Developing Multidisciplinary Laboratories with the Cypress PSoC

Masters Project Report

Stephen Hammack
sch7373@cs.rit.edu

Committee
Jim Vallino
James Heliotis
Zack Butler

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

Table of Contents

1. Overview

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 report that may be unfamiliar.

- ADC – Analog to Digital Converter: a device that converts an analog voltage 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, such as a digital camera image processor, rather than general use. 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
- 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.
- IDE – Integrated Development Environment: a program used for software development
- IC – Integrated Circuit: a set of circuits embedded into a single chip, rather than made of multiple discrete components
- I^2C – Inter-Integrated Circuit: a simple method of communication between a device and peripheral devices
- 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: a method of controlling power to a device, such as the brightness of an LED, by producing pulses a specified amount of the time
- 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
- USBUART – a method available to the PSoC where it can present itself as a serial device to a computer over USB, allowing a user or computer program to interact with it over a serial terminal

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 use software to interface with it. 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. These labs will take advantage of various ways to abstract the configuration of an FPGA.

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][5]. 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 temperature sensor, that may be used in a larger system like a cooling system.

In summary, the purpose of these labs is to teach high level computer hardware design and interactions between hardware 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 5LP

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. Since the hardware is configured using a visual programming approach, 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. Figure 1 indicates some of the hardware that is available. Note that the prototyping area allows custom hardware to be installed for use. There are also expansion ports, but these are not used in the labs.

5. PSoC Creator

PSoC Creator is the IDE that Cypress developed for configuring and programming most of their PSoC devices. It has been designed to simplify the process of configuring the hardware and generating a software interface (APIs) for the CPU to use. It is also possible to do advanced things with it, including creating custom hardware components with Verilog, but that is not necessary and is not covered in this project.

There are three steps to creating a project in PSoC Creator: design the hardware schematic, configure design-wide resources, and program the firmware to run on the CPU.

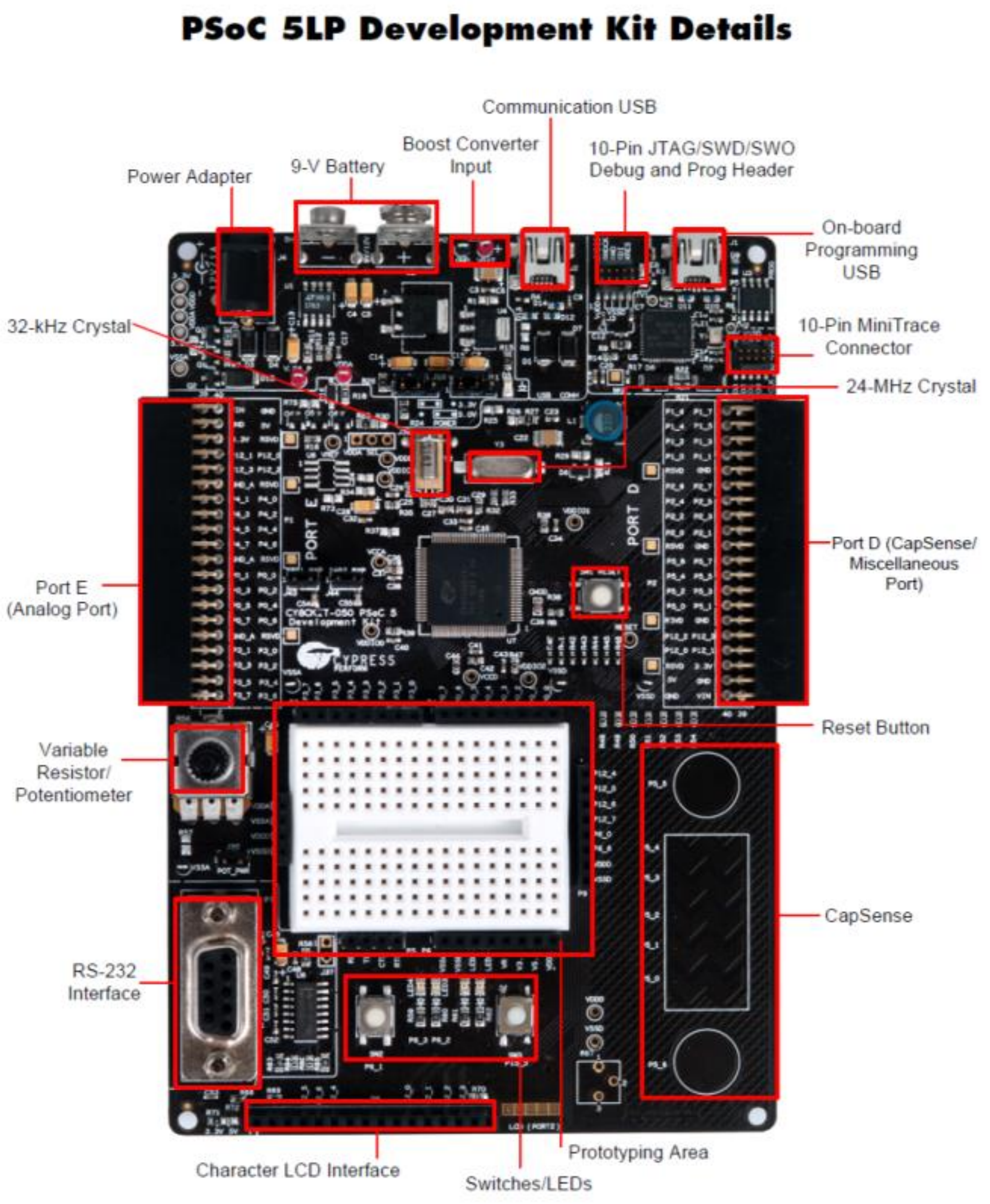


Figure 1: Diagram of the PSoC 5LP Development Kit [7]

5.1. Designing the Hardware

Because the hardware on the PSoC is reconfigurable, the developer needs to define how the hardware is configured. Hardware is configured using a schematic editor, as seen in Figure 2. This editor includes a library of components to be dropped onto the schematic, configured, and wired together. There are a wide variety of components, from parts that are fixed function digital or analog components to parts that are created with reconfigurable digital logic blocks. For the most part, these components can be connected together in any way desirable due to the flexible routing capabilities in the PSoC.

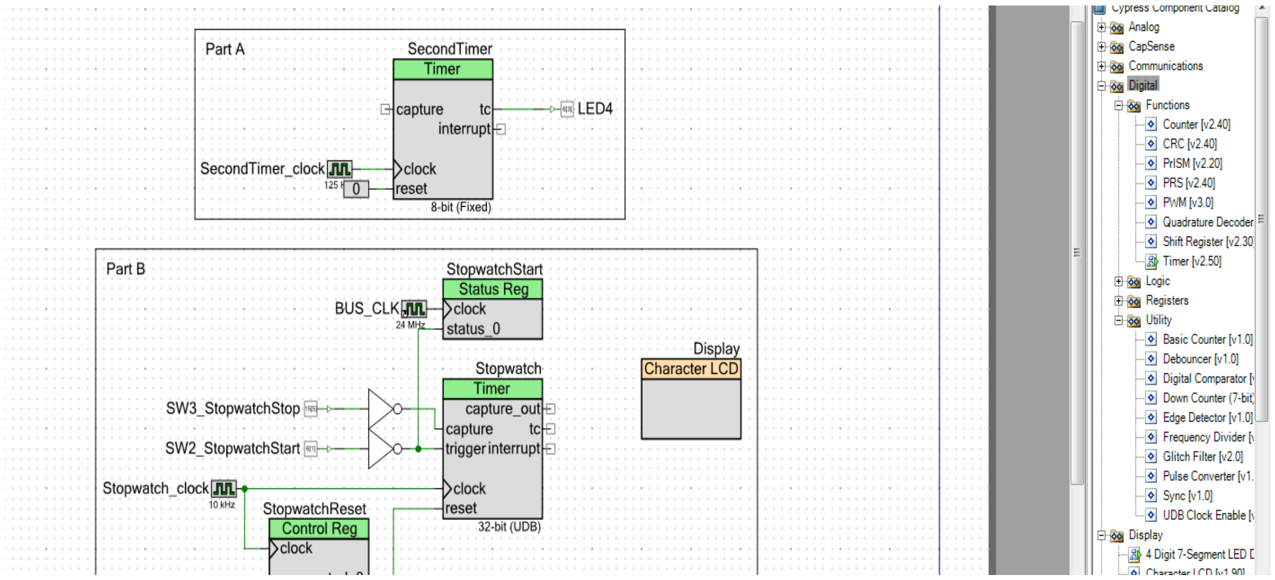
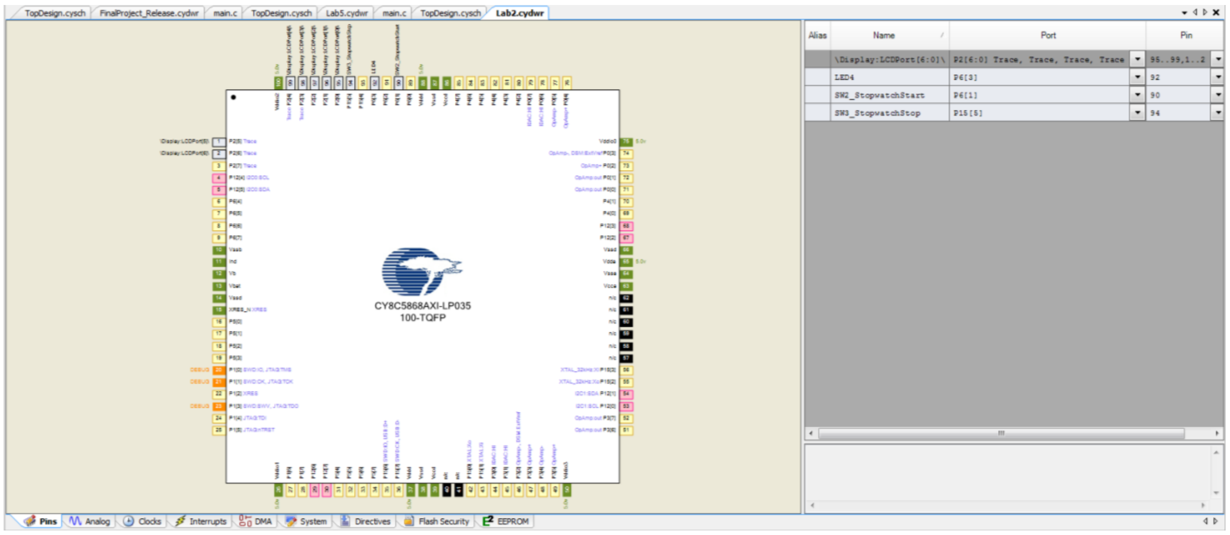


Figure 2: The PSoC Creator schematic editor

5.2. Configuring Design-Wide Resources

There are properties of the PSoC that are configured on a design-wide level, so there is a file for configuring these shown in Figure 3. For example, pin assignments, system clocks, and interrupt priorities are configured in this file. Most labs only need to assign the pins in this file.



Alias	Name	Port	Pin
	\Display:LCDPort[6:0]\	P2[6:0] Trace, Trace, Trace, Trace	95..99,1..2
	LED4	P6[3]	92
	SW2_StopwatchStart	P6[1]	90
	SW3_StopwatchStop	P15[5]	94

Figure 3: The design-wide resources editor

5.3. Programming the Firmware

After configuring the schematic and design-wide resources, the application needs to be generated. Application generation takes the schematic and design-wide resources and generates the necessary files for configuring the hardware accordingly. Additionally, code for interfacing the CPU to the hardware is created.

Now that the interface to the hardware is generated, the developer writes any code necessary for the CPU, which becomes the firmware for the device. The firmware usually has the responsibility of initializing the hardware, modifying variables within the hardware, and performing higher-level functionality like processing data. The programming language for the firmware is C.

After programming the firmware, the project can be built and loaded onto the PSoC. The program can also be debugged if necessary.

6. Products of the Project

6.1. Lab Manuals

The lab manuals are the primary product of this project. Each lab has been created to present new hardware concepts and demonstrate how to use them through a procedure to create an application using the PSoC. There are a total of six labs, plus two incomplete labs made of cut portions from labs that were too long.

The goals for each lab:

- Can be completed in two hours in a laboratory environment
- Includes sufficient background information to complete without a lecture (since creating lectures is outside of the scope of this project). Reading the background information is part of the two hours spent working on the lab.
- Is a meaningful learning experience for the students

The sections included in each lab:

- 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 of 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.

Topics covered in each lab:

- Lab 0 – Introduction to PSoC and PSoC Creator
- Lab 1 – ADC and basic digital I/O
- Lab 2 – Synchronous circuits, clocks, and timers
- Lab 3 – Interrupts and debouncers
- Lab 4 – USBUART and PWM
- Lab 5 – I^2C
- Lab 6 – Flip-flops and glitch filters
- Lab 7 – CapSense

See the appendix for full summaries of each lab. Note that labs 6 and 7 are not complete labs; they are cut content from labs 3 and 4, respectively. See the corresponding sections in the appendix for an explanation.

All of these labs only require the PSoC 5LP Development Kit except for Lab 5, which requires a peripheral device that uses I^2C. The lab is written to use a temperature sensor from Adafruit, which is linked to from the lab.

6.2. Instructor's Guides and Solutions

These guides supplement the lab manuals by having any information necessary to run the lab and suggested solutions. Some suggestions for evaluating the labs may also be included.

A complete solution that can be opened in PSoC Creator will also be included for each lab. These solutions are the projects created while developing the labs.

6.3. Final Project

The final project combines 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. The application should be implemented in hardware as much as possible, since the point of these labs is to teach hardware concepts.

Several suggested projects have been given along with a fully implemented project as an example. The instructor may assign a topic for a final project or encourage the students to propose their own projects to work on with instructor approval.

7. Final Project Example

Making a final project example has two purposes. The first purpose is to present an example final project that the instructor might expect from a student. The second purpose is to show the level of complexity of work that was done as part of this project.

7.1. AY-3-8910/YM2149 Music Synthesizer

The AY-3-8910 is a music synthesizer chip used in several home computers in the 80s, including the ZX Spectrum. The YM2149 is essentially the same chip (different manufacturer) used in the Atari ST. Its primary feature is the ability to produce three voices of music using square waves.

The components required for generating signals, choosing which signals to use, and producing the output are completely implemented in hardware on the PSoC. The software chooses a file from an SD card that contains the data for music playback and updates the hardware 50 times a second with the playback data.

This project has successfully replicated the synthesizer in the hardware using the Atari ST implementation, which sets the master clock speed and the hardware update rate, with the CPU in the PSoC controlling it. The only limitation is that some Atari ST music took advantage of using quicker interrupts to create special effects. The way those effects work isn't well documented and most music tracks don't use them, so special effects were not implemented.

See the appendix for more details about the specifications of the synthesizer and implementation details.

8. Lab Evaluations

To evaluate the labs, five student volunteers finished all six completed labs under my supervision. Of these, two were Computer Science majors, two were Electrical Engineering majors, and one was a Software Engineering major. All but one had little to no experience with using computer hardware. One had previously used a PSoC, but only once. Two of them did not complete Lab 5, but they completed all of the other labs.

Lab 0 was completed by the students without supervision, excluding the part that ran the lab on the hardware. That way, the students could have PSoC Creator installed and working on their laptops ahead of time since computers could not be provided. All other labs were each completed in one supervised session per sequential week, where each week had two different session times to accommodate everyone's availability. These sessions were scheduled to be two hours, although extra time was needed in some circumstances. The final project was not included in the evaluation, as that would require too much time from the students outside of the evaluation sessions to complete.

There were also plans to evaluate a few of the labs in a lab environment, but the instructor was not able to make the lab time available. Despite this setback, the evaluations performed produced useful data for improving the labs.

In addition to the student evaluations, my advisor reviewed each lab once or twice before evaluations and once more before declaring labs as final versions. The pre-evaluation reviews helped catch most of the errors before students evaluated them.

8.1. Process

Before starting the first lab, all students completed an opening survey, where they stated their major and rated their general knowledge of computer software and hardware, C, and the PSoC. They also chose a unique identifier so the results of surveys were anonymous.

Before starting a lab, each student completed a pre-lab survey asking which version of the lab they were using and their general knowledge of topics being covered in the labs.

During the labs, I listened to feedback from the students and assisted them as necessary as an instructor would. If any issues with the lab manuals arose, they were edited to fix the problems.

After the lab, each student completed a post-lab survey asking for their new level of familiarity with the topics covered by the lab along with several open-ended questions: what part was most useful, what part was least useful, confusing parts, and any suggestions for improvement.

8.2. Results

For detailed results per lab, see the appendices for each lab.

Labs 0 and 1 were well accepted, as they covered simple topics and were easy to understand. It may be beneficial to add in a short section about commonly encountered issues with the version of C used in PSoC Creator. An example issue: the variable used in a `for` loop cannot be declared in the `for` statement.

Lab 2 started to show some issues. One issue was that the lab took a bit too long for some people to complete, as it was the first lab that introduced a more complex application: a stopwatch. The code was not provided for this lab either; instructions for writing the code were provided instead.

Lab 3 was the longest lab. The average completion time was 2.5 hours, because the second half of the lab was taking the stopwatch from Lab 2 and re-implementing it with interrupts. The second half of Lab 3 has since been removed and placed into Lab 6. Another issue was that many students were skimming the instructions and skipping important parts. Even when I explicitly pointed at a part in the manual that they missed, sometimes they insisted they had done everything the manual had said at that point.

Lab 4 was better received by the student evaluators. It was made an appropriate length by removing the CapSense component and placing it into Lab 7, so this lab was completed in less than two hours. However, the instructions for programming Lab 4 were very exact and one of the students suggested that the instructions should just list out what the code should do rather than say exactly what to do. It also seemed that the students were not retaining much of the content of past labs, which suggests that the ways they are written are not conducive to learning the components being presented. These observations lead to writing Lab 5 differently (evaluating Lab 5 was not delayed due to this change).

Lab 5 was the best received lab because the programming section only stated what the program should do, so the students had to figure out how things work by using the background section alone. The background section was usually skimmed over and never looked back at again, but this lab gave them an incentive to learn what to do. Being explicitly told what to do does not encourage learning. The only downside is the lab took longer to complete than expected, albeit still a bit less than two hours on average, but it would have been faster if there was time to do some pre-lab prep work, like a lecture on the content.

9. Future Work

9.1. Improvements

Based on the results of the evaluation, the structure of these labs and their organization should be changed.

For the labs themselves:

- When the code is not provided (which happened in most parts of the labs starting in Lab 2), provide instructions that allow the student to figure out what to program. Labs 2-4 provide explicit instructions for what to program, which leads to the students not fully understanding what they are doing. Lab 5's evaluation results show that letting the student figure things out is much more beneficial for them, although it is more time consuming.

- It would be best if the labs were preceded by lectures on the material to be used in the labs. Not only can they go deeper than the labs are designed, but the students also demonstrated issues reading through the background section and retaining knowledge in a short period of time.
- Develop applications that are simpler than the stopwatch to replace it in labs 2 and 3.
- While developing a lab, be more mindful of how long it may take to complete that lab.

For the organization of the labs:

- Add a few labs that do not introduce new concepts. Rather, make them revisit old concepts by presenting a more challenging problem that combines concepts from past labs to complete. For example, insert a lab after Lab 3 that implements a stopwatch with interrupts. Labs like these should help the students prepare for the final project.
  - The current labs appear to be insufficient for preparing students to work on the final project by themselves based on feedback from the evaluators.
- There were only two other components not covered by the written labs (including the incomplete labs) that were considered for inclusion. These components are DMA (Direct Memory Access) and DAC (Digital to Analog Converter). The DAC was used in the final project example and DMA was considered for it, so it's suggested that these concepts should be put into labs. Most other components are either redundant, have very specific applications, are difficult to demonstrate without additional hardware, or require significant knowledge of electronics.
- It may be good to have a procedure in a lab that uses more digital logic, especially logic gates and multiplexers. These were used extensively in the final project example and they weren't used much in existing labs.
- There should be around 11 labs, as originally proposed, as those along with the final project should fill out a semester. Only six labs were fully completed because too much content was put into some of them.

9.2. Suggestions for Lab Environment

These labs require the use of a hardware platform, so each student needs access to one of these. There are two options:

- Obtain PSoC 5LP Development Kits through the Cypress University Alliance (free kits for professors) and allow students to use the kits in the labs. It would be unwise to allow the students to borrow the kits in case they break them or fail to return them.
- Have the students purchase the PSoC 5LP Development Kits for themselves. The kits are $99, which is reasonable compared to some other development kits.

In both cases, it would probably be best to purchase the temperature sensors for Lab 5 for the students to borrow. These sensors do not come with the headers soldered on, so it would be best if headers were soldered on the sensors before giving them to the students.

Additionally, there should be a computer lab with PSoC Creator available for students to use. One limitation of PSoC Creator is that Windows is required to use it and not every student uses Windows. Therefore, there should be a place where students in that situation can go to for their final project.

9.3. Other Hardware Options

Cypress offers two other, cheaper kits that utilize the PSoC 4 that are $25 and $4. While more affordable, using them would ultimately be more complicated than using the PSoC 5LP Development Kit. The $25 kit doesn't have much external hardware, such as the character display, so it would be necessary to develop something to attach to it to add more devices. Developing extra hardware like that may end up being more expensive than simply using the 5LP kit. The $4 kit has nothing extra, and it's meant to plug directly in to a USB port, so it would be difficult to connect anything to it. Finally, the PSoC 4, while containing the same sort of flexibility as the 5LP, is much more limited. For example, the PSoC 4 only has up to four blocks of programmable digital hardware, while the 5LP has up to 24. The extra capabilities of the 5LP make it more flexible, and it should make the final project easier since the students would not need to be careful about tighter limitations.

10. Conclusion

Working with the PSoC 5LP has been a great experience, as creating these labs required reading through many datasheets and understanding the functionality of many hardware components. The final project example in particular demonstrates this since it goes a bit beyond the effort required for a final project.

Responses to the labs have been generally positive, but there are several issues with them that need to be fixed. However, if these labs were revised to fix the issues, they could become a solid part of a college course. I am considering continuing development of these labs to implement the points covered in the Future Work section if any professors are interested.

11. 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.

[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.

[3] Y. Dandass, "Teaching application implementation on FPGAS to computer science and software engineering students," *Comput. Educ. J.*, vol. 18, no. 1, 2008.

[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.

[5] M. Lukowiak, S. Radziszowski, J. Vallino, and C. Wood, “Cybersecurity Education: Bridging the Gap Between Hardware and Software Domains,” *Trans. Comput. Educ.*, vol. 14, no. 1, pp. 2:1–2:20, Mar. 2014.

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

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

[8] “PSoC® 5LP: CY8C58LP Family Datasheet,” *Cypress*, 2013. [Online]. Available: http://www.cypress.com/?docID=47570.

Appendix A. Survey Information and Opening Survey

Survey results are presented in order of subject ID. Tables present questions with answers on a scale from 0 to 10: from completely unknown to master of the subject. When stating which version of the lab is being used, D stands for draft and B stands for beta, which is followed by the version number. The time elapsed between pre-lab survey and post-lab survey submissions are also presented in h:mm format, unless the student did not complete the post-lab survey immediately after the lab was finished. This usually happened when the lab took too long to complete and they had to leave. Lab 0 will not present elapsed time, however, since most of the lab was done outside of the lab session.

Only relevant, verbatim responses to open-ended questions will be presented for each lab. Responses that already resulted in an improvement in the lab will be skipped.

Here is information collected about the evaluation subjects before they started any labs. All of the subjects assigned themselves a unique identifier so they would remain anonymous.

- Subject 42 – Electrical Engineering
- Subject 327 – Electrical Engineering
- Subject 666 – Computer Science
- Subject 709 – Computer Science
- Subject 762 – Software Engineering

Opening Survey: Topic \ Subject	42	327	666	709	762
General knowledge of computer software	6	8	6	8	10
Knowledge of the programming language C	4	6	7	2	8
General knowledge of computer hardware	9	5	3	1	5
Familiarity with the Cypress PSoC	0	3	0	0	1

Appendix B. Lab 0

B.1. Description

This lab introduces PSoC Creator and the PSoC 5LP Development Kit to the students so they can get a feel for how development on the kit works. The result of this lab is an LED that turns on while a pushbutton is pressed and a count on the LCD display that is incremented when the pushbutton is pressed. The CPU handles all of the logic in this lab.

B.2. Evaluation

Response to this lab was positive overall because it is a simple lab. I asked what their familiarity with PSoC Creator was before and after the lab and the improvement was considerable. Of course, most of them had not seen PSoC Creator before, but an average 3-4 point increase (out of 10) suggests they were beginning to understand how things worked.

B.3. Survey Results

Pre-Lab Survey: Topic \ Subject	42	327	666	709	762
Lab version	B2	B3	B3	B3	B3
Familiarity with Cypress PSoC development	0	2	0	0	1

Post-Lab Survey: Topic \ Subject	42	327	666	709	762
Familiarity with Cypress PSoC development	4	6	4	2	3

- Subject 42
  - What parts of the lab did you find most useful?
    - It was nice to be able to use a schematic-based design system to add components to the general design, and interfacing was relatively simple. It was a really solid way of creating an FPGA core
  - Other comments
    - Very helpful in getting started with the PSoC Creator software and PSoC platform
- Subject 666
  - Any suggestions for improvement?
    - Unless you're assuming your students have experience with C, maybe giving some preliminaries on how C is different than Java would help (such as 0 = false, 1 = true).
- Subject 709
  - Any suggestions for improvement?
    - Important words in writeup should be bolded or underlined so as to not skip them.

Appendix C. Lab 1

C.1. Description

This lab introduces basic hardware design in the PSoC 5LP using basic digital I/O and an ADC (Analog to Digital Converter).

The result of this lab is an LED that turns on while a pushbutton is pressed without CPU intervention. Additionally, an ADC is connected to the potentiometer on the development kit, converting the voltage going through the potentiometer to a digital value, and the CPU is continually converting the value to mV and printing it to the display.

C.2. Evaluation

Response to this lab was positive overall because it is a simple lab. The main complaint is that the background information section was daunting, but that can't really be fixed when there is no lecture before the lab. Most stated that they learned a bit more about the topics in this lab.

C.3. Survey Results

Pre-Lab Survey: Topic \ Subject	42	327	666	709	762
Lab version	B1	B2	B2	B2	B2
Familiarity with general purpose I/O	8	0	2	1	3
Familiarity with analog-to-digital converters	7	1	1	0	0
Experience with choosing between a hardware and software solution to a problem	6	0	0	0	0

Post-Lab Survey: Topic \ Subject	42	327	666	709	762
Time between survey submissions	1:15	0:54	-	0:52	0:57
Familiarity with general purpose I/O	9	5	3	1	5
Familiarity with analog-to-digital converters	8	3	2	1	4
Experience with choosing between a hardware and software solution to a problem	7	2	1	1	4

- Subject 42
  - What parts of the lab did you find most useful?
    - Being able to implement solutions on a completely hardware basis is a very useful ability, especially since I am much more familiar with digital logic solutions.

- Subject 327
  - Any suggestions for improvement?
    - remove some of the intro stuff, it is a little daunting, though it does make sense after reading through it all. other than that, the lab make perfect sense. had a cool output, too.
- Subject 709
  - List any confusing parts of the lab.
    - the fact that you can't delcare variables in a loop in C should be mentioned.

Appendix D. Lab 2

D.1. Description

This lab introduces concepts of synchronous circuits through clocks and timers. Clocks are digital signals that oscillate between high and low a specified number of times per second. Timers are components that are generally used to generate periodic events by generating a signal after counting an input signal (usually a clock) a specified number of times.

There are two procedures in this lab, and two results. The first result is a simple timer that causes an LED (that is, a light) to blink once a second and the second result is a stopwatch that utilizes a hardware timer to keep track of time.

D.2. Evaluation

Some people started having issues with completing this lab. Subjects 42 and 709 took more than two hours to complete this lab because it was the first lab that did not provide much code to the students. Instead, detailed instructions for how to program the firmware was used. It seemed that there was some confusion with the detailed write-up, particularly with the stopwatch. This issue and the excessive length of Lab 3 led to the conclusion that the stopwatch should become a separate project meant to supplement what has been learned, rather than being the introduction for new concepts. In doing so, the amount of instruction provided in that lab could be reasonably reduced since the students would have more experience by then. However, changing Lab 2 to reflect this would cause most of the content to be removed, so an alternative task would need to be created in place of the removed material. Since all of the subjects had already completed Lab 2 when that conclusion was made, there was no time to fix Lab 2 and test it again on new subjects.

D.3. Survey Results

Pre-Lab Survey: Topic \ Subject	42	327	666	709	762
Lab version	B1	B2	B2	B2	B2
Familiarity with synchronous circuits	6	3	2	1	3
Familiarity with clocks	8	3	3	2	3
Familiarity with hardware timers	8	1	3	1	3

Post-Lab Survey: Topic \ Subject	42	327	666	709	762
Time between survey submissions	2:19	1:27	-	-	1:24
Familiarity with synchronous circuits	7	4	6	2	5
Familiarity with clocks	8	5	4	2	5
Familiarity with hardware timers	8	3	4	2	5

- Subject 42
  - Any suggestions for improvement?
    - Some more clarity on why different blocks are used in the labs (like the registers) would be helpful
- Subject 666
  - Any suggestions for improvement?
    - I think you might consider separating the information about PSoC Creator from the background technical information, or presenting it with the activities instead of before so the student isn't flipping back and forth. Also, perhaps instead of giving a strict procedure for creating the design (put this pin here, hook it up to this timer), it might be better to present the design, describe what the design is supposed to do and how the individual components are supposed to work together and let the user figure it out.

- Subject 709
  - Any suggestions for improvement?
    - The beginning portion of the write-up contained a lot of information that had little context at the moment. As a result, it was difficult to absorb any of the information as I could not understand it's relevance until I started the lab. It is fine to introduce subjects in the beginning, but when they are explained in such detail without context to the lab, it can be difficult to follow.

Appendix E. Lab 3

E.1. Description

Lab 3 has been split up due to the results of the evaluation.

The current version of the lab introduces interrupts, debouncers, and flip-flops. Interrupts are a signal to the CPU that causes it to stop what it’s currently executing, run a routine to handle the interrupt, and resume normal execution. Debouncers are a simple form of input filtering that ensures button presses are read cleanly. Flip-flops are synchronous devices that store one or more bits of information that is modified based on inputs to it.

The result of this lab is a debounced pushbutton that, when pressed, toggles the state of an LED and increments a count on the display. The LED and debouncing take place in hardware while the count is incremented in the CPU via an interrupt.

The cut content from this lab has been moved to Lab 6. This content involved taking the stopwatch from Lab 2 and modifying it to use interrupts, rather than polling on the CPU side.

Filtering was also applied to the pushbuttons using glitch filters. This proved to be far too much for the students to do, which is why it was cut.

E.2. Evaluation

The primary issue with this lab is that, in its original form, it was far too long. Subject 42 failed to complete the lab (finished through part C) and subject 709 took about three hours to finish, and as a result neither had time to immediately submit a survey. In particular, the part where the stopwatch is improved on with interrupts and glitch filters was the part that took too long. In response, those parts were removed and turned into Lab 6.

E.3. Survey Results

Pre-Lab Survey: Topic \ Subject	42	327	666	709	762
Lab version	D4	B2	B2	B2	B2
Familiarity with interrupts	4	1	4	1	5
Familiarity with debouncers	8	0	0	0	5
Familiarity with flip-flops	7	4	4	0	5
Familiarity with glitch filters	7	1	1	0	1

Post-Lab Survey: Topic \ Subject	42	327	666	709	762
Time between survey submissions	-	1:53	2:33	-	1:43
Familiarity with interrupts	8	3	5	3	6
Familiarity with debouncers	9	2	3	2	6
Familiarity with flip-flops	9	4	4	2	6
Familiarity with glitch filters	8	2	3	2	3

- Subject 42
  - What parts of the lab did you find most useful?
    - The ability to implement functions in hardware and have a good bit of control on the software side was particularly useful, and allowed for some interesting combinational circuit and logic designs. The debouncing and interrupt portions of the lab, although slightly complicated, were really interesting and useful.
  - What parts of the lab did you find least useful?
    - Having to completely re-arrange the schematic to fix a glitch that was intermittent. Glitch filtering could have been shown off in a more meaningful way.
  - Other comments
    - The lab seemed a bit long, Part C was long enough to be a lab unto itself. The combining of somewhat unrelated topics could have been implemented better, but overall a good lab.

- Subject 666
    - What parts of the lab did you find most useful?
        - Migrating the code from the main() to the interrupts helped me understand the difference between polling and interrupts, plus how PSoC projects work.
- Subject 709
    - Any suggestions for improvement?
        - Possibly bolding really important parts of a paragraph might help. While reading long paragraphs, my eyes tended to skip over certain parts of a sentece.

Appendix F. Lab 4

F.1. Description

Lab 4 was split up before being evaluated due to the evaluation of Lab 3. The cut part involved CapSense, or capacitive touch input, and has been moved into Lab 7.

This lab introduces USB UART and PWM (Pulse-Width Modulation). USB UART is a method of serial communication with a computer over USB, which in this case is used for user input over a serial terminal. PWM is a simple control signal that periodically drives a pin high for a portion of the period (known as a duty cycle) and drives low otherwise.

The result of this lab is a program that waits for the user to enter a number on the terminal from 0 to 255 (accepts out of bounds inputs too). When the number is entered, the CPU modifies the duty cycle of a PWM component that is driving an LED. As the duty cycle increases, the LED gets brighter, as it is on for longer. There is also no perceivable flicker, because the period is short.

F.2. Evaluation

This lab was generally well accepted, especially since it was shortened before it was evaluated. However, after this lab it became clear that the procedure for this lab and previous labs have been too extensive. That is, the procedures were written with every detail necessary to complete the project, excluding some recurring details. Writing the labs this way made it so the students were only going through the motions and not actually learning what they are doing. Fixing this issue would require a total rewrite of the procedure and everyone had already evaluated the lab, so this issue has remained unresolved. However, Lab 5 was written with this feedback in mind, so fixing the labs in this way has been investigated.

F.3. Survey Results

Pre-Lab Survey: Topic \ Subject	42	327	666	709	762
Lab version	D4	D3	D3	D3	D3
Familiarity with PWM	10	4	0	0	4
Familiarity with serial terminals	8	3	4	0	4
Familiarity with parsing console input	6	1	8	8	8

Post-Lab Survey: Topic \ Subject	42	327	666	709	762
Time between survey submissions	1:51	1:47	2:07	-	1:25
Familiarity with PWM	10	5	3	2	7
Familiarity with serial terminals	8	4	6	2	7
Familiarity with parsing console input	7	2	8	8	7

- Subject 42
  - What parts of the lab did you find most useful?
    - Being able to serially connect to the psoc was a useful trait, especially if I wanted to write an application that monitored the serial input from USB
  - What parts of the lab did you find least useful?
    - PWM wasn't really that big of a part in the lab, and was only very lightly used. Some more in-depth review of the hardware options would be nice
- Subject 666
  - Any suggestions for improvement?
    - Giving step-by-step instructions for the code, while better than providing the code, made it made it somewhat too mechanical; depending on how experienced programmers your students are, you might want to consider telling them at a more abstract level how the program is supposed to function; as in, giving the interface and functions, and the desired specs of the terminal (maximum of 3 characters per line, backspace deletes, newline alters display) and letting them figure out how to program it. Actually thinking about how to program it, instead of simply converting the instructions into C syntax, might make them think more about what is actually happening.
- Subject 762
  - What parts of the lab did you find least useful?
    - Code needs to be provided as a .c file because copying from a pdf doesn't work.
  - Other comments
    - This lab was fun, and almost felt applicable.

Appendix G. Lab 5

G.1. Description

This lab introduced I^2C, a simple two-wire serial communication protocol that connects one or more master devices to one or more slave devices. This lab was written differently from all past labs; the programming section simply listed what the program should do to complete the lab. The students need to depend on the background section in order to complete the lab. This change was made in response to the feedback from Lab 4.

This lab has the student write a program that continuously reads the temperature from an I^2C temperature sensor and displays it on the character LCD.

G.2. Evaluation

Subjects 327 and 762 did not complete this lab.

This lab was well-received by the subjects. They stated that they learned a lot more from this lab than past labs because they were made to learn about how I^2C worked so they could implement it.

One problem is that it took them a long time to complete this lab, but most completed within the two hour time period. It would probably run smoother in a real lab setting where the student should have a lecture beforehand, so they will be somewhat familiar with the topic before starting the lab.

Another problem is that the subjects took a while to understand that the method of reading from the device utilizes I^2C, but the method itself is not part of I^2C. To be more specific, reading from the device requires writing a value that indicates what the device should return, then reading from the device. I^2C does not strictly work this way; it just provides the method for reading and writing from/to devices. Having a lecture before doing the lab should also help clear up this confusion.

G.3. Survey Results

Pre-Lab Survey: Topic \ Subject	42	327	666	709	762
Lab version	D3	-	D2	D2	-
Familiarity with device communication protocols	6	-	2	2	-
Familiarity with I^2C	7	-	0	0	-

Post-Lab Survey: Topic \ Subject	42	327	666	709	762
Time between survey submissions	1:58	-	1:50	2:13	-
Familiarity with device communication protocols	8	-	4	2	-
Familiarity with I^2C	9	-	3	2	-

- Subject 42
  - What parts of the lab did you find most useful?
    - The freedom of choosing my own implementation of the circuit made the coding process a good bit easier. Also, the fact that it was centered around only one idea helped.
  - List any confusing parts of the lab.
    - Some of the generation of the code and interfacing with the I2C was a bit vague, but otherwise fine.

- Any suggestions for improvement?
    - This lab was an improvement in that it centered on one topic, and allowed the end user to figure out the best approach to solving the problem
- Subject 666
  - What parts of the lab did you find most useful?
    - The table of instructions was actually useful this time now that we had to come up with the code ourselves. In general, making us implement the code ourselves made the background information much more important to read and understand. Also, the tips were useful as well.
  - Any suggestions for improvement?
    - To make students read more carefully, or force them to read supplementary material in the background that might not be strictly necessary to completing the lab, you might ask questions about that material.
- Subject 709
  - What parts of the lab did you find most useful?
    - I found that the the lab being more open ended then the previous labs forced me to better understand the material. I would prefer if more labs were written this way, instead of every step being given to us.

Appendix H. Labs 6 and 7

Both of these labs are incomplete, as they are just the content cut from labs 3 and 4. Thus, neither of them has been evaluated. They only exist to present the content that was created for these labs, since they may still be useful in a future improvement to the labs.

Lab 6 has two parts. The first part takes the stopwatch from Lab 2 and modifies its implementation so it uses interrupts instead of polling. The second part uses glitch filters to fix a minor problem with how the pushbuttons are read. Based on feedback from the original version of Lab 3, modifying the stopwatch in this way was beneficial to understanding the difference between polling and interrupts, but doing so simply took too long. It would probably be a good idea to make a simpler application in Lab 2 that is updated in Lab 3 with interrupts. The glitch filter part, on the other hand, was not accepted well at all, as it fixed a minor problem that would not be something a normal user would encounter. This section should be removed entirely, but it may be beneficial to introduce glitch filters with another application.

Lab 7 introduces CapSense, which is Cypress' capacitive touch solution. This was removed from Lab 4 because that lab would have been too long otherwise. Additionally, this lab only told the student to copy a pre-calibrated component from another project so it would just work. It would be more beneficial if CapSense was presented in its own lab. That way, the student would have time to learn how to tune the sensors and gain a better understanding of how they work.

Appendix I. Evaluation as a Course

After evaluating Lab 5, I asked all subjects to complete a closing survey about all of the labs together as a course. Subject 327 did not complete this survey. Here are the results.

Closing Survey: Topic \ Subject	42	327	666	709	762
General knowledge of computer software	7	-	7	7	9
Knowledge of the programming language C	7	-	8	3	8
General knowledge of computer hardware	8	-	5	2	5
Familiarity with the Cypress PSoC	7	-	4	4	5
Overall satisfaction for taking these labs	8	-	6	7	9
Rate preparedness for doing a final project	9	-	5	4	0
Interested in taking a course with revised labs	Yes	-	Yes	Yes	Yes

- Subject 42
    - Which of the labs did you find the most useful?
        - The last lab was extremely useful, as well as the PWM lab, I can see myself using this for a future project.
    - Do you feel that the content of some of the labs should have been presented in a different order? Please elaborate.
        - Yes, some of the lab structures were a bit unrefined, in that somewhat unrelated topics were heaped together. However, with more refinement and focus on the material, the labs could easily be very helpful to any computer science or software engineering student.
    - One of the conclusions of this project is that there should be labs that present a problem that does not use anything new, but rather reinforces things already learned. Do you agree?
        - Yes, having the student solve problems with the device opens up significantly more solutions to the same problem, and helps the student learning experience by finding different ways around the same problem.
    - Any other suggestions for improvement?
        - Some more refinement on the lab content would be nice, basically to focus each week on one topic and thoroughly explain the topic in a concise format.
- Subject 666
    - Which of the labs did you find most useful?
        - The first one or two labs were useful for getting acquainted with PSoC, and I learned more from the final lab than the previous 3 simply because of the approach taken.
    - Which of the labs did you find least useful?
        - For labs 2-4, while for the most part I understood ~why~ the design worked, I didn't understand the fundamental concepts that we were ostensibly learning because it wasn't necessary to understand the background.

    - Do you feel that the content of some of the labs should have been presented in a different order? Please elaborate.
        - No, I felt like the order of the labs was appropriate; introducing PSoC first and going step by step, then introducing fundamental concepts such as timers and filters. I think
    - One of the conclusions of this project is that there should be labs that present a problem that does not use anything new, but rather reinforces things already learned. Do you agree?
        - Yes, although there's something to be said for improving a design by introducing new concept (i.e. modifying the stopwatch to use interrupts instead of polling)
- Subject 709
    - Which of the labs did you find most useful?
        - I found lab 5 to be very useful as since the lab did but give exact instructions, it forced me to have a better understanding of the material.
    - Which of the labs did you find least useful?
        - Labs 1 and 2 contained a lot of background information such that it made the lab difficult too follow.c though this information was not useless, it works likely be beneficial to introduce less topics in each lab.
    - One of the conclusions of this project is that there should be labs that present a problem that does not use anything new, but rather reinforces things already learned. Do you agree?
        - I do agree that it would be beneficial to have labs that reinforce material we leaned earlier. This works test if we are actually learning from the labs. A good idea might bee to combine a few topics from previously and have a new jab that incorporates that information, though I understand that might be difficult to do in such a short time.
    - Any other suggestions for improvement?
        - I noticed that the wiring instructions for the schematics would tell us what to wire, but but necessarily what's going on with them. I for one am not very familiar with wiring and connecting electronics, and I would have found it interesting to have some sort of explanation as to how The connections work on a lower level. Though this is likely something that would require more time to cover in a lab

Appendix J. Final Project Implementation

J.1. AY-3-8910/YM2149 Features

The AY-3-8910 is a programmable sound generator that was developed for microcomputers and game consoles in the 70s and 80s. The YM2149 is the same chip with one inconsequential difference, except it was made by Yamaha. The YM2149 was used in the Atari ST, which is what this project is simulating.

Features of the synthesizer chip:

- All of the timing, including the tone frequency, is controlled by a clock input
- Three square wave tone generators
- One pseudo-random 1-bit noise generator with variable update rate
- One volume envelope generator
    - Ranges from 0-15
    - Ten envelope shapes, where the volume increments, decrements, or stays the same based on the shape. The time between increments or decrements depends on the specified period, which divides the main clock
- Three voice mixers, where each voice has:
    - One tone from a tone generator, either on or off
    - Noise from the noise generator, either on or off
        - If both tone and noise is on, the noise is ORed with the tone
    - Two options for setting the volume of the voice
        - Fixed value, 0-15
        - Current value of the volume envelope
- Three DACs to output each of the voices

Features specific to the Atari ST:

- The clock input to the YM2149 is 2 MHz
- Registers on the YM2149 are updated 50 times per second

J.2. PSoC Hardware Design

All parts of the YM2149 are implemented in the hardware, short of using an external interface for modifying its state. The state of the hardware is controlled by the CPU instead of an external device. For more details of how the hardware is designed and pictures of the schematic, see the full write-up of the example project.

Some things of note:

- This implementation uses many multiplexers and logic gates. There are no labs that extensively present any of these elements, so it would be a good idea to create a lab that uses these.
- Other components not covered in labs: Sync, Digital Comparator, Basic Counter, and emFile (used for SD cards). These components could be discovered by the student on their own though. For example, Sync is only necessary when clocks are used a certain way, so it wouldn’t be very practical to create a lab that uses it.
- I am guessing most students who would take the labs as they are currently written would be unable to do a final project as complicated as this one. Revising the labs to be more like Lab 5 should help. Even so, it probably isn’t reasonable to expect many students to complete a project this large. In particular, the envelope generation portion is fairly complex to someone new to combinational logic. It may be more reasonable if the student implemented parts like this in the software with the understanding that they could improve the project further by moving that logic into hardware.

J.3. PSoC Software Design

In order to play music on the hardware, the software has to update the hardware. The YM2149 has 16 registers, and there are music files for the Atari ST that have 50-times-per-second register dumps for the music chip. Therefore, the software updates the hardware based on a register dump 50 times per second.

These register dumps are too large to store in the flash memory on the PSoC 5LP, so these are instead placed on files on an SD card. PSoC Creator includes the ability to access file systems on an SD card.

The main loop is a state machine. First the loop will allow the user to select which music file to playback using the pushbuttons. Then, after a user selects a file, it will continually fill up a circular queue (or wait for an entry in the queue to open up if it's full) that holds a section of register dumps from the SD card. Meanwhile, an interrupt that is triggered 50 times per second pops a dump off the circular queue and updates the hardware based on the registers. After the file runs out and the interrupt exhausts the queue, the program turns off the synthesizer's voices and asks the user to select another file.

J.4. Results

When comparing the output of the PSoC to playback of an emulated YM2149 on a PC, the music sounds identical. The only exception is that some Atari ST songs took advantage of advanced techniques that allowed the YM2149 to output more complex audio by updating the registers quicker than 50 times per second, thanks to the speed of the Atari ST's CPU. For example, due to how audio volume is handled, the chip can be manipulated to produce 4-bit PCM audio samples. The method for doing this and other effects is not well documented, so they have been excluded from this implementation. Songs that use these effects usually sound fine otherwise, they are just missing the extra effects.

This project could also be modified to simulate the AY-3-8910 that came in the ZX Spectrum. To do so, the clock speed and register update rate would have to be changed, as those are both slightly slower than the Atari ST. Additionally, it would be necessary to look up how to parse the register dump files for the ZX Spectrum, or how to convert them to an easier format to parse.

Appendix K. PSoC 5LP Features

The following is a simplified list of the features of the development kit's PSoC 5LP [8].

- 32-bit ARM Cortex-M3 CPU
  - Up to 67 MHz
  - 256kb flash, 64kb SRAM
  - 24-channel DMA
- Low voltage and low power modes

- Versatile I/O
    - 72 I/O pins; 62 of those are GPIO
    - All GPIOs can be routed to any digital or analog peripheral
    - CapSense capabilities on any GPIO pin
    - Multiple drive modes
- Digital peripherals
    - 24 programmable universal digital blocks (UDBs)
    - Full-speed USB interface; PSoC can act as a USB device
    - Four 16-bit timer, counter, PWM fixed function blocks
    - Digital filter block for simple digital signal processing
    - Other fixed function peripherals
    - Large library of peripherals that use the UDBs
- Analog peripherals
    - One delta-sigma ADC and two SAR ADCs
    - Four 8-bit DACs
    - Four comparators
    - Four op-amps
    - Four configurable analog blocks
    - CapSense hardware
- Clock system
    - Internal oscillator for generating the master clock internally. Accuracy generally reduces as it runs faster
    - 4-25 MHz crystal oscillator input for higher master clock accuracy
    - Internal PLL for generating faster clock speeds from the internal or crystal oscillator
    - 32.768 kHz watch crystal oscillator input for RTC
    - Low power internal oscillator running at 1, 33, or 100 kHz