Teaching Hybrid HW/SW Embedded System Design Using FPGA-Based Devices

Alfonso Rodríguez, Jorge Portilla, Eduardo de la Torre, and Teresa Riesgo
Centro de Electrónica Industrial
Universidad Politécnica de Madrid
Madrid, Spain
{alfonso.rodriguezm, jorge.portilla, eduardo.delatorre, teresa.riesgo}@upm.es

***Abstract*—Complex computing platforms involving pipelined processors, memory hierarchies, multi-core and many-core architectures are very common nowadays. These approaches require a deep understanding of the underlying hardware and the corresponding programing model to be able to decide which alternative is more suitable, i.e. obtain the best performance at the minimum cost, for a given application. Hence, it is important to cover all these aspects in academic curricula in order to provide engineers with competitive advantages and increase industrial productivity.**

In this paper, the methodology followed in a subject on Advanced Processing Architectures from a MSc program is presented. The theoretical content is complemented using hands-on exercises to further analyze the concepts discussed in class. As an example, the practical lessons on single-core Systems on Programmable Chip are reviewed in detail, showing the key ideas that are to be acquired by the students enrolled in the subject. Moreover, the proposed strategy has been evaluated using a voluntary and anonymous questionnaire to detect the strong and weak points of the proposed approach. This feedback is essential to provide students with valuable knowledge and meet quality criteria in academic education.

***Index Terms*—Education, Embedded Systems, FPGAs, SoPC, Industrial Electronics.**

I. Introduction

The use of FPGAs (Field Programmable Gate Arrays) in basic digital electronics courses is becoming widespread due to the flexibility that these devices provide. Students are now able to implement almost any digital circuit using either schematics or HDL-based descriptions of the functionality that is to be generated. However, in more advanced courses (for instance in MSc of PhD programs), the applicability of FPGAs can be extended to design embedded systems, which have added complexity and can introduce design tradeoffs (performance vs. cost, for instance). This approach is known as System on Programmable Chip (SoPC), and it has been active for more than a decade [1]. In fact, the use of SoPC-based learning courses is not new, and some examples can be found throughout the literature. These examples include processor microarchitecture assessment [2], embedded algorithm programming [3], robotics projects [4], real-time system design [5], or custom hardware accelerator designs [6] among others.

In this work, a practical approach for students to understand the foundations of complex computing platforms using SoPCs on FPGAs is presented. The use of high-level abstractions to design single-core architectures with pipelined structures, caches between the external RAM memory and the processors or even hardware accelerators to offload certain computations, allows students from different backgrounds (electronic engineering, computer science, etc.) to successfully design, program and test embedded systems.

The rest of this paper is organized as follows. Section II presents the organization of a course on Advanced Processing Architectures, and its context within a MSc program focused on applied research. Detailed information on some of the practical lessons of the course is presented in Section III. The methodology used is also presented in Section III, and later evaluated in Section IV. Section V shows some conclusions and presents the future work.

II. Course Organization

The Centro de Electrónica Industrial of the Universidad Politécnica de Madrid (CEI-UPM) is a public research center whose activity is mainly focused on the fields of power electronics and embedded systems. In order to boost all the efforts around these topics, the center also offers graduate programs at both MSc and PhD levels. Although the PhD program is clearly oriented towards academic research, the Master on Industrial Electronics has a strong component of applied research, fostered by continuous interaction with leading companies in the field of industrial electronics. As a result, the curriculum of the Master on Industrial Electronics covers a wide range of topics, both theoretical and practical, around the areas of expertise of the center.

Students enrolling in the MSc program should choose a minimum of 36 ECTS from a pool of subjects, attend three specialized seminars (3 ECTS each), and conduct a research work to present a Master Thesis (this is worth 15 ECTS). Hence, and assuming that students do not need to undergo additional lessons to complement their background knowledge, the total amount of work in the MSc program adds up to 60 ECTS (the curriculum is shown in Table I). The pool of subjects provides enough flexibility to allow candidates to customize their profile, focusing their effort on either power electronics, embedded systems or a combination of both.

TABLE I
MASTER ON INDUSTRIAL ELECTRONICS (CEI-UPM): CURRICULUM

Profile	Subject	ECTS
Power	Modeling of Devices and Electronic Systems	4.5
	Electromagnetic Compatibility	4.5
	Design of Magnetic Components	3
	Modeling and Control of DC-DC Converters	4.5
	Advanced Power Supply Systems	4.5
	Industrial Applications of Power Electronics	4.5
	Digital Control of Power Converters	3
	Power Factor Correction	3
Embedded	DSPs: Methods and Tools	4.5
	Wireless Sensor Networks	3
	Languages and Tools for Electronic System Design	4.5
	Advanced Processing Architectures	**4.5**
	Advanced Methodologies of Digital System Design	4.5
	Communication Networks	4.5
Common	Specialized Seminars	9
	Master Thesis	15

This paper is focused on the Advanced Processing Architectures subject (from the embedded systems profile), which covers the following topics:

- Advanced single-core architectures.
- Systems on Programmable Chip.
- Multi-core architectures.
- Many-core architectures.

In the *advanced single-core architectures* section, the foundations for the rest of the theoretical content are presented. First, the basic single-core microprocessor architecture, which is considered the starting point for all the later discussions, is reviewed. The analysis is carried out paying special attention to performance vs. cost tradeoffs. Students learn common performance metrics, e.g. number of Cycles Per Instruction (CPI), Instructions Per Second (IPS), or Floating-Point Operations Per Second (FLOPS), as well as the influence of the instruction set (machine type, addressing modes, etc.) on these metrics. The well-known Amdahl's law [7], which can be used to obtain rough estimates of the improvements that might be achieved by modifying a given architecture, is also presented in this section. Once this knowledge has been acquired, architectural enhancements on top of the basic single-core microprocessor architecture are discussed: pipelining and memory hierarchy. Pipelined architectures and their associated problems (structural, data and control hazards) are deeply analyzed, providing examples in assembler code and proposals to overcome these limitations. The rationale behind having multi-level memory hierarchies, with intermediate caches that take advantage of locality principles (temporal and spatial localities are covered) to reduce execution time is presented, and the discussion is further extended by analyzing the associativity level in the caches (direct mapping vs. n-way associative).

In the *Systems on Programmable Chip* section, students are encouraged to learn how to build complex systems in which certain functions are not executed on the main processor but on dedicated, application-specific hardware accelerators. Although the approach is still based on software applications running in one single processor, systems are assumed to be larger in size and number of elements: microprocessor, on-chip memories, ADCs, DACs, or even serial communication interfaces (SPI, I^{2}C, UART, etc.) are all within a single device and thus, require interconnections to exchange information among them. To face this type of systems, the students learn high-level design techniques such as platform-based design or High-Level Synthesis, which are able to hide complexity behind high-level abstractions that ease the design process. Moreover, students also learn the differences between point to point, bus-based and NoC-based (Network on Chip) communication approaches.

In the *multi-core architectures* section, Flynn's taxonomy [8] is introduced to analyze the different types of computer execution (mainly SISD, SIMD, and MIMD) and, as a consequence, the different levels of parallelism that exist in today's applications (data-level parallelism, task-level parallelism, etc.). Systems with multiple cores are analyzed, showing the pros and cons of using shared, distributed or hybrid memory architectures, together with several interprocess communication techniques (threads vs. message passing) and the synchronization mechanisms available (barriers, semaphores, etc.).

In the *many-core architectures* section, students learn the approach followed in GPUs from the hardware basics (huge amount of parallelism using small, simpler processing cores) to the high-level programming paradigms that are most commonly used (CUDA and OpenCL). The focus of this section is put on understanding how massive data-level parallelism is deployed on top of GPU architectures.

These theoretical concepts are further extended using a hands-on approach in the practical lessons, which are mandatory. Students exercise the acquired knowledge using single-core architectures with configurable custom microprocessors and cached-enabled memory hierarchies, single-core SoPC designs (hardware development, software programming, hardware acceleration, and HW/SW co-debugging are covered), multi-core SoPC designs based on shared memory architectures (highlighting different interprocess communication and synchronization mechanisms), and OpenCL-based GPU programming to efficiently take advantage of massive data-level parallelism. This paper shows, with far more detail, the specific contents of the first two practical topics: single-core custom microprocessors with memory hierarchies and single-core SoPC designs.

III. PRACTICAL LESSONS ON SOPC

As it has been already introduced in the previous section, the aim of the practical lessons in Advanced Processing Architectures is to complement the theoretical knowledge with simple, specific examples that highlight only those aspects that are considered relevant. Lab sessions targeting SoPC design are built around Xilinx devices and tools. The ZYBO development board (Fig. 1) is used as the base platform for all designs, since it features a Zynq device. Xilinx Zynq-7000 devices combine, in a single chip, a dual-core ARM-

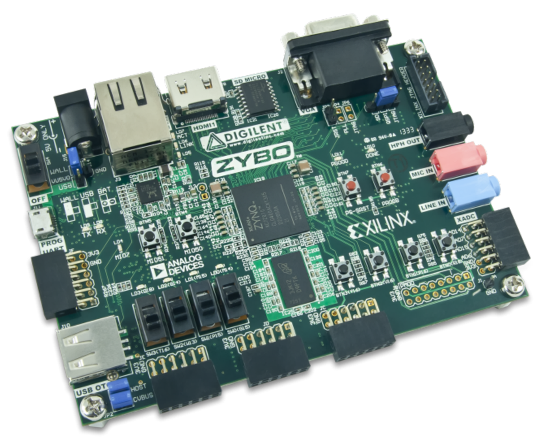

Fig. 1. ZYBO development board used in the practical lessons on SoPC.

based CPU, several peripherals with common functionalities (SD card controller, UART, SPI, Ethernet MAC, etc.), and an FPGA matrix, which makes them more than suitable to cover three out of the four topics of the subject (being many-core architectures the odd one out). In addition, all designs have been developed using Vivado (and Vivado HLS when required).

It is important to highlight that the content and the equipment used in the practical lessons has been continuously updated due to the fast changes in certain technologies (FPGAs mostly). The current approach has been active for two academic years, and up to that point the lessons were based on Virtex-5 devices (XUPV5-LX110T development boards) and Xilinx EDK tools.

The practical activity around single-core SoPCs is divided in four lessons (L) and a homework (H), and organized as follows:

- Embedded hardware lab [L].
- Custom microprocessors and memory hierarchy [H].
- Embedded software lab [L].
- Custom hardware accelerator design lab [L].
- Hardware/Software co-debugging lab [L].

In the *embedded hardware lab*, students learn how to use the Vivado design suite to build basic SoPCs using the platform-based approach provided by Vivado IP Integrator. The lesson consists of two different exercises. On the one hand, a system with one ARM processor (hard core) and a GPIO peripheral (implemented as a soft core inside the FPGA matrix) attached to four switches and four LEDs is built and programed to light the LEDs whenever a switch is active. Note that the program is very simple to keep the focus on the hardware part of the system. On the other hand, students implement a system completely inside the FPGA using a MicroBlaze (soft core), a timer (to measure performance), and a memory controller. This second setup is used as the baseline for the *custom microprocessors and memory hierarchy* homework, in which students have to implement a matrix multiplication algorithm and measure its execution performance when modifying the caches (sizes, line lengths) or when incrementing the instruction set of the microprocessor (for instance, to add a dedicated hardware multiplier able to speed up computations). Students are also required to write a report with graphic representations of the obtained results (as shown in Fig. 2), and encouraged to propose algorithmic modifications to increase the efficiency of cache-based computations.

In the *embedded software lab*, the focus is moved from the hardware onto the software component of a SoPC. Students learn how to set up complex interrupt schemes using peripherals in both the Processing System (PS) and the Programmable Logic (PL) of the Zynq device, and adopting a hierarchical approach using interrupt controllers to manage different interrupt sources. The application developed in this lesson is still simple (making LEDs blink and change mode whenever a

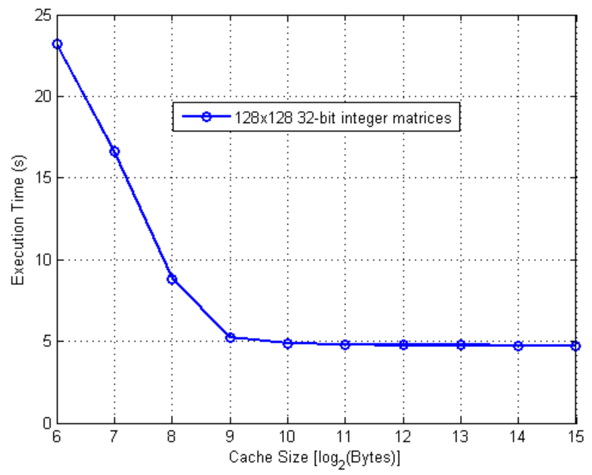

Fig. 2. Results obtained in the matrix multiplication algorithm running in a MicroBlaze without using hardware multipliers (DSP resources) and with a fixed cache line of 4 bytes when changing the size of the caches.

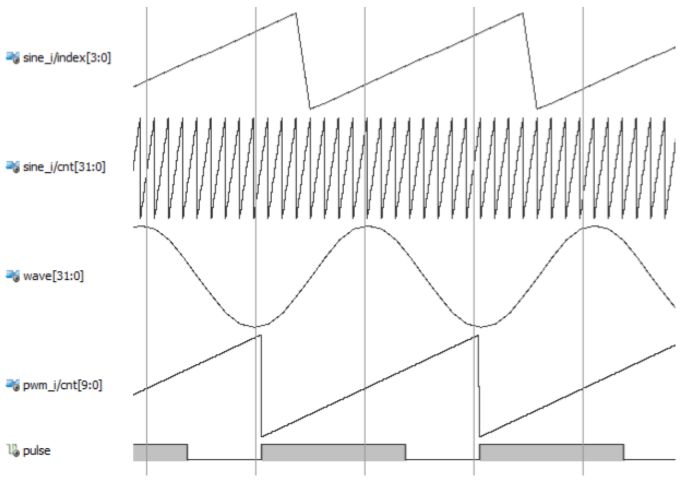

Fig. 3. Results obtained during a hardware/software co-debugging session. The custom peripheral implements a PWM generator and a configurable sine waveform generator.

button is pressed, for instance), because the importance resides in learning how to use the predefined drivers (provided by the vendor) to interface user applications written in C/C++ with the underlying hardware (already built using the concepts from the first practical lesson).

In the *custom hardware accelerator design lab*, students learn how to generate application-specific hardware accelerators able to speed up a given algorithm, and how to develop their own peripherals (for instance, the PWM controllers used in digital control of power converters). The first part is carried out using, again, the matrix multiplication algorithm (which is commonly used as the ”Hello World!” application in parallel computing, and acts as a reference design for comparisons and further tests in the many-core practical lessons) and Vivado HLS, and aims at making students understand that low-level knowledge of the underlying hardware is required whenever the performance of a given algorithm is to be improved when performing a migration from software to hardware. Students are shown an HLS design flow based on optimization directives oriented to first optimize the datapath (loop unrolling, loop pipelining) and then optimize data accesses (array partitions), while at the same time keeping a fixed interface (AXI4-Stream, also enforced by HLS directives). This accelerator is put to the test comparing its performance with the equivalent algorithm running in software, which is outperformed even when using cache-based computations. The second part involves HDL coding of a simple peripheral and its integration in one of the wrappers provided by the vendor.

In the *hardware/software co-debugging lab*, students learn how to debug an embedded software application using breakpoints and watchpoints (they are also told the differences between both elements), how to monitor a hardware design inside an FPGA using Integrated Logic Analyzers (ILAs), and how to combine both approaches to debug any hardware-accelerated application. Some results can be seen in Fig. 3, where a custom-made peripheral to generate sine waveforms and PWM pulses is monitored at run-time.

The methodology used in all these practical lessons is the same. At the very beginning of the class, a brief introduction of the lesson is performed using some slides, which also contain a summary of what has been done so far in previous SoPC lessons. After the introductory speech, students are guided through the scheduled tasks with continuous interaction with the professor to clarify doubts, analyze problems, and propose solutions. At the end of the lessons, a questions and answer turn is opened to cover all loose ends that might still exist with regard to the content.

IV. Results

To evaluate the effectiveness of the approach used in the single-core SoPC practical lessons, students were given a questionnaire after all the lessons. The contents of the evaluation form are listed below:

1) If you had to write the previous skills required to undergo these lessons, i.e. something that you definitely have to know beforehand, which ones would you highlight? (If more than one option, please provide a list with priority order)
2) Did you have previous experience on digital system design? (If yes, please provide some information and/or details)
3) Did you have previous experience on C/C++ programming? (If yes, please provide some information and/or details)
4) Did you have previous experience on embedded system design? (If yes, please provide some information and/or details)
5) Do you consider that the level of abstraction used during the lessons is enough to hide low-level design details? (If not, please write down what you think is missing)
6) Evaluate the knowledge you have acquired during the lessons (previous vs. current), pointing out the thing (or things, if you wish) you consider the most important/relevant
7) Evaluate the methodology followed during the lessons, providing strong and weak points. Key aspects in which you should focus (but not restrict to) are:
    a) Initial explanation: Would you increase/reduce/modify it? How important do you think it is that first part to understand the rest of the things we do in class?
    b) Previously on APA: How important is, in your opinion, this section? Does it help you clarify what has been done and what is to be done?
    c) Use of slides: Is it necessary? Would you change/remove them?
    d) Source code: Should it be provided at the beginning of the lesson? Would it help you improve your understanding or your skills if you had to manually write everything down from scratch?
    e) Interaction with students: How important is it for you? Do you consider that it can further increase the understanding of the class? How would you foster this interaction to make it attractive and not something negative?
8) Rank the lessons (descending order) according to how interesting you found them (add comments if needed)
9) Rank the lessons (descending order) according to how much you learned in them (add comments if needed)
10) How relevant do you think these lessons are regarding your career? Do you see yourself using anything we have seen so far in the future?
11) Propose a real-world use case in which the acquired knowledge can be applied (please, try to first state a problem and then your proposed solution)
12) Additional comments/suggestions/complaints (anything that has not appeared yet in this questionnaire, but you think is important)

From a total amount of 11 students enrolled, 7 of them handed out their responses (filling in the questionnaire was absolutely voluntary and anonymous). Moreover, as students came from very different backgrounds (foreigners under the Erasmus program, computer scientists, electronic engineers, etc.), some of the answers were significantly different from the others. For instance, some of the students had previous experience on digital electronics and FPGAs whereas others had never worked with hardware designs during their former studies. However, all of them were already familiar with C/C++ programming languages to some extent and, as it can be inferred from their responses, the abstraction provided by the high-level approach (HDL-based module description is only used in one of the lessons) made it possible for them to successfully follow the classes. In fact, the best ranked lesson was the one on custom hardware accelerator design (mainly due to the novelty of HLS methodologies using commercial tools).

Regarding the methodology, all of them agreed on the importance of the initial explanation, specially when backed up with slides to help the understanding of what to do next. Moreover, positive comments were received about summarizing the content of previous lessons at the beginning of the class. In addition, positive feedback was also obtained regarding the interaction between professor and students, highlighting the importance of active discussions to ease understanding of key concepts.

However, and due to the complexity of learning too many concepts in only four lessons, most students did not find any real-world use case in which the acquired knowledge could have proven useful. Comments regarding the excessive complexity of some lessons (HLS optimization techniques were only reviewed but not deeply discussed, for instance) were also found, and will be taken into account for the next academic years.

V. Conclusions and Future Work

In this paper, the methodology used to teach Advanced Processing Architectures has been presented and discussed, putting special emphasis on the practical content regarding single-core SoPC designs. The results obtained from the final questionnaire used to evaluate the subject confirm the validity of the approach. Students find it really useful to complement the theoretical concepts with hands-on exercises that, although very specific (only some key ideas are implemented), are enough to provide a real-world overview of the different problems at hand.

In the academic years to come, the curriculum of the MSc program is expected to be slightly modified to host two types of students: those willing to start a research career (research profile) and those willing to increase their knowledge and apply it in industrial scenarios (professional profile). As a result, the contents of the subject will be extended to cover the already present topics with far more detail and also to introduce new key elements.

In addition, standardization activities, together with the rise of new high-level design environments, encourage the continuous evolution of the practical lessons. Hence, vendor-specific tools such as SDSoC [9] or SDAccel [10], both from Xilinx, are expected to be included as part of the curriculum for the next academic year. SDSoC, able to efficiently map software applications written in C/C++ into hardware accelerators, can be used to emphasize the importance of custom hardware accelerator design from high-level language descriptions. SDAccel, on the other hand, able to efficiently deploy massive data-level parallelism using OpenCL in FPGAs, can be used to complement the lessons on many-core architectures, HLS and hardware acceleration.

Acknowledgments

The authors would like to thank the Spanish Ministry of Education, Culture and Sport for its support under the FPU grant program.

References

[1] J. O. Hamblen and T. S. Hall, "Using System-on-a-Programmable-Chip Technology to Design Embedded Systems," *International Journal of Computer Applications (IJCA)*, vol. 13, no. 3, pp. 1–11, Sep 2006.

[2] M. Nakkar, "Design based tutorials for system-on-chip teachings," in *Engineering Education (ICEED), 2009 International Conference on*, Dec 2009, pp. 117–121.

[3] A. A. C. Atoche, J. V. Castillo, J. S. O. Aguilar, and C. R. Cruz, "Laboratory projects for engineering students with fpga," *IEEE Latin America Transactions*, vol. 6, no. 2, pp. 130–136, June 2008.

[4] T. S. Hall and J. O. Hamblen, "System-on-a-programmable-chip development platforms in the classroom," *IEEE Transactions on Education*, vol. 47, no. 4, pp. 502–507, Nov 2004.

[5] A. Kumar, S. Fernando, and R. C. Panicker, "Project-based learning in embedded systems education using an fpga platform," *IEEE Transactions on Education*, vol. 56, no. 4, pp. 407–415, Nov 2013.

[6] R. B. Foist, C. S. Grecu, A. Ivanov, and R. F. B. Turner, "An fpga design project: Creating a powerpc subsystem plus user logic," *IEEE Transactions on Education*, vol. 51, no. 3, pp. 312–318, Aug 2008.

[7] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485.

[8] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, vol. C-21, no. 9, pp. 948–960, Sept 1972.

[9] Xilinx, "SDSoC Environment User Guide (UG1027)," May 2016.

[10] ——, "SDAccel Development Environment User Guide (UG1023)," Feb 2016.