and something else)
- Lab 4 – Synchronous Circuitry
- Lab 5 – PWM Control (plus another simple component)
- Lab 6 – Digital Logic
- Lab 7 – CapSense (capacitive touch buttons and sliders)
- Lab 8 – Communication (UART between two PSoCs)
- Lab 9 – Advanced Digital I/O
- Lab 10 – Advanced Analog I/O

I plan to write two labs and evaluate them using the techniques described in Section 7 before continuing with the other labs. I will adjust my approach based on the results of this early evaluation.

5.6. Final Project

The final project will combine concepts learned in previous labs together into a non-trivial real-world application. A significant number of PSoC hardware components should be used to create the final project. A minimum number of components to use will be determined at a later time.

I will write one or more final project examples, complete with an implementation, as a guide for how a final project may be devised. I will also provide several other suggested projects. The instructor may assign the final project, or encourage the students to propose their own projects to work on pending instructor approval.

6. Design Specification

Each lab will consist of a professor's guide including an example solution and a student lab manual.

6.1. Professor's Guide

This guide includes the same information in the student lab manual, with an example implementation at every step. Guidelines to evaluate the students' lab submissions will also be part of the guide.

The example solution will show an implementation of the lab that follows the instructions and produces the intended results. The code will be fully documented and the hardware diagram will be annotated to show how the PSoC hardware is connected to the auxiliary hardware. A readme will be included to list which generated files were modified (for example, interrupts generate an ISR, which the student needs to add code to).

6.2. Student Lab Manual

This is the document the students get for each lab. It will include the following sections.

- Objectives: The goal of the lab and what will be produced.
- Background: Information about the new concepts being taught in the lab. Includes detailed information about what the concept is and examples on how it can be used.
- Procedure: Instructions for completing the objective. The instructions will provide the steps and information needed to complete the lab. Screenshots of the IDE will be included when necessary. Important precautions, especially electrical safety, will be highlighted.

7. Evaluation

To evaluate the results of this project, the following will need to be tested:

- Do the example solutions work and produce expected results? To evaluate, thoroughly test the software running on the hardware, where each test suite is tailored to how each example should behave.
- Is the hardware stable enough for safe use by someone who may not be completely familiar with it? To evaluate, ensure that the hardware will be safe when used by students who do not have prior knowledge of using embedded computer hardware. Document and properly note important potential hazards, such as voltage differences, in both the labs and instructor notes.
- Will a student be able to learn how the platform works and develop on it with relative ease? This evaluation requires students that match the target audience and an instructor to hold the labs (to reduce creator bias). Perform trial runs of the platform with interested students, who will be instructed to complete one or more of the labs. The students will take a pre- and post-lab survey to categorize their prior knowledge and experience with the lab. The instructor will run the lab sessions using the instructor material while I observe. Once students and instructor are able to understand and complete labs with minimal issues, with more weight on students with little to no knowledge of computer hardware, this evaluation has passed.
- Ensure that the labs comprehensively cover the important I/O capabilities of a PSoC, especially ones commonly used with microcontrollers. I will have instructors of current courses that could potentially use the labs created by this project to evaluate the first two labs that I create. I will use their evaluation to guide the content creation of the rest of the project. I will also survey these instructors to get a list of important topic areas for other labs. At the end of the project, I will again seek instructor evaluation of the labs, to the extent that I can.

8. Deliverables

The following material will be produced by this project.

- Set of learning outcomes achieved by a student who completes all of the labs
- Setup guide

- General usage notes for professors and students
- Eleven complete labs (including lab 0)
    - Professor guide
    - Student manual
    - Example solution (fully commented and tested code and hardware design)
- Final project description, suggestions, and sample implementation
- Final report for this Master's Project

9. Schedule

This project will be executed across two semesters. The following is the proposed schedule, with room for adjustment.

Item	Notes	Due Date
Lab 0 written and peer reviewed	Labs are peer reviewed by chairperson. Includes all material for the lab.	10/11/2013
Full evaluation plan	Includes filling out necessary paperwork and starting the recruiting process	10/18/2013
Lab 1 written and peer reviewed		10/18/2013
Labs 0 and 1 evaluated	Depends on full evaluation plan.	10/27/2013
Lab 2 written and peer reviewed		11/01/2013
Setup guide written and reviewed	Experiences in evaluation needed to finalize	11/08/2013
Lab 3 written and peer reviewed		11/15/2013
Labs 2 and 3 evaluated		11/22/2013
Lab 4 written and peer reviewed		11/29/2013
Outline of all labs peer reviewed	Fulfills comprehensiveness evaluation	12/13/2013
Lab 5 written and peer reviewed		12/13/2013
Lab 4 evaluated		12/13/2013
Lab 6 written and peer reviewed	Delayed for winter break	01/10/2014
Lab 7 written and peer reviewed		01/17/2014
Lab 8 written and peer reviewed		01/31/2014
Labs 5 and 6 evaluated	Delayed for intersession	02/07/2014
Lab 9 written and peer reviewed		02/14/2014
Labs 7 and 8 evaluated		02/21/2014
Lab 10 written and peer reviewed		02/28/2014
Labs 9 and 10 evaluated		03/07/2014
Final project written and reviewed		03/28/2014
Usage notes finalized	Document is worked on throughout the project	04/04/2014
Course overview completed		04/04/2014
Master's project report completed		04/21/2014
Defense		05/04/2014

10. References

[1] K. Benkrid and T. Clayton, "Digital Hardware Design Teaching: An Alternative Approach," *Trans. Comput. Educ.*, vol. 12, no. 4, pp. 13:1–13:20, Nov. 2012.

This paper presents how a lab focused on digital circuit design (specifically, FPGAs) was created and how it was taught. Although this is geared toward hardware design rather than using hardware, it appears to be useful for researching methods used in computer hardware-based labs.

[2] A. Shoufan and S. A. Huss, "A Course on Reconfigurable Processors," *Trans. Comput. Educ.*, vol. 10, no. 2, pp. 7:1–7:20, Jun. 2010.

This paper presents an FPGA course that is geared toward Computer Science students. As an FPGA is a piece of hardware, it is similar to the kind of course I am looking to create. It has its differences, such as needing to know low level programming, but it is similar enough to be a valuable resource.

[3] Y. Dandass, "Teaching application implementation on FPGAS to computer science and software engineering students," *Computers in Education Journal*, vol. 18, no. 1, 2008.

This paper presents a way to teach Computer Science/Software Engineering students how to work with FPGAs using tools based on C. This paper can be used for determining how to explain certain concepts in the labs.

[4] P. Schaumont, "A Senior-Level Course in Hardware-Software Codesign," in *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*, 2007, pp. 7–8.

This paper discusses how designing hardware and software together is common in the real world and presents an undergraduate course that uses a CPU and FPGA to teach co-design. My labs are centered around co-design, although at a higher level than this course, so this paper can be useful as a reference point when creating labs.

[5] M. M. McGill, "Learning to Program with Personal Robots: Influences on Student Motivation," *Trans. Comput. Educ.*, vol. 12, no. 1, pp. 4:1–4:32, Mar. 2012.

This paper discusses how using robots in introductory programming courses can be a positive influence on student motivation. This paper could be used to reference a way that working with hardware can have benefits when defending the motivations of my project.

[6] "PSoC® 5LP," *Cypress*. [Online]. Available: http://www.cypress.com/?id=4562.

Home page for the PSoC 5LP. Start here to find specifications and documentation for the PSoC.

[7] "PSoC 5LP Development Kit Quick Start Guide," *Cypress*. [Online]. Available: http://www.cypress.com/?docID=40923.

Has an annotated image of the PSoC 5LP development kit. It is useful to show off its features.