Teaching HW/SW codesign with a Zynq ARM/FPGA SoC

Balasch, Josep; Beckers, Arthur; Bozilov, Dusan; Sinha Roy, Sujoy; Turan, Furkan; Verbauwhede, Ingrid

DOI:
10.1109/EWME.2018.8629481

Teaching HW/SW codesign with a Zynq ARM/FPGA SoC

Josep Balasch, Arthur Beckers, Dušan Božilov, Sujoy Sinha Roy, Furkan Turan, Ingrid Verbauwhede
KU Leuven COSIC
Kasteelpark Arenberg 10 bus 2452
B-3001 Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

Abstract—**In this paper we describe a lab session-based hardware/software (HW/SW) codesign course for implementing embedded systems. The goals of the course are to teach the fundamental concepts of embedded system design, develop hands-on HW/SW codesign skills, and to show that there are many possible ways to explore the design space. The reason behind choosing HW/SW codesign approach is that it brings the best of the two worlds: the flexibility of SW and the power/energy/computation efficiency of HW. As an example project, students codesign the well-known RSA public-key cryptosystem in the Xilinx Zybo boards that contain a Xilinx 7-series FPGA coupled with an embedded ARM processing unit. Students are required to explore the design space, weigh the various alternatives and take design decisions. Besides, the project cultivates non-technical skills such as team building and management, sharing of work-load, decision making, presentation and technical report writing.**

I. Introduction

The *Design of Digital Platforms* course is a compulsory module in the Master of Science in Electrical Engineering programme taught at the Department of Electrical Engineering (ESAT) from KU Leuven, Belgium. The course takes place during the first semester and has a workload of 6 ECTS. Its aim is to give students insight into what a digital platform is through an overview of the different design steps and important design decisions in the development of a digital platform. It serves as an introduction to the topic of *hardware/software* (HW/SW) codesign [1].

The course is split into lectures (2.41 ECTS) and hands-on sessions (3.59 ECTS). The former give students a theoretical base to the design of digital integrated circuits. Topics covered include, among others: abstraction levels, design goal trade-offs (area, throughput, delay, power, energy, flexibility), control/data flow graphs or gate level design for low power and for low energy. The latter consists of a series of project-oriented sessions where students develop a digital platform that implements a public-key (PK) cryptographic algorithm. In recent editions of the course, the final assignment is to implement encryption/decryption based on the RSA cryptosystem [2]. However, earlier editions targeted alternative constructions such as Elliptic Curve Cryptography (ECC) [3], [4].

The goal of the project sessions is to experience first-hand the many options in the HW/SW codesign space by allowing students to explore different design strategies and implementation trade-offs. In this context, the selection of a PK cryptographic application is a suitable target. Its modular nature enables to arrange assignments that gradually build on top of each other. Moreover, the complexity of certain underlying operations stimulates the search of solutions in the HW/SW design space. In the first half of the sessions, students develop all necessary arithmetic blocks both in SW and HW. In the second half of the sessions, they combine the blocks into a HW/SW codesign according to their own design strategy. Grading of the project involves evaluating how the implementation results (in terms of silicon-area, speed and flexibility) match the original design goals of the students. Mapping functionalities to the SW side generally results in slower but smaller designs, while leveraging to the HW side yields faster but larger solutions. Flexibility can be incorporated by enabling extra functionalities, for instance, generic interfaces that allow re-using (parts of) the HW blocks for other purposes or capabilities for the system to be upgraded if required (e.g. to larger key lengths or to implementations with built-in resistance against physical attacks).

In the first editions of the course, the platform used in the project sessions was a resource-constrained 8-bit micro-controller (8051-based) connected to an FPGA module (Xilinx Virtex-4). Software development was done in C and assembly, while both hardware development and co-design simulation were done using the GEZEL environment [5]. However, in 2014 a more modern target platform was introduced in the project: the Zybo Zynq-7000 ARM/FPGA System-on-chip (SoC) [6]. Equipped with a dual-core 32-bit ARM Cortex-A9 processor interfaced with a Xilinx 7-series FPGA, the architecture of this platform offers native support for the development of codesign projects.

The aim of this paper is to introduce the current format of the course project after migration to the Zybo platform. For more details about earlier editions of the course using the 8051+FPGA platform, we refer the reader to [7].

II. Project Sessions

Around 50 students take part in the course. Of these, roughly half come from the Bachelor of Engineering taught at KU Leuven. The rest are international students from various countries. This diversity has a positive impact on the teaching, as the project assignment is carried out in teams of three. It allows students to work in an international environment

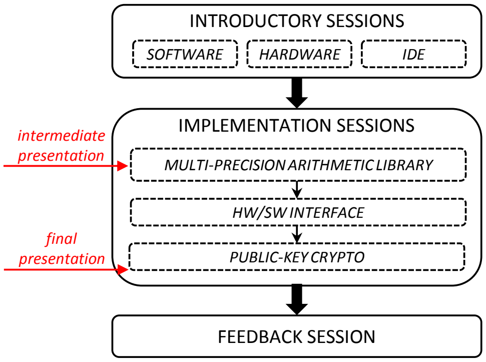

Fig. 1. Project schedule.

with people from different backgrounds, a useful experience towards their future careers. On the other hand, such diversity demands to incorporate several introductory sessions to an already demanding schedule in order to bring all students to a common technical base.

The complete project consists of 16 sessions of 2.5 hours each. A high-level view of the schedule is shown in Figure 1. A description of each block follows.

A. Introductory Sessions

The goal of these sessions is to provide the technical background necessary to enable the development of the project. This includes the fundamentals of our selected programming languages for SW and HW, as well as the inner workings of the used Integrated Development Environment (IDE)

1) Software: students are given an introduction to the *C* programming fundamentals with particular emphasis on data types and usage of pointers. To this end, they are given a few warm-up tasks to be completed on the lab PCs. Next, we move the target platform to the Zybo boards. Due to the lack of simulation support for this platform, we provide students with a board and a project template that demonstrates how to interface over serial port for stdout, e.g. using the dedicated `xil_printf()` function. We also show how to debug code (line-by-line or setting breakpoints), and how to set watch to variables. Functions to read cycle-counter registers of the platform are also provided, so that students can measure execution time and profile the performance of their code. Lastly, we introduce the basics of big-number representation necessary to build up a multi-precision arithmetic library.

2) Hardware: we begin by describing logic elements for combinatorial circuits, memory elements and clocking, respectively, for sequential and synchronous design. We also introduce the fundamentals of a hardware definition language (HDL). For this project, we select *Verilog* as the target HDL due to its syntax similarities to C. Along with the basics of *Verilog*, we also explain datapaths and finite-state machines (FSM) while highlighting good coding practices, e.g. separation of control and datapath in the code. Similarly to the SW sessions, we provide students with a preconfigured project and let them start with a warm-up task. This task involves several logic circuits and demands students to identify various design types and memory elements. Next, we introduce a traffic light controller task by giving students an exemplary datapath and FSM from which they can built upon in the next phases of the project.

3) IDE: throughout the project we use the tools provided by Xilinx for development with Zynq SoC. SW development is done in XSDK, while HW development is done in Vivado. Our department has licenses for these tools. They are installed in the classroom PCs and even offer remote desktop access. However, as our project does not make use of advanced features, it is possible for students to install local copies using free licenses. Students are given a custom project package consisting of fundamental code files (in *C*, *ARM ASM* and *Verilog*), a TCL script and a *Makefile*. The *make* targets (i.e. create, open, clean) make use of the TCL script to construct both Vivado and XSDK projects from scratch. This method is useful for reducing the project package size to few kilobytes, and giving us full control of the project configuration. It allows us to change the severity of some important warnings to an error so that the compilation flow is interrupted and the students are forced to address the issues presented by these warnings. Our experiences collected in previous years show us that students often miss paying attention to synthesis and implementation results, which may hide very important problems such as latches or multi-driven nets. In such cases, even though the design works on simulation, it often fails when programmed to the device. Therefore, we use the TCL scripts to adjust the severity levels. Demonstration on how to use the IDE is done through video tutorials prepared with screen recordings showing the following steps: opening a HW design in Vivado, simulation, synthesis and implementation, exporting the bitstream to XSDK, compiling SW, programming the device and interfacing with it.

B. Implementation Sessions

The implementation sessions represent the bulk of the project. In a first phase, students develop the necessary building blocks to enable arithmetic computations (both in SW and in HW). In a second phase, they tackle the final codesign assignment by combining their blocks according to their own design strategy.

1) Multi-precision Arithmetic Libraries: starting with basic multi-precision arithmetic functionalities, e.g. addition and subtraction, students are asked to gradually build on them to enable modular arithmetic operations. The target operation at this level is modular multiplication. To debug and/or verify the correctness of all implementations, we provide a *python* script that generates random test vectors of all target arithmetic functionalities.

In the SW sessions, students are given pseudo codes from Chapter 14 of [8] to implement basic arithmetic. For modular multiplication, we opt to use the efficient algorithms due to Montgomery [9]. More specifically, students are asked to implement one of the algorithms in [10] offering different memory/computation trade-offs. Lastly, students are motivated to improve their code using assembly-instruction level optimizations.

In the HW sessions, students start similarly with basic arithmetic and later target a multiplication module build on top of it. For these assignments, we set a minimum clock frequency goal that prevents implementing a single cycle adder. Therefore, smaller word sizes are used to execute the addition in multiple cycles. We motivate the students to find the optimum word size which meets with the timing goal, while minimizing the number of execution cycles. In the following sessions, the task is changed to implement the Montgomery multiplication following single-bit scanning. The HW tasks are introduced with *Verilog* template files defining the modules with preferred IO behavior, which enforces the implementation of start and done signals. Each module is associated with a ready-made testbench file which follows the same IO behavior for simulation.

2) HW/SW Interface: making the HW in FPGA accessible by the SW running on the ARM cores, requires to build a suitable interface. The Zynq platform supports only AXI based interfacing, a sophisticated interface that is hard to get across entirely. Therefore, we provide a design that introduces a simplified interface at both SW and HW endpoints, while uses AXI at the background to transfer data and command between them. This simplified interface enables two methods of communication. The first method makes a set of 32-bit registers available to SW with port-mapping for sending commands to HW and reading its status. The second enables bulk data transfers with DMA between the DDR system memory and block RAM respectively accessible to SW and HW. Once the interface to HW/SW is made available, students find means to compare the performance of modular arithmetic functions in SW and HW. On the Zynq device, optimizing a *C* code with Assembly do not always offers great returns without cleverly handcrafted techniques, since the CPU architecture and compiler are already advanced. In comparison, execution on HW is naively faster including the overhead of interfacing; however, demands significant amount of work for such gains.

3) PK Crypto: the last development task of the project focuses on a PK crypto application. As students may be inexperienced with cryptography, we first describe the basics of cryptography at a very high level by exemplifying their use in our daily life. Later, we describe the target application which encrypts and decrypts a message by calculating modular exponentiation of it with either the encryption or decryption exponent. For that purpose, Montgomery modular exponentiation [8] is introduced and students are asked to implement it using their Montgomery multipliers. While implementing the exponentiation, students prefer either managing the underlying multiplications with controlling the HW multiplier in SW, or implement the whole exponentiation scheme in the HW. The former is obviously easier to implement, favour flexibility and lower area utilization. The latter cuts-off the communication overhead and yields to a performance optimized design. The RSA algorithm favours a small exponent for encryption but a large one for decryption as it requires better protection. Therefore, executing the decryption requires more multiplications, making the performance differences become clear in the comparison of various implementations.

C. Presentation Sessions

We expect students to explain their design and optimization twice during the semester. The first is the intermediate presentation held after they complete library implementations in SW. The second is the final presentation which centers upon the HW/SW codesign. Both are five minute presentations followed by questions. To urge students evaluate each other, we provide a list nominating each group to another in a wrap-around fashion. In addition to presentations, a report is expected which discusses, among other, design decisions, handling of trade-offs, implementation results or debug/test strategies.

D. Feedback Session

The goal of this recently introduced session is to provide a clean end of the project. Its main purpose is to wrap-up the the project by presenting students a summary of the results of all teams. We highlight the features and/or optimizations of the best designs, emphasize on the good coding/debug practices taught during the course, and discuss other relevant design criteria (e.g. low power or low energy) which are not in the scope of the assignment. Additionally, it is also a opportunity for students to provide feedback on the course so that further editions can be improved.

III. Project Outcome

In this section we present the outcomes of the project. We provide some notes on the development of soft skills, summarize the implementation results of the last edition of the course, and discuss which mechanisms are introduced in order to prevent plagiarism.

A. Development of Soft Skills

Besides teaching and helping students develop their understanding of embedded systems, hands-on hardware and software system design skills, throughout the sessions we put effort in cultivating non-technical skills such as team building and management, sharing of work-load according to the expertise of the members, decision making, project presentation and technical report writing etc. We encourage novelty and out-of-the-box thinking during the implementation of the project. For example, to add a competition among the project teams, we set three different optimization goals, namely silicon-area, speed and flexibility and ask the students that they have to 'sell' their products to us. This friendly competition indirectly motivates the students to 'innovate' and every year we see innovations such as use of algorithmic optimization, fast adder structure, pipelining, etc.

B. Implementation Results

The implementation results of the project for the edition 2017 are presented in Figure 2. The plot visualizes where each codesign implementation sits in the area vs. speed trade-off.

In general, submitted designs can be clustered into two categories: teams who leverage on the HW for most computations fall into the top-left corner (green), while teams who keep functionalities on the SW side, sit into the bottom-right corner (red). Such results are to be expected: adding HW resources often results in faster yet larger designs, while SW keeps area consumption low at a cost of less performance.

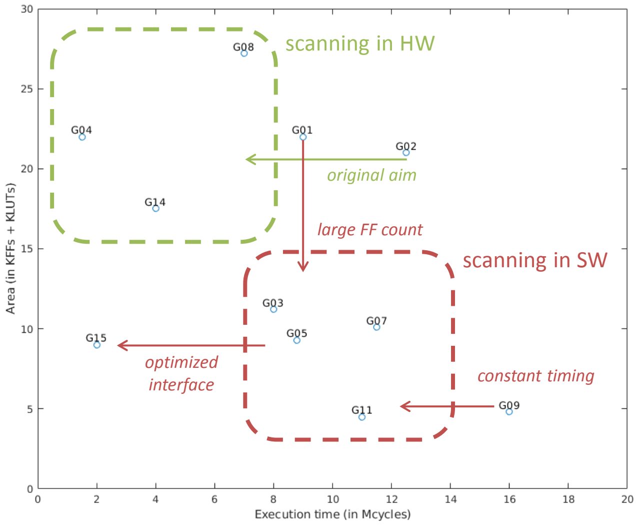

Fig. 2. Project results: area vs. speed trade-off.

Differences within clusters are a product of the various optimizations carried out by the teams and/or their skills in coding. In the HW side, for instance, some groups do not opt for a basic ripple-carry adder. Instead, they explore other constructions such as carry-select and carry-save adders. The reasons why some teams fall outside their ‘expected’ cluster are varied. Looking at the plot, one can see for instance that G15 achieves in a SW-oriented design a speed comparable to the HW-oriented designs. In this case, this was the result of heavy optimizations in the ARM code. Other groups, e.g. G09, were significantly slower that the rest of SW-oriented groups. The reason, however, was that the students opted to modify their high-level implementation in order to provide security against timing attacks [11]. Such out-of-the-box thinking is naturally rewarded during the grading phase.

C. Prevention of Plagiarism

To prevent use of source codes that were developed in the previous academic years, every year we adapt the project mildly. For the SW sessions, the selected Montgomery multiplication algorithm from [10] intentionally changes from year to year to limit plagiarism. For the codesign assignment, we leverage on the rich literature of implementing the RSA algorithm. The RSA algorithm can be implemented in various ways. For example, in the year 2015, the students were asked to implement a unified architecture that can compute both 1024 and 2048 bit RSA algorithms. In 2016, the goal of the project was to explore the design-space by tuning the width of the adder circuit for implementing the 1024-bit RSA. Till 2016, the performances of the HW/SW codesign projects were mostly determined by the hardware as most of the computation took place in the hardware. The role of the SW was limited to instructing the HW-based coprocessor. In 2017, we increased the contribution of the SW by introducing the Chinese Remainder Theorem (CRT) based RSA decryption. The HW computed the CRT-based decryption and the SW computed the relatively cheap encryption.

IV. Conclusions

In this paper we have presented a HW/SW co-design project for Design of Digital Platforms course, which aims at acquainting students with development in both SW and HW. They experience the development effort in each level, observe the design challenges, and find out the optimization potentials. The design tasks gradually building on top of each other allow them develop insights on how to partition problems, and how to handle verification and debugging steps. Furthermore, they are given plenty of freedom, which enable approaching to the problems diversely and introducing their particular solutions. Moreover, they acquire soft-skills such as project management, teamwork and presenting their work.

Acknowledgment

The authors would like to thank Xilinx for providing the Zybo development kits used in the course through the Xilinx University Program (XUP). Thanks are extended to former teaching assistants of the course (Begül Bilgin and Ruan de Clercq) for contributing to the development of the project sessions. This work is supported in part by the the Horizon 2020 research and innovation programme Cathedral ERC Advanced Grant 695305.

References

[1] P. Schaumont, *A Practical Introduction to Hardware/Software Codesign*.

[2] R. L. Rivest, A. Shamir, and L. M. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[4] V. S. Miller, “Use of Elliptic Curves in Cryptography,” in *Advances in Cryptology - CRYPTO ’85*, ser. Lecture Notes in Computer Science, H. C. Williams, Ed., vol. 218. Springer, 1985, pp. 417–426.

[5] P. Schaumont, “GEZEL: Hardware/Software Codesign Environment,” http://rijndael.ece.vt.edu/gezel2/, 2010, last visited: July 2018.

[6] Digilent Inc., “Zybo: Zynq-7000 ARM/FPGA SoC Trainer Board,” https://reference.digilentinc.com/reference/programmable-logic/zybo/start, 2012, last visited: July 2018.

[7] L. Uhsadel, M. Ullrich, A. Das, D. Karaklajic, J. Balasch, I. Verbauwhede, and W. Dehaene, “Teaching HW/SW Co-Design With a Public Key Cryptography Application,” *IEEE Trans. Education*, vol. 56, no. 4, pp. 478–483, 2013.

[8] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[9] P. L. Montgomery, “Modular multiplication without trial division,” *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[10] C. K. Koc, T. Acar, and B. S. Kaliski, “Analyzing and comparing Montgomery multiplication algorithms,” *IEEE Micro*, vol. 16, no. 3, pp. 26–33, 1996.

[11] P. C. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems,” in *Advances in Cryptology - CRYPTO ’96*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 104–113.