Design and Implementation of a Reconfigurable Computing Course for efficient Hardware/Software Co-Design in Reconfigurable Systems

Daniel Llamocca
Electrical and Computer Engineering
Oakland University

This work describes the implementation of a Reconfigurable Computing course for both senior undergraduate students and graduate students. This class provides students with the theory and techniques to design hardware/software systems that can be reconfigured (usually at running time). Students were evaluated in their ability to successfully partition a system into hardware and software components to implement a reconfigurable system. We elaborate on the course structure, list enhancements for further course implementations, and provide details on the final projects developed by the students.

Results are encouraging: students successfully completed the assignments and final projects, and were highly satisfied with the overall course learning experience. The course material (lecture notes, assignments and tutorials) is freely available online. The main aim of this course (offered as an elective) is to motivate students, foster collaboration, and provide further research opportunities.

Corresponding Author: Daniel Llamocca, llamocca@oakland.edu

Introduction/Background

Electrical and Computer Engineering students usually take standard classes in digital logic and embedded design. Digital logic deals with the digital circuit fundamentals while embedded design deals with software and peripheral control. Reconfigurable Computing expands on material from these two subjects and enables the implementation of reconfigurable systems that alter (often on the fly) the functionalities and interconnection of their components to satisfy time-varying requirements. This emerging area is rapidly finding its way into numerous applications (e.g.: image analysis, video compression, automotive, smart antennas). As depicted in Fig. 1, reconfigurable systems include static and dynamic (alterable on-the-fly) components that are often connected to an embedded microprocessor via an interface; as such, it builds on embedded system design and digital logic design.

Reconfigurable Computing comprises a large and diverse number of topics: computer arithmetic, pipelining, parallelization, embedded interface design, embedded software development, hardware/software co-design, and high-level modeling (e.g. MATLAB®, Octave). Programmable System-on-Chip (SoC) devices are a great platform for teaching Reconfigurable Computing concepts as these devices incorporate a reconfigurable fabric and a processor system [1].

The results of this class implementation are encouraging: improved student engagement, availability of open-source material, and opportunities for research in state-of-the-art topics. Students successfully completed the assignments and final projects, and were highly satisfied with the overall course learning experience.

Similar efforts with reconfigurable devices have been reported: the authors in [2] and [3] describe approaches and experiences when teaching digital signal processing with reconfigurable devices. The combination of digital logic and digital signal processing is a powerful tool that enhances the learning experience as it puts together a number topics and uses different software tools.

This work builds on previous experiences to offer a curriculum for reconfigurable computing. We provide details on the selected hardware/software tools, course structure, and outcomes. We also analyze the practices that support student learning.

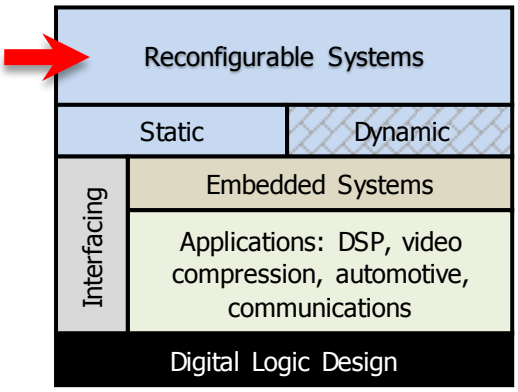
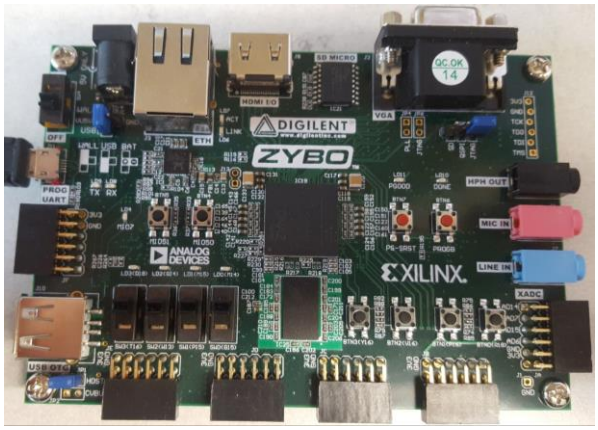
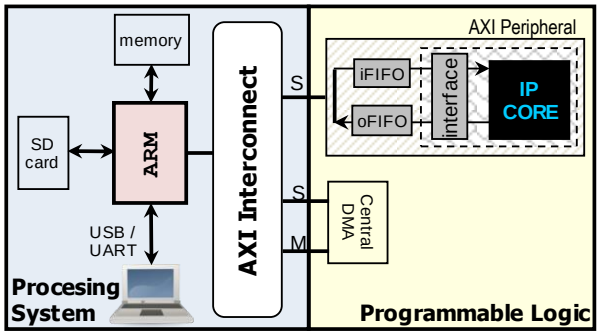


Figure 1. Reconfigurable Computing topics and applications



(a)



(b)

Figure 2. (a) ZYBO Board. (b) Generic embedded system with a custom AXI peripheral. A software routine on the ARM® process provides input data and retrieves output data from the IP core.

Hardware/Software Tools

Programmable System-on-Chip (SoC) devices are very versatile as they include reconfigurable fabric and a processor system that allows the user to design and test embedded systems that contains both software and hardware components. While many common peripherals are available as libraries from the software tool, the user can include custom-built peripherals. The design of these peripherals involves digital hardware and embedded interface design. The Xilinx® Zynq-7000 devices are chosen as they integrate a feature-rich dual-core ARM® Cortex™-A9 processor as well as programmable logic that allows for run-time reconfiguration [1].

In a Zynq-7000 device, the peripherals and the processor are interconnected via the AXI (Advanced eXtensible Interface) bus [1]. The AXI Interface allows for different types: AXI4-Lite (low-speed, simple version), memory-mapped AXI4-Full (high speed), and AXI4-Stream. Fig. 2(a) depicts a block diagram of an embedded system with a custom hardware peripheral.

As for hardware, the ZYBO Board was selected due its low cost and the fact that it is an entry-level embedded software and digital circuit development platform. The ZYBO Board, shown in Fig. 2(b) houses a Zynq-7000 device along with on-board memories, video and audio I/O, dual-role USB, Ethernet, and SD slot.

The course relies on the use of both the AXI4-Lite and the AXI4-Full 32-bit Slave Interfaces. An AXI4-Lite peripheral features a simple register-based interface. An AXI4-Full peripheral allows for burst transfers, thereby requiring a custom interface around the hardware core (or IP). This custom interface includes two FIFOs and state machines [4]. Then, any custom hardware IP would only require interfacing to the FIFOs without considering the AXI transactions. This design strategy is very useful as it allows for re-usability of the hardware IP for other interfaces. Fig. 3 depicts this interface in detail: the 'Input Interface' is made out of registers, while the 'Output Interface' usually contains multiplexers and registers.

Structure of the Course

Here, we detail the organization of the course: topics, assignments, and final projects.

A. Topics

The selected topics as well as their order were carefully selected and grouped into six units. Lecture notes were developed for each unit:

- *Advanced topics in Computer Arithmetic*: Hardware implementation of algorithms requires the specification of the numerical representation as well its numerical bounds and accuracy. We include fixed point arithmetic, floating point arithmetic and the non-standard dual fixed-point arithmetic [5].
- *Specialized arithmetic circuits and techniques*: Here, we deal with hardware implementation of simple and complex common arithmetic units as well as techniques for resource-efficient implementation.
- *Advanced coding in Hardware Description Language (HDL)*: This unit provides details on custom-defined types, parametric coding, and dealing with I/O files for Synthesis and Simulation.
- *Pipelining and unfolding*: These powerful techniques leverage the power of FPGAs by allowing a system to output data at every clock cycle.
- *Embedded system on a System-on-Chip (SoC)*: Here, students learn to partition into hardware and software components. This includes the design of dedicated hardware and high-speed interfaces as well as development of embedded software routines.
- *Dynamic Partial Reconfiguration*: This powerful technology allows hardware portions to be altered (or turned off) on-the-fly, while the rest of the system is still operating [6].

B. Assignments

The evaluations were organized as follows: four homeworks, five laboratories, an in-class midterm exam, and a final project. Students were allowed to form groups of two in the laboratory and final project. Table I lists the topics along with the associated assignments.

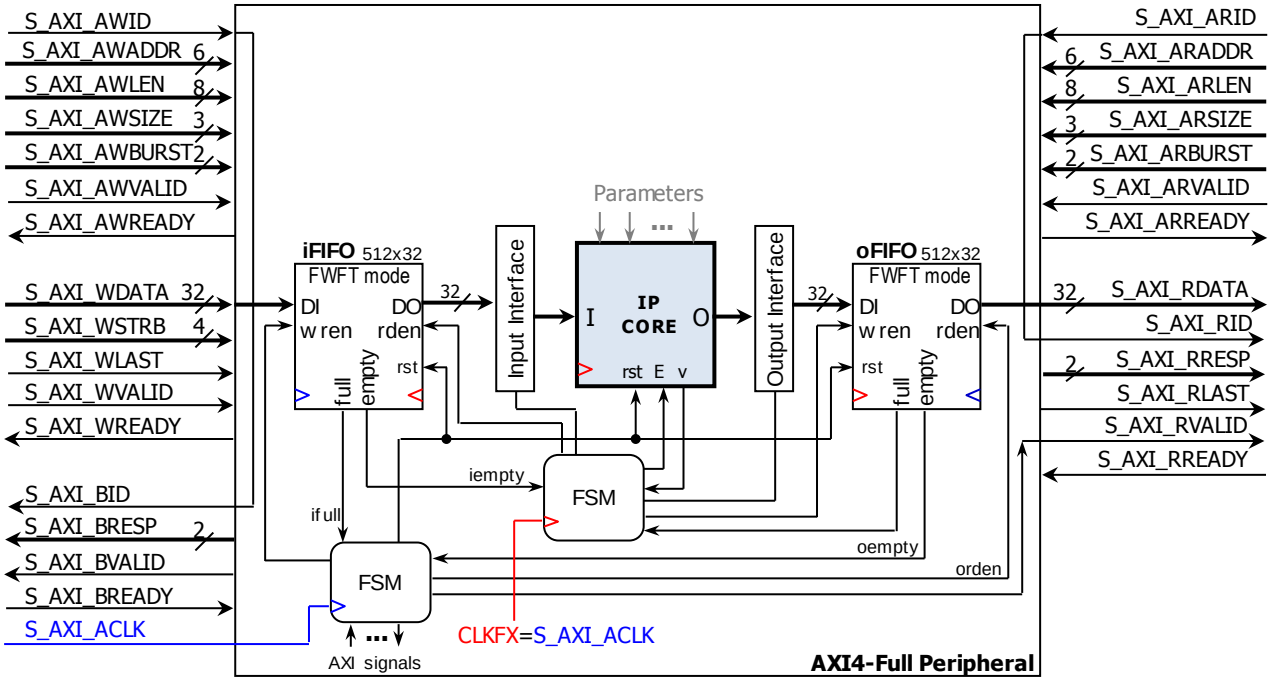


Figure 3. AXI4-Full 32-bit Slave Peripheral. Each FIFO can hold up to 512 32-bit words. And input and an output interface are required as we might need more than 32 bits to process in parallel or output more than 32 bits.

TABLE I. TOPICS AND THEIR ASSOCIATED ASSIGNMENTS: HOMEWORKS (HW), LABORATORIES (LAB)

Topic (# of lectures)	HW	LAB
Advanced topics in Computer Arithmetic (4)	1, 2	
Specialized arithmetic circuits and techniques (4)	2, 3	3
Advanced coding in HDL (2)	4	1, 3
Pipelining and Unfolding (4)	4	
Embedded System on a SoC (6)		2, 4, 5
Dynamic Partial Reconfiguration (3)		

Laboratory assignments included tasks related to the design of embedded systems on a Programmable System-on-Chip (SoC): hardware design, embedded interface design, and embedded software development. The laboratory experiments along with their description are listed in Table II.

TABLE II. LABORATORY EXPERIMENTS

Lab	Description
1	Xilinx® Vivado Webpack Design Flow: Synthesis, Simulation, Bitstream Generation, and Programming
2	Introduction to Embedded System Design: block-based design and embedded software development
3	Hardware implementation of iterative algorithm for computation of trigonometric functions
4	Embedded System: Design of a custom peripheral using the AXI4-Lite Interface.
5	Embedded System: Design of a custom peripheral using the high-speed AXI4-Full Interface.

In the final project, students were evaluated in their ability to successfully partition a system into hardware and software components to implement a reconfigurable system. In addition, students submitted a final report that includes their methodology and results.

C. Online Material

The developed material was organized as follows: i) six units with lecture notes, ii) four homeworks and midterm exam with solutions, iii) five laboratories, and iv) final project guideline along with students' final reports and presentations. The entire material is available online at the Reconfigurable Computing class website.

For the laboratory experiments, we developed step-by-step tutorials on embedded system design with the Vivado™ and SDK software tools. The material, that includes five different units that includes software/hardware examples, is available at the Embedded System Design for Zynq SoC website.

Analysis

A. Practices

The work in [7] provides a comprehensive list of practices that support student learning and teacher effectiveness. In this new course, the following practices were implemented:

- *Knowledge organization*: The class includes a detailed syllabus listing student learning outcomes, grading scheme, schedule of assignments, laboratory materials, and final project guidelines. We also included online material in the form of lecture notes, assignments, and tutorials with project examples.
- *Motivation*: This course featured the design of real-world applications combining both software and hardware components, discussed state-of-the-art topics, and provided research opportunities.

- *Practice*: The class includes homework assignments, student demonstration of working systems in laboratory assignments and final project, oral presentation of the final project, and final report preparation.
- *Group Learning*: The class relies heavily on team-based laboratory assignments and final project.

B. Impact of Learning

We grouped the class objectives into five learning outcomes that include competence in topics, proficiency in embedded system design, and oral/ written presentation. Table III lists the learning outcomes and the associated class activities.

TABLE III. LEARNING OUTCOMES AND ASSOCIATED ACTIVITIES

Student Learning Outcomes	Activities
Design custom architectures using fixed-point, dual fixed-point and floating-point arithmetic	HW1, L1, HW2, L3
Learn advanced coding and testbench techniques in Hardware Description Language	HW3
Describe how to unfold a sequential algorithm to turn it into a fully pipelined architecture	HW4
Design an embedded system using FPGA fabric and an embedded microprocessor	L2, L4, L5
Work in a team environment to design a reconfigurable system and communicate the results in a written report and an oral presentation	Final Project

Two main reasons have been identified that improve student engagement, learning outcomes, and student success:

- Hardware/software co-design: The class brings together topics students have learning throughout college. As such, student engagement improved by using an embedded system platform that allows for integration of dedicated hardware components with embedded software routines.
- Opportunities for research in state-of-the-art topics: The topics dealt with research on cutting edge topics on reconfigurable systems (e.g.: non-standard numerical representations, dynamic partial reconfiguration, AXI embedded interfaces).

Outcomes

A. Student Projects

Here, we list the projects students successfully completed, where they integrated hardware components with software on an embedded microprocessor:

- *Hardware for powering function* (x^y): Based on the expanded hyperbolic CORDIC algorithm [8], a floating point hardware core for powering [9] was developed for both single and double precision. An embedded AXI4-Full interface was built around the hardware cores. Hardware was verified by an embedded software routine reads data from an SD card, streams data to the IP core, retrieves data from the IP core, and writes results back on the SD card.
- *Development of Dual Fixed Point Arithmetic Units*: Hardware architectures for addition, subtraction, multiplication, and division were developed. An AXI4-Lite interface was included for the components. Hardware validation was performed by an embedded software routine in the microprocessor.
- *Hyperbolic CORDIC in Dual Fixed Point Arithmetic*: This architecture, based on the expanded CORDIC [8] illustrates the benefits of using a numerical representation that features high dynamic range and precision without the high resource requirements of floating point arithmetic. Extensive testing was carried out for the functions $atanh, sinh, cosh, exp$.
- *Architectures for floating point units*: Addition, subtraction, and multiplication architectures were developed. An AXI4-Lite interface was included for hardware testing using a software routine.
- *Image filtering and RGB-to-grayscale conversion*: These two hardware components were independently developed and tested using an AXI4-Lite interface and embedded software.
- *Graphics Processing units on an FPGA*: A small GPU architecture was developed that included matrix multiplication, division, and a line drawing algorithm in fixed-point arithmetic. An AXI4-Lite interface was used to configure the hardware, while a Master AXI4-Full interface was used to send video frames to the DDR memory for display.

B. Student Survey

At the end of the semester, students completed an anonymous survey. Out of the 13 registered students, 11 students completed the survey. The following is a selection of the questions asked:

- Q1: The instructor did a good job of making the objectives of the course clear to me.
- Q2: The instructor stimulated and deepened my interest in the subject.
- Q3: The instructor motivated me to do my best work.
- Q4: Value of the laboratory component of the course.
- Q5: Overall rating of this course as a learning experience.

Figure 4 shows how students rated each question (Q1-Q5) and the number of student that gave a specific rating. For this relatively small number of respondents (11), we consider that students rated their experience very highly.

C. Course Implementation Challenges

To be able to design dedicated architectures and then incorporate them into an embedded system, students are required to learn computer arithmetic, specialized arithmetic circuits, and resource-efficient and high performance techniques (e.g.: distributed arithmetic, LUT-based approaches, pipelining, unfolding).

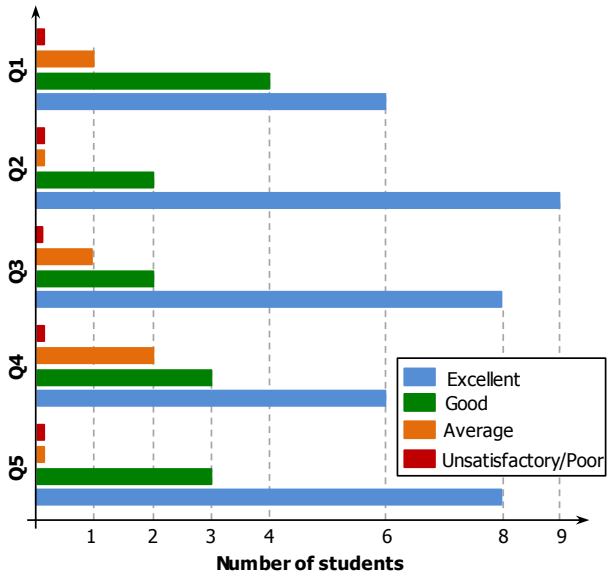


Figure 4. Ratings from eleven (11) students to each of the survey questions (Q1-Q5): Unsatisfactory, Poor, Average, God, Excellent.

However, several students commented that they would have preferred to integrate the microprocessor and hardware earlier in the semester instead of focusing on theoretical topics. This valid point can be addressed by a complete overhaul of the course structure to allow students to start working with embedded systems after the third week of class. The theoretical topics can be introduced along the way.

Also, most students successfully displayed their results using a serial interface (for the laboratory and final project presentations). To improve student engagement, a series of tutorials must be included on how to work with other audio and visual interfaces.

Finally, Dynamic Partial Reconfiguration was presented in class and in the embedded design tutorial, but it was not included in the assignments. The lack of GUI (Graphical User Interface) support in the Vivado™ 2015.3 software tool hindered our efforts [10], and the intricate procedure could only be taught in command-line mode. Extra effort needs to be put on the step-by-step tutorials, and more examples need to be included.

Conclusions

We presented results and analysis from the implementation of a course in reconfigurable computing using hardware/software co-design for embedded systems. The covered material, laboratory experiments, and final projects were successful in terms of student engagement and learning outcomes.

Students rated their overall experience very highly. Based on student feedback and instructor experience, several improvements are listed that will be addressed in future implementations of this course: the freely available class material and embedded design tutorials will be updated on a regular basis.

References

1. Crockett, L.H., Elliot, R.A., Enderwitz, M.A., Stewart, R. W., *The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*, 1st ed. 2014.
2. Hall, T.S., Anderson, D.V., "A Framework for teaching Real-Time Digital Signal Processing with Field Programmable Gate Arrays", *IEEE Transactions on Education*, vol. 48, no. 3, pp. 551-558, Aug. 2005.
3. Meyer-Base, U., Vera, A., Meyer-Base, A., Pattichis, M., "An Undergraduate course and Laboratory in Digital Signal Processing with Field Programmable Gate Arrays", *IEEE Transactions on Education*, vol. 53, no. 4, pp. 638-645, Nov. 2010.
4. Llamocca, D., Aloi, D., "A Reconfigurable Fixed-Point Architecture for Adaptive Beamforming", to appear in *Proceedings of the 23rd Reconfigurable Architectures Workshop (RAW'2016)*, Chicago, IL, May 2016.
5. Ewe, C.T., Cheung, P.Y.K., Constantinides, G.A., "Error modelling of dual fixed-point arithmetic and its applications in field programmable logic", *in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'2005)*, Aug. 2005.
6. Llamocca, D., Pattichis, M., "Dynamic Energy Performance, and Accuracy Optimization and Management using Automatically Generated Constraints for Separable 2-D Filtering for Digital Video Processing", *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 4, Article 4, Jan. 2015.
7. Wieman, C., Gilbert, S., "The Teaching Practices Inventory: A New Tool for Characterizing College and University Teaching in Mathematics and Science", *CBE-Life Sciences Education*, vol. 13, no. 3, pp. 552-569, Fall 2014.
8. Llamocca, D., Agurto, C., "A fixed-point implementation of the expanded hyperbolic CORDIC algorithm", *Latin American Applied Research*, vol. 37, no. 1, pp. 832-91, Jan. 2007.
9. Mack, J., Bellestri, S., Llamocca, D., "Floating Point CORDIC-based Architecture for Powering Computation", in *Proc. of the 10th International Conference on Reconfigurable Computing and FPGAs (ReConFig'2015)*, pp. 1-6, Riviera Maya, Mexico, Dec. 2015.
10. *"Vivado Design Suite User Guide: Partial Reconfiguration (UG909)"*, v2015.4, Xilinx Inc., Nov. 2015.