New Orleans JAZZED about Engineering Education

Paper ID #16481

Leveraging New Platforms to Provide Students with a Realistic SoC Design Experience

Dr. Andrew Danowitz, California Polytechnic State University, San Luis Obispo

Andrew Danowitz received his PhD in Electrical Engineering from Stanford University in 2014, and is currently an Assistant Professor of Computer Engineering at California Polytechnic State University in San Luis Obispo. His engineering education interests include student mental health, retention, and motivation.

Antonio Leija, California Polytechnic State University, San Luis Obispo

Antonio Leija is now a Test Engineer at Green Hills Software in Santa Barbara, CA. He attended California Polytechnic University, San Luis Obispo for a master's degree in Electrical Engineering, focusing on embedded systems and digital design.

Leveraging New Platforms to Provide Students with a Realistic SoC Design Experience

Abstract

Recently there have been a slew of digital design products released that promise to simplify the task of giving students a real-world System-on-Chip (SoC) design experience. These "programmable SoCs" from companies such as Xilinx, Cypress, and Altera combine modern multi-core ARM processors connected to an FPGA through a widely used SoC interconnect standard. This paper discusses a Real Time Embedded System Course I designed that uses the Xilinx Zynq platform to give students first-hand experience with modern System-on-Chip design methodologies and the challenges that designers face in both hardware and software bring-up for a modern IP-based design.

The first portion of this paper discusses how students were trained to use the Zynq platform. The first weeks of the class were dedicated to teaching students the basics of real-time system and custom hardware design. Students used a Zynq-based port of Free-RTOS to learn about Real-time operating systems. Through a series of laboratory assignments, students are taught how to interface the RTOS with custom hardware that they place on the FPGA portion of the chip. The course material developed for this portion of the class will be posted online so that other educators may use it in their teaching.

The second part of this paper discusses some of the projects proposed and completed by students, and any difficulties the students faced along the way. From two weeks into the class, students are asked to form groups of up to four and propose a final project. For their final project, students are required to design and build a complete working system of their choice. Their final project is required to make use of both the processor running RTOS and at least one custom IP block running on the FPGA.

In the final section of this paper I examine student feedback for the course, and comment on some of the challenges I faced in integrating the Zynq PSoC platform, and its corresponding development tools, into the classroom.

Introduction

System-on-Chip (SoC) design methodology has come to dominate the space of mobile computing. SoCs incorporate traditional microprocessor cores for running general purposes programs, and any number of highly-efficient digital accelerator blocks to handle certain specialized tasks. Even traditional desktop processor manufacturers increasingly incorporate more and more SoC-like functionality on their devices with modern processors from both Intel and AMD incorporating embedded graphics, memory controllers, and other functionalities on dye [1,2]. Given the potential for cost, performance, and energy benefits from integrating different digital blocks onto a single die, it is likely that the SoC design methodology will remain a major fixture in industry for some years to come. As a result, it would be beneficial to provide undergraduate and graduate students with a representative SoC design experience so they can familiarize themselves with this important technology.

The challenge of teaching modern SoC design, however, is that SoCs are incredibly complex devices, incorporating complete software stacks, multiple processor cores, specialized high-performance on-chip interconnects, and a mix of proprietary and commercially available digital intellectual property (IP) blocks that must all work perfectly in tandem to create a fully functional system. The complexity is such that as of 2011, non-recurring chip design costs were routinely reaching the tens of millions of dollars [3]. Given these constraints, how can we as educators hope to provide a meaningful SoC design experience in a 10-week quarter?

This paper attempts to answer that question by detailing a 1-quarter SoC design course I piloted in Spring quarter 2015, CPE 439 Real Time Embedded Systems. As discussed in Enabling Technology, I based the course around a new type of technology called a Programmable SoC (PSoC). PSoCs abstract away many of the low-level design decisions and development work that goes into a traditional SoC bring-up, allowing students to focus on developing custom IP, connecting it to the processor through an industry standard interconnect bus, and writing driver software to control and synchronize the various IP blocks. In Course Curriculum, I discuss both how I train the students to implement system designs on their PSoCs and the open-ended project-based course structure I use to motivate the students to build their own functioning SoC-based embedded systems. Finally, in Project Results and Student Feedback, I discuss the results of the pilot class, including overall student feedback for the course.

Enabling Technology

Selecting a suitable development platform is key to providing students with a "representative" SoC design experience. The flexibility and limitations of the platform determine which aspects of SoC design students are able to experiment with. My goal with CPE 439 is to expose students to how SoCs are architected, so I determined that the development board must allow students to integrate custom or provided IP blocks, and must expose students to industry standard SoC interconnect interfaces such as ARM's AXI [4]. I also wanted a platform that provided a stable and predictable API for programming the processor and accessing any custom IP blocks. To give students an idea of how SoCs can be used in the field, I wanted a hardware-based development board that they could use as the basis of their own embedded system designs, rather than a simulation-based SoC development tool. While important, I considered the low-level aspects of SoC design such as placement and optimization of the interconnect routing, chip layout, and low-level software bring-up to be beyond the scope of what could be accomplished in a 10 week course, so I wanted a platform with a toolset that would automate these as much as possible.

Given my constraints, I based the course around a relatively new type of product called a Programmable SoC (PSoC). PSoCs—available from companies such as Xilinx [5], Altera [6], and Cypress [7]—are devices that incorporate hard processor cores on a die with programmable logic. I chose the Xilinx Zynq-based Digilent Zybo board [8] due to its low-cost, variety of I/O ports, use of the AXI interconnect standard, and Cal Poly's use of Xilinx products and design tools in other courses.

The digital design flow for the Zybo is very similar to Xilinx's FPGA design flow, relying on the same tool, Vivado [9], for creating IP blocks and connecting them to the processor core. Once the digital layout is complete, the Xilinx SDK tool provides a basic software runtime for the processor and read/write based access functions for any user-added digital peripherals. Using these tools, a student can design and program a very rudimentary SoC in a few hours.

An additional benefit of the Xilinx solution is that it is compatible with FreeRTOS [10], a free, open-source, real-time operating system that had been used in previous Cal Poly course offerings. FreeRTOS can be put on the Xilinx PSoC with minimal effort, and I have included a video and lab resources for teaching students how to do this at http://adanowitz.com/home/real-time-embedded-systems. With the Xilinx toolchain handling the low-level details of system construction, and with a full real-time operating system to build off of, students are left to spend their time exploring SoC optimization and how SoC design decisions affect overall system performance.

Course Curriculum

The primary goal of offering CPE 439 is to give students a representative SoC design experience that introduces them to the realities, both good and frustrating of developing a modern embedded system. Beyond that, however, with the increasing commoditization of PSoC technology (the Zybo board costs under $200 without academic discount), and the incredible amount of computational ability that this technology represents, I wanted to show my students that PSoCs are great platforms for engaging in independent learning, tinkering, and invention even after they graduate. To this end, I built the course around a project-based self-directed study model. The only prescriptive coursework required in the class is a series of four introductory labs designed to walk students through the construction and programming of a basic real-time embedded system. From day one, however, students are set to work on an open-ended "final" project.

Introduction to PSoC

I worked to minimize the amount of time students spent on prescriptive coursework so they could focus on the project. Therefore, I limited formal training on the PSoC development board to four simple lab assignments. The assignments, summarized in Table 1, attempt to ease students into SoC design by starting with an experiment they are familiar with—blinking LEDs on a development board—and slowly adding to it. With each of the labs, students also gain familiarity with the various tools in the Xilinx design chain used to fully program the PSoC, including Vivado and the Xilinx SDK. These labs also have side benefit of exposing students to important, but sometimes overlooked aspects of embedded system design such as the creation of a Board Support Package—the set of APIs and libraries that allow them to control various processor peripherals. The labs were given 1-per-week for four weeks.

I created a series of lectures to go along with the labs to introduce students to the key concepts and programming paradigms of Real Time Operating Systems, the benefits of SoCs, and the types of algorithms that benefit most from hardware acceleration. These lectures are rudimentary by design, and are delivered with the expectation that students will read the FreeRTOS manual or consult other sources as needed to supplement their knowledge.

When designing and pacing this curriculum, I ran into the challenge that my students had varying levels of academic preparation for this course. This course is offered as a technical elective to both Computer Engineering students, who have taken courses on Operating Systems and C programming, and purely Electrical Engineering students who have much less formal preparation. As a result, while a proportion of students are able to complete the labs early with no outside help, there is still a sizeable portion of students who struggle to learn and apply the new material inside of the one-week per lab schedule that I have set. I revisit this topic in Project

Results and Student Feedback, but this issue of varied background is still one that I am attempting to find a solution for as I revise and continue to offer this course.

Table 1. Summary of labs used to introduce students to the PSoC

Lab	Description	Concepts explored
RTOS and You	Program Zybo with FreeRTOS	Configuring processors and IP blocks in Vivado Board support packages Software development and debug flow
Blinky	Use FreeRTOS to control an LED connected to a processor pin at various frequencies	Memory-mapped I/O access with Xilinx APIs FreeRTOS tasks and queues
Blinkenlights	Connect an LED to the processor through the AXI peripheral bus and control	Configuring AXI peripheral bus in Vivado Instantiating AXI-connected registers Accessing digital peripherals through Xilinx APIs
Some Verilog	Build a finite state machine (FSM) to control the LEDs. Use the processor to control the FSM	Connecting custom HDL blocks to the AXI bus Building software controllable digital blocks HDL review ***Students have a simple SoC!***

The Project

The project is the primary focus of the course. On the first day of class, students are given a little under two weeks to form teams of up to four and propose a real-time embedded system that they would like to build. The rest of the quarter (outside of time spent on the four introductory labs) is dedicated to implementing, documenting, and demonstrating this system. To ensure project quality, the final project assignment advertises five "Criteria for success":

1. Use at least one custom peripheral implemented in the FPGA portion of the PSoC board
2. Use a software program running on FreeRTOS to control your overall application
3. Make use of the Real-time guarantees in RTOS for functionality
4. Include a means for your application to display results/transfer data/or otherwise interact with the outside world
5. Create an innovative, novel, or otherwise comprehensive system worthy of being a quarter long project

While the project requirements may seem overly open-ended and the second-week project proposal may appear rushed, I intentionally designed the project this way to try to maximize student learning. I left the project open-ended to encourage students to build SoCs for domains that they are already interested in, with the hope that this would make students put more effort into the project. I require a project proposal by week two since by that point students are still learning the basics of the PSoC platform, and are less likely to have preconceived notions about the sorts of projects that are "too hard" to accomplish.

To help mitigate the risk that students will propose an overambitious project, I carefully read each project proposal, providing general project advice and suggestions on reducing project

scope where necessary. I also let students know that they can renegotiate project scope throughout the quarter with no grade penalty.

After the initial proposal stage, I act as more of a consultant than a teacher. When students have questions about board functionality or how to create a system, I point them to documentation and other resources that might help. I also let them know if other teams are working on a similar issue and encourage them to collaborate to find a solution. While I do this largely to encourage independent learning and problem solving skills, as a practical matter, the variety of student projects and the newness of the toolchain essentially guarantees that some students are facing issues I have never seen before, and as the quarter goes on, students are likely to know more about their design than I do.

I use inter-team collaboration and peer-to-peer interactions as a major form of learning in this class. Students are required to make weekly posts to a course forum where they may ask or answer any technical questions that come up. I also dedicate 10-20 minutes per week of class time to an open "debug forum" where students are encouraged to ask the class about any challenges they are facing with their designs. Finally, for any student or group that manages to get a new peripheral, functionality, or IP block working, I offer them extra credit to write a tutorial for how to use the feature. New tutorials are immediately posted to the course website for students to use, and I have included anonymized versions of these tutorials at http://adanowitz.com/home/real-time-embedded-systems. Finally, at least twice during the quarter I required students to present their designs and their current progress to the class so that everyone is aware of what each group is working on and has a chance to ask questions.

As designed, the project framework forces students engage in self-learning, online research, and peer collaboration to manage the complexity of implementing and debugging a heterogeneous system. All the while, the pressure to meet the course deadlines—the course is offered in a ten-week quarter—forces students to strike a careful balance between overall system functionality on one hand and optimization, polish, and performance on the other. I believe that all of these skills help to make for a representative SoC design experience, and transforms students into the type of self-directed, innovative professionals that the field of SoC design requires.

While "representative" this project is also quite demanding on students. The next section discusses the results of the class, including student feedback.

Project Results and Student Feedback

Students in my pilot CPE 439 course created a total of sixteen projects that included a diverse set of designs ranging from an obstacle avoiding autonomous robot, to a Zybo-based oscilloscope. A full listing of the projects is available at http://adanowitz.com/home/real-time-embedded-systems. While the project was successful for many, one team was not able to present a working prototype by the end of the course, and three others were unable to get their entire projects on the Zybo in time, instead relying on laptop-based proofs of concept to complement what they were able to demonstrate on the board.

The group that failed to produce a working demo chose a particularly difficult project: they decided to use the FPGA portion of the Zybo-board to re-implement a variant of the Zilog Z80 processor, and use the ARM core of the Zybo to act as a bootloader and memory controller, loading old proprietary software onto the Zilog. The group did not anticipate how much effort

would be required for this project, and ultimately failed budget enough time. In the future, I plan to take two steps to rectify the problem. First, I will work with students interested in reverse engineering a system to ensure that they understand how difficult such tasks can be. I also plan to instantiate weekly in-class chats with each group to track progress and ensure that groups are sticking to a realistic project schedule.

The other problem came up is that while several groups chose projects involving image and video processing, the process to read in video through the Zybo's HDMI port proved more difficult than first anticipated. While students were able to find various reference designs to help with this problem, many relied on features or IP versions only found in older versions of the toolset (we used Vivado 2014.4), or were not clearly documented. Ultimately, one student was able to get the video-in flow working roughly two-thirds of the way through the quarter and wrote a tutorial for other students to follow. By this time, some of the affected groups had either already changed their project proposal to avoid using video, or, in the case of two groups, were too far behind in their design process to be able to get an entire working system on the board in time for the demo, forcing them to show some algorithmic proofs-of-concepts on their laptops on the last day of class.

The final group that had difficulty getting their design on the board was a special case. This group, made up of Master's students, wanted to do advanced real-time image processing by using High Level Synthesis (HLS) on the OpenCV [11] library to create hardware image processing accelerators. The group wanted to use full Linux rather than FreeRTOS for its OpenCV and USB webcam support. Finally, the group wanted to use the more powerful Xilinx-based Zedboard [12] rather than the Zybo due to the differences in the capabilities of the PSoC between the two boards. While this project seemed incredibly ambitious for a 10-week quarter, I chose to allow the students to go forward since HLS and Linux-based embedded are both relevant and important topics in the realm of modern SoC design. Due to a number of issues including the complexity of bringing up a port of Linux to work with a custom hardware implementation, the students were not able to get a completely working system on the Zedboard, and had to demonstrate many of their algorithms on a laptop computer instead. While these students technically failed to meet all of the project guidelines, I still felt that they succeeded in learning about the complexities of SoCs, and I would certainly approve this project again.

The remaining groups were able to accomplish what they initially proposed, or, despite having to drop some features—the group behind the autonomous obstacle avoiding robot initially planned to create and store a map of the room—still ended up with fully functional projects based on an SoC of their own design. Therefore, I considered the projects and the class to be an overall success.

The students' reaction to the course was primarily measured through my end of quarter teaching evaluations. While the results are summarized below, both the numerical and free response results of my evaluations are available at http://adanowitz.com/home/real-time-embedded-systems. I am providing the full evaluation results to give the reader more context about how the course was received, and to potentially educate the reader about some of the pitfalls and potential mistakes to watch out for the first time a course like this is offered.

My numerical teaching evaluations, included in Table 1 to indicate a high degree of satisfaction with my teaching, which came as a relief since this was the first time I had offered a largely student-driven class. Unfortunately, my department's evaluation form omitted any numerical

questions about how students felt about the course itself, so that must be gleaned directly from the "free response" section of the evaluation.

Table 2. A listing of numerical response evaluation questions and average response. 25 out of 40 enrolled students responded

Question	Average Response (out of 5)
How well prepared does the instructor seem to be in the subject matter?	4.04
Evaluate the instructor on his/her ability to convey subject matter	3.95
Evaluate the instructor on his/her availability and effectiveness during scheduled office hours	4.22
Overall, I would rate this instructor	4.12

The free response questions exposed a range of student reactions covering both the teaching and the course content. Some of the comments were overwhelmingly positive, with one student saying "Awesome project-based class and an instructor who knows his material very well." Other comments were overwhelmingly negative with examples like "this class was not worth it for us to have taken." Other students seemed to have concern with the self-directed learning process:

> A lot of what he seems to be doing is geared at helping students handle their workload and learning process. This might be good for a lower level class but as Seniors I want the professors to understand that I know how to manage my time and learn the material effectively. Especially in an elective where we chose to take the class and want to learn the material. Much more focus needs to be placed on the lectures and the material and a lot less needs to be placed on how students learn. We know how to learn. That is all I have done for my entire life.

A good number of comments addressed my newness as a professor (it was taught at the end of my first year).

Of the comments that addressed the course material, many expressed frustrations with their experience using the toolset and the development board: "Materials and toolchain very hard to use, need more support or documentation." While some amount of frustration is likely inevitable given the challenges of building and programming an SoC, based on my firsthand experience in class, I believe that there were three areas of unnecessary frustration that could be addressed with minor revisions to the course: lack verification and testing experience, handling bugs in the tool-chain, and trouble finding reference designs for our board and tool version.

While students were equipped from previous courses to build test-benches for simple digital blocks in their design, they lacked the skillset to fully test and understand complex IP blocks provided by the Vivado software. As a result, many tended to rely on reference design configurations to try to determine how blocks should be set up, and only tested and debugged their systems at the prototype level. This process often proved difficult and time consuming, since, at the system level, it wasn't always clear which block was causing an error, or what the precise error was. For future versions of the course, I hope to provide additional training to

students in the area of verification. I also plan to impress upon my students the importance of testing components individually before attempting to combine them into a full system. I believe that this course of action will significantly reduce student frustration and debug time.

The second source of frustration is that students encountered bugs with both Vivado and the related SDK, with one extreme comment suggesting that "the equipment and software is in a debugging stage and is not stable enough for people to become familiar enough to do extensive projects." To my knowledge, nothing catastrophic resulted from software bugs, but there were enough small incidents to keep students wary. Frankly, I believe this to be a good learning experience for the students. Vivado is a relatively new toolset under active development—there were 8 substantial updates released between January 2014 and December 2015—and dealing with bugs is often a cost any designer must pay to use the latest tools. Additionally, in my experience all digital design tools from all of vendors have bugs and quirks, and the ability to find workarounds and use the vendor's support mechanisms are a fact of life for many digital designers: Exposing students to bugs in this course just adds to the "representative" nature of the project. In future offerings, however, I plan to address the issue of design tool bugs during the expectation setting phase at the start of class. Additionally, as I continue to offer this course and am exposed to an ever greater number of projects, I believe that I will become better equipped to help students resolve common issues and errors, hopefully alleviating some of the frustration they experience from bugs.

Finally, the lack of widespread documentation and example projects using the board was a concern: "Unfortunately, the Zybo is a relatively new device which makes online research particularly difficult and forces most of the work done by the students to be trial and error. This proved to be particularly frustrating." As the Xilinx Zynq chip and the Zybo board are both relatively new, and since Vivado and its included IP library has been revised several times over the past few years, it was often difficult for students to find compatible reference designs that used all of the peripherals they wished to use. This challenge was made more acute by the fact the Zynq development community is split between those using Vivado and those using Xilinx's legacy ISE tool-chain [13]. I attempted to mitigate the issue during the class by offering extra credit to students who wrote up tutorials for any peripherals they figured out how to use. This policy proved popular with students—"I really hope students make lots of these"—but a common refrain from students was that they wanted tutorials for more peripherals than were on offer: "Need more tutorials." I believe that this issue will only get better with repeated offerings of the course as more students write tutorials for new applications. I have included the tutorials I have collected so far at http://adanowitz.com/home/real-time-embedded-systems, and will upload more as they come in.

An additional, potentially institution specific, concern came from the pacing and level of material covered. CPE439 was open to Electrical Engineering (EE) students, Computer Engineering (CPE) students, upper-level undergraduates, and entry-level Master's students, all with varying degrees of embedded systems, operating systems, digital design, and programming experience. As a result, it was difficult to find a pacing and level of depth that worked for everyone. Despite my best efforts, pacing is still an area that needs work, with student feedback expressing sentiments like "CPEs learn nothing in this class. It is a reiteration of concepts of OS in a simplified way," and "focus a little bit on the basics. A lot of the students came into the class with only the experience in embedded systems and digital design that we've had in [basic pre-requisite embedded systems and digital courses]."

Despite the concerns raised by the students, I believe that my trial run of CPE439 provided a strong introduction to the field of System-on-Chip design, and that it was an overall positive experience for students involved. With increased expectation setting at the beginning of the course, implementing brief weekly chats with each project team, and the increased instructor familiarity with the board and toolset that comes with repeated offerings of the course, I believe that the student experience will only improve.

Conclusions

In my view this course succeeded in giving students a representative SoC experience. It gave students firsthand experience with both the promise of the SoC methodology, and firsthand knowledge of how tough building a heterogeneous system can be.

While authentic, CPE439's two-sided of view of SoC design may have some drawbacks: Given the number of comments expressing frustration over certain aspects of the design, there may be a risk that the course discouraged some students from pursuing further study or employment in SoC work. If that were the case, some students might be better served by a more traditional course offering that focuses more on standardized course learning objectives rather than open-ended student-driven learning. This last point is a serious consideration, and one I intend to study in the future. I believe, however, that as I gain more experience teaching this course and accumulate more sample projects and Zybo tutorials, students' perceived experience will improve. I also believe that students may appreciate CPE439 and its focus on self-directed problem solving once they enter industry or advanced graduate studies.

Bibliography

1. Intel. *Desktop 5th Generation Intel Core Processor Family Datasheet;* Intel, 2015.
2. Munger, B.; Akeson, D.; Arekapudi, S.; Burd, T.; Fair, H. R.; Farrell, J.; Johnson, D.; Krishnan, G.; McIntyre, H. M. E.; Naffziger, S.; Schreiber, R.; Sundaram, S.; White, J.; Wilcox, K. Carrizo: A High Performance, Energy Efficient 28 nm APU. *Solid-State Circuits, IEEE Journal of* **2016,** *51* (1), 105-116.
3. Semiconductor Industry Association. Design. In *The International Technology Roadmap for SemiConductors (ITRS)*, 2011th ed.;, 2011.
4. ARM. AMBA Specification. http://www.arm.com/products/system-ip/amba-specifications.php.
5. Xilinx. Zynq-7000 All Programmable SoC. http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.
6. Altera. SoCs-Overview. https://www.altera.com/products/soc/overview.html.
7. Cypress. Programmable System-on-Chip (PSoC). http://www.cypress.com/products/32-bit-arm-cortex-m-psoc (accessed 2016).
8. Digilent. Zybo Zynq-7000 ARM/FPGA SoC Trainer Board. http://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/ (accessed 2016).
9. Xilinx. Vivado HLx Editions. http://www.xilinx.com/products/design-tools/vivado.html (accessed 2016).
10. FreeRTOS. Quality RTOS & Embedded Software. http://www.freertos.org/ (accessed 2016).
11. OpenCV. OpenCV (Open Source Computer Vision). http://opencv.org/ (accessed 2106).

12. Digilent. ZedBoard Zynq-7000 ARM/FPGA SoC Development Board. http://store.digilentinc.com/zedboard-zynq-7000-arm-fpga-soc-development-board/ (accessed January 2016).
13. Xilinx. ISE Design Suite. (accessed 2016).
14. ABET. Criteria for Accrediting Engineering Programs, 2016-2017. http://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-engineering-programs-2016-2017/ (accessed 2016).