Practical experiences based on MIPSfpga

Daniel Chaver[1], Yuri Panchul[2], Enrique Sedano[2], David M. Harris[3], Robert Owen[2], Zubair L. Kakakhel[2], Bruce Ableidinger[2], Sarah L. Harris[4]

***Abstract*— In this paper we describe how to use MIPSfpga, a soft-core MIPS processor, to teach undergraduate and masters-level computer architecture courses. The most recent release of MIPSfpga (version 2.0), consists of three packages: the MIPSfpga Getting Started Guide, MIPSfpga Labs, and MIPSfpga System on Chip. After giving an overview of these packages, we provide examples of how to integrate MIPSfpga into curricula by describing three teaching experiences that used the MIPSfpga packages: an undergraduate course at the University Complutense of Madrid, a course at the Technical University of Darmstadt, and several seminars held at various Russian research centers and universities. MIPSfpga enabled students to bridge the gaps between theoretical concepts, hands-on practice, and industrial cores by allowing them to explore, modify, and test the MIPS core and system with the support of commercial compilers and tools.**

I. Introduction

The MIPSfpga project, developed by Imagination Technologies, is a comprehensive resource for computer architecture education. This infrastructure not only provides open access to the MIPS microAptiv UP soft-core processor, but also includes a large set of teaching materials and software tools. While soft-core processors have been available for several decades, MIPSfpga is the first unobfuscated commercial MIPS soft-core openly available to academics. The availability of the MIPSfpga core bridges the gap between existing curricula, that include toy MIPS processors, and industrial-level work with a real MIPS processor and its supporting tools.

The first version of MIPSfpga, released in June 2015, includes the Getting Started Guide package, which, in addition to the microAptiv UP soft-core processor, provides a thorough overview of the processor and system, the installers for the programming and debugging tools, and a set of scripts and examples. A companion MIPSfpga Fundamentals package provides nine labs that guide the students through setting up the hardware, programming and debugging the MIPS soft-core processor, and extending the processor to interact with various peripherals. Finally, a third package, called MIPSfpga-SoC, shows how to synthesize a System-on-Chip design centered on the MIPS soft-core that runs Linux and includes interfaces such as memory (DDR), UART16550, I2C, Ethernet, and an interrupt controller.

[1]Group of Architecture and Technology of Computing Systems (ArTeCS), University Complutense of Madrid, Spain

[2]Imagination Technologies Ltd., Kings Langley, United Kingdom

[3]Department of Engineering, Harvey Mudd College, Claremont, CA, U.S.A.

[4]Electrical and Computer Engineering, University of Nevada, Las Vegas, Las Vegas, NV, U.S.A.

In June 2017 the second version of MIPSfpga (MIPSfpga 2.0) was released. This new version still includes three packages: MIPSfpga Getting Started Guide (GSG), MIPSfpga Labs (which relpaces MIPSfpga Fundamentals), and MIPSfpga System on Chip (SoC). The GSG is extended with several new features in 2.0, such as a guide for downloading and debugging programs on the FPGA board without using the Bus Blaster probe, instructions for installing all MIPSfpga related software on either a Windows or Linux-based computer, and VHDL versions of the top-level MIPSfpga system modules. The MIPSfpga Labs package extends the Fundamentals package to include 17 new labs that examine and modify the core at a microarchitectural level, analyze and modify the cache structures and the cache controller, add user defined instructions (UDIs) via the CorExtend Interface, use interrupts and direct memory access (DMA) for Input/Output, and show how to modify the MIPS core itself.

This paper gives an overview of the MIPSfpga 2.0 infrastructure in Section II by describing its three packages (MIPSfpga GSG, MIPSfpga Labs, and MIPSfpga SoC) and by discussing how well these materials adapt to the IEEE/ACM Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering established at [1]. Section III shows how to integrate MIPSfpga into course curriculum by describing three example courses using the MIPSfpga 2.0 infrastructure. The last two sections discuss related work (Section IV) and conclude (Section V).

II. Overview of MIPSfpga 2.0

The MIPSfpga project encompasses three main sets of materials, available through [2], which we present in Subsections II-A, II-B and II-C (more details can be found in [3]). These materials align perfectly with the theoretical concepts explained in typical computer architecture courses. Actually, in Subsection II-D, we briefly describe the IEEE/ACM Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering [1] and justify how all of the topics included in those guidelines are covered by MIPSfpga 2.0.

A. MIPSfpga Getting Started Guide

The first package in MIPSfpga 2.0 is the MIPSfpga Getting Started Guide (GSG), which offers access to an unobfuscated commercial MIPS soft-core processor targeted to a field programmable gate array (FPGA). This soft-core is a version of the microAptiv UP core used in the popular Microchip PIC32MZ microcontroller and is composed of a set of Verilog HDL files that implement the MIPS32r3

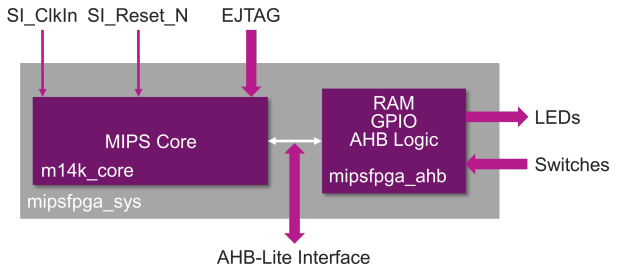

Fig. 1. MIPSfpga System

instruction set architecture in a 5-stage pipeline [4]. The package also includes the installers for the programming and debugging tools (Codescape MIPS SDK Essentials and OpenOCD), an overview of the MIPSfpga core and system, instructions on how to use the MIPSfpga hardware and programming tools, and a set of scripts and examples.

The MIPSfpga System illustrated in Figure 1 includes the MIPS soft-core and peripherals that communicate with the core via the AHB-Lite Interface. The peripherals include memory, implemented as block RAM on the FPGA, and general-purpose I/O (GPIO) that interacts with the LEDs and switches on an FPGA board. Given that many universities around the world opt to use Verilog or VHDL in their curricula, MIPSfpga 2.0 provides both Verilog and VHDL versions of the MIPSfpga top-level system modules.

The hardware required to use MIPSfpga are an FPGA board and a Bus Blaster probe. Although two boards are used as the example FPGA targets (Nexys4 DDR and DE2-115 boards), a detailed guide is provided for those who wish to retarget the MIPSfpga system to smaller boards (such as Basys3 or DE0 boards). The MIPSfpga GSG describes how to download, install, and use MIPSfpga development tools, which include a CAD tool, such as Vivado or Quartus II, for simulating and loading the MIPSfpga system onto an FPGA, and programming tools. These tools require a Windows or Linux-based operating system.

It is worth mentioning that Imagination Technologies has partnered with Europractice and MOSIS to offer academics and researchers access to a MIPS core for Multi Project Wafer (MPW) runs of silicon up to 100 pieces. The cores offered are the Warrior M-class 5100 or 5150 cores. The Warrior M-class is an extension of the microAptiv family aimed at the Internet of Things, wearable, and other embedded applications. Therefore, researchers have access to the latest evolution of the same core used in MIPSfpga. These cores offer the full configuration options including a Floating Point Unit (FPU), DSP, microMIPS Instruction set, and Hardware Virtualisation for enhanced security. These agreements position MIPSfpga as a comprehensive collection of teaching resources, providing materials to use from the first courses in Computer Architecture up to advanced architectural topics for master courses and a route to silicon for researchers.

B. MIPSfpga Labs

The second package in the MIPSfpga 2.0 infrastructure uses the MIPSfpga core and system provided in the Getting Started Guide to teach computer architecture and SoC design through hands-on learning. Some prior knowledge of digital design, computer architecture, and the MIPS instruction set architecture (ISA), for example the topics taught in [5], is required. Prior software programming experience is useful, but it can be taught concurrently if necessary.

This package includes 25 labs that guide the student through the underlying MIPSfpga setup and increasingly complex interactions with and extensions of the MIPSfpga core and system. The labs are divided into four parts: introduction, I/O, core, and memory system. The introductory labs (Labs 1-4) illustrate how to set up the MIPSfpga hardware and how to program and debug the MIPS soft-core processor. The I/O section (Labs 5-13) explains how to extend the system to interact with new peripherals. The third group of labs (Labs 14-19) delves into the microarchitectural details of the microAptiv core at the heart of MIPSfpga. Finally, the last set of labs (20-25) explore and modify the memory hierarchy. Table I gives a brief description of each lab, and further detail is given below.

TABLE I
MIPSFPGA LABS

Lab	Description
1	Create a Project in Vivado or Quartus-II
2	Learn how to compile, debug and run C programs
3	Learn MIPS Assembly Programming system
4	More C Programming Practice (optional)
5	Expand the system to add 7-segment displays
6	Expand the system to add a counter
7	Expand the system to add a buzzer
8	Expand the system to add an SPI-Light Sensor
9	Expand the system to add a SPI-LCD
10	Interact with peripherals using interrupts
11	Build a DMA engine for transfers between peripherals
12	Build a Data Encryption Standard (DES) engine
13	Learn how to use the Performance Counters
14	Execution of ADD and other arithmetic instructions
15	Execution of AND and other logic instructions
16	Execution of LW and other related instructions
17	Execution of BEQ and other related instructions
18	Learn how the Hazard Unit is implemented
19	Learn how to use the CorExtend interface
20	Introduction to the caches available in MIPSfpga
21	Analyze the D$ and implement new configurations
22	Cache Controller: Analyze a cache hit and miss
23	Cache Controller: Analyze D$ management policies
24	Cache Controller: Analyze the Store and Fill Buffers
25	Implement an Instruction Scratchpad RAM

Part 1 consists of four labs that introduce the tools for working with MIPSfpga. Lab 1 teaches how to build a MIPSfpga project targeted to an FPGA using either Xilinx's Vivado or Altera's Quartus II design software. This lab also shows how to target MIPSfpga to other FPGA boards, using the Basys3 and DE0 boards as examples. Labs 2 and 3 explain how to use the Codescape SDK (which consists of gcc and gdb targeted to MIPS) and the Bus Blaster probe to compile, download, run, and debug C and MIPS assembly programs on the MIPS core running on an FPGA. Lab

4 provides optional exercises for additional programming practice.

Part 2 begins with five memory-mapped input/output (I/O) exercises (Labs 5-9) for interfacing MIPSfpga with increasingly complex peripherals: the 7-segment displays on the FPGA boards, a millisecond counter for timing, a buzzer to play music, and two SPI devices (a liquid crystal display and a light sensor). A few extra components are needed to complete labs 7, 8, and 9, as detailed in [3].

Labs 10-12 analyze advanced I/O topics (such as interrupts and DMA). Lab 10 explains the basic usage of interrupts in MIPS CPUs. That lab also demonstrates how interrupts keep a processor from needing to constantly poll I/O ports, thus increasing the number of cycles available for computation and other non-I/O tasks. The next two labs analyze how to design, build, and test a direct-memory access (DMA) engine (Lab 11) and a Data Encryption Standard (DES) engine (Lab 12). Finally, Lab 13 explains how to configure and use the performance counters available in microAptiv. Example code is provided that sets up and uses the performance counters, and several exercises are proposed where the user evaluates the performance of example programs using various events. Labs 14-25 use this resource for evaluating the program performance.

Part 3 of MIPSfpga Labs delves into the internals of the core by showing how to use several microAptiv features and CorExtend, as well as describing detailed instruction flow through the pipeline. The first four labs (Labs 14 to 17) dive into the implementation of the microAptiv core and its pipeline. Students learn how the ADD, AND, LW and BEQ instructions are handled by the pipeline. The labs first introduce the stages of the microAptiv pipeline, showing how the analyzed instruction passes through each stage. This is followed by a step-by-step example simulation, showing where the main signals related to the given instruction are in the Verilog code (RTL). Finally, students are asked to analyze specific control signals, examine additional instructions, and add new instructions to the ISA supported by microAptiv.

Lab 18 explains and demonstrates microAptiv's Hazard Unit. This lab also introduces a switchable clock so that the system can run at a range of frequencies, from the usual multi-megahertz frequency down to about 1 Hz. At the slow frequency, users can explore program behavior in real-time by connecting system signals (such as pipeline, hazard control, or cache eviction signals) to LEDs.

The final lab in this part (Lab 19) shows how to use the CorExtend Interface available in MIPS processors. This interface is a powerful tool that allows designers to specify and implement their own instructions (User Defined Instructions, or UDIs). Through this interface, users can connect specialized hardware to boost the performance of critical algorithms beyond what can be achieved through the standard MIPS32 ISA. The lab describes the CorExtend interface, its capabilities and limitations, the placement of the module within the MIPS core, its timing properties, the interaction of the UDI unit with the microAptiv pipeline, and several exercises directing how to experiment with the UDI.

The final group of labs explores the MIPSfpga memory system, starting with the cache memories (Labs 20-24) and finishing with the implementation of a Scratchpad RAM (Lab 25). The analysis of MIPSfpga's cache memory system begins by demonstrating cache hits and misses using LEDs (Lab 20) in several example programs.

Lab 21 analyzes the various cache arrays that make up the cache used by microAptiv (i.e. the Data, Tag, and Way Select Arrays). The lab describes both the array interfaces and their internal implementations. After these detailed explanations, the students are asked to implement and test new cache configurations and to test several code optimization techniques using the performance counters.

The next three labs (22-24) delve into the cache controller. Lab 22 analyzes the management of cache hits and misses and describes the main stages, structures, and signals involved in a cache hit or miss. Provided simulations also illustrate the described concepts. Finally, proposed exercises explore the cache system by, for example, evaluating the miss penalty involved in a cache miss. Lab 23 describes the cache management policies supported by the microAptiv UP processor. The exercises prompt students to evaluate several typical allocation and write policies and also to implement new replacement policies. Finally, Lab 24 explains the operation of the Store Buffer, which temporarily holds the data to write in the data cache by a store, and the Fill Buffer, which temporarily holds the block to fill into the data cache after a miss.

Finally, Lab 25 shows how to add an Instruction Scratchpad RAM to MIPSfpga. The basic MIPSfpga configuration includes the Scratchpad interface but no actual Scratchpad RAM. This lab shows how to add this Scratchpad RAM block and how to communicate with it through the I/D cache controller.

Once the students complete this package, they are ready to develop more advanced projects, such as adding interfaces to drive additional peripherals (for example I^2C or UART), adding new features to the core (such as a hardware prefetcher or a branch predictor) or to the memory system (such as a second cache level or a way predictor), or any other project that the instructor may choose.

C. MIPSfpga-SoC

The final package in the MIPSfpga 2.0 materials is the MIPSfpga-SoC package, which shows how to extend MIPSfpga to build an SoC system that loads the open source Linux operating system. This package gives a detailed view of how the SoC in embedded systems is designed and built up in layers to run complex software.

The Linux SoC is built with the MIPS core as the master controlling peripherals, acting as slaves, across the AHB-Lite bus (Figure 2). The slaves are implemented mostly using Xilinx IP blocks, which greatly reduces design time. All peripherals connect to the MIPS core using the AHB-Lite interface and memory-mapped I/O. However, because the supplied Xilinx blocks use an Advanced eXtensible Interface (AXI), an AHB-Lite to AXI bridge is used to connect the

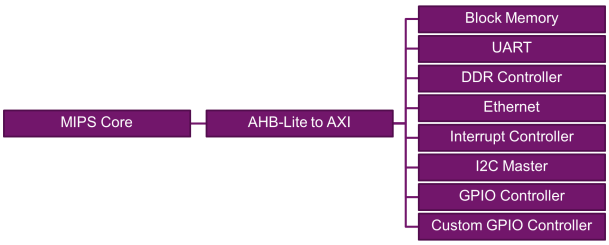

Fig. 2. Simplified block diagram for the Linux SoC

MIPS core with the Xilinx-supplied IP. All of the blocks are Xilinx IP blocks except the MIPS core and the custom GPIO module, which are supplied by Imagination. This simple GPIO module allows students to understand what it takes to build a slave peripheral, interface it to the interconnect, and use it to communicate with the MIPS CPU and interact with the physical world via simple switches and LEDs.

A Linux OS can be divided into two parts: the Linux Userspace and the Linux Kernel. The Userspace interacts with the hardware via standardized system calls provided by the Linux Kernel. MIPSfpga-SoC uses Buildroot, one of the most scalable Linux Userspace flavors. The Kernel interacts directly with hardware and provides a layer of abstraction. A Linux kernel can be implemented with minimum hardware support: the system must have a processor with an MMU, interrupt controller, timer interrupts, UART, memory, and an e-JTAG interface. Note that all these features are included in the MIPSfpga Linux SoC described above (Figure 2).

Additional source code in the form of patches to the Linux Kernel is provided as part of the MIPSfpga-SoC package. This adds support for the MIPSfpga soft-SoC platform in the Linux Kernel. As microAptiv UP and the IP blocks we used are already supported in the kernel, we reuse existing code while only needing to add the MIPSfpga-SoC platform description. Buildroot compiled for the mips32r2 ISA can then be loaded by the kernel running on MIPSfpga-SoC. The resulting system can run applications relying on the GNU standard C libraries.

D. IEEE/ACM Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering

The IEEE/ACM Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering [1] establish nine relevant units for the Computer Architecture and Organization knowledge area, as summarized in Table II.

The labs in MIPSfpga offer extensive coverage for the units detailed in the IEEE guidelines. The MIPS ISA has been around since the early 80's and it did set the base ideas for many other later architectures, therefore playing a key role in the history of computer architectures covered by CE-CAO-1. Labs 2 and 3 instruct students on creating projects in Vivado and compiling, running and debugging programs, therefore lining up with CE-CAO-2. Also in lab 2, along with labs 3 and 4, many aspects of the MIPS ISA are studied and detailed, which can be used as part of the studies covered by unit CE-CAO-3. Performance measurements, for

TABLE II
IEEE CURRICULUM GUIDELINES

Unit	Name
CE-CAO-1	History and overview
CE-CAO-2	Tools, standards and/or constraints
CE-CAO-3	Instruction set architecture
CE-CAO-4	Measuring performance
CE-CAO-5	Computer arithmetic
CE-CAO-6	Processor organization
CE-CAO-7	Memory system organization and architectures
CE-CAO-8	Input/Output interfacing and communication
CE-CAO-9	Peripheral subsystems
CE-CAO-10	Multi/Many-core architectures
CE-CAO-11	Distributed system architectures

CE-CAO-4, are introduced in lab 13 through the Performance Counters. This resource is used to analyse the impact of different events and combinations of instructions in various elements related to performance across the core. Moreover, Performance Counters are relied upon through labs 14-25. In particular, lab 14 delves into the ADD operation and other arithmetic instructions, providing hands-on materials for unit CE-CAO-5. Processor organization (CE-CAO-6) is studied in labs 14-18 which, as stated earlier, study the pipelined structure of the MIPS core. Labs 20-25 present an exhaustive look at the memory hierarchy, including caches and scratchpad RAMs, offering more than enough coverage for unit CE-CAO-7. Units CE-CAO-8 and CE-CAO-9 are addressed in labs 5-12, where various I/O devices are connected with the MIPSfpga system, interrupt-based communication is analysed, and DMA transfer and encryption are examined.

The two final units, CE-CAO-10 and CE-CAO-11, are not directly addressed in the MIPSfpga materials, as they study multi-core and distributed architectures, and MIPSfpga is a single-core system. Nevertheless, the availability of MIPSfpga as an open system for academia make it possible for professors to use it as a base for their advanced materials. As an example, the authors in [6] have engaged in some heavy modifications of the source code of the microAptiv core contained in the MIPSfpga system to develop a 120-core system that then can be downloaded onto a Terasic DE5-NET FPGA.

III. PRACTICAL EXPERIENCES

MIPSfpga 2.0 has already been used in several academic courses, workshops, and hackathons. In this section we describe three of these uses to exemplify how to integrate MIPSfpga into courses and to illustrate some additional uses. These courses focus on computer architecture and SoC design using MIPSfpga as the development platform.

A. Course at University Complutense of Madrid

During the second semester of 2016/2017 (February to June 2017), we used MIPSfpga 2.0 as part of an undergraduate course on *Computer Architecture & Integrated Systems*, a compulsory subject in the fourth year of the *Telecommunications Engineering* degree offered at University Complutense of Madrid. The students of this course

have a strong background in digital design, VHDL, computer organization (MIPS ISA, single/multi-cycle processors, and Input/Output systems), and programming (C++). This section gives an overview of the course and describes the labs, assessment, and student feedback.

1) Course Overview: The 12-week course consists of 24 1.5-hour lectures (2 per week) with four modules. Module 1 reviews material taught in prior courses: namely, the MIPS ISA, the single- and multi-cycle processor and the Input/Output system. Module 2 describes the MIPS pipelined processor and explores its implementation using examples and exercises. Module 3 explores the cache hierarchy, and Module 4 introduces System on Chip and Embedded System design. The MIPSfpga 2.0 resources are perfectly suited to this course.

We use [5] as the main textbook, focusing on Chapters 4 (Hardware Description Languages), 6 (Architecture), 7 (Microarchitecture) and 8 (Memory and I/O Systems). Moreover, we use the lecture slides provided with that book. In order to link the theoretical contents with MIPSfpga during the lectures, we have extended the original slides with some extra material that explain MIPSfpga and compare the pipelined processor and memory and I/O systems explained in [5] with those of MIPSfpga (see Figure 1).

2) Labs: Interleaved with the 24 1.5-hour lectures, the course includes 12 weekly 2-hour lab sessions. Given that MIPSfpga includes 25 extensive labs, we cannot complete all of them but have to select a smaller subset. We selected the following labs from the MIPSfpga Labs package (see Table I for an explanation of the lab contents) and from the MIPSfpga SoC package: 1, 2, 3, 4, 5, 13, 14-18, 22 (MIPSfpga Labs) and Advanced Starter Tutorial (MIPSfpga SoC). Table III specifies the association between the labs and the course modules described in the previous subsection.

TABLE III
ASSOCIATION BETWEEN MODULES AND LABS

Module	Module contents	Labs
1	Review MIPS ISA	Labs 2, 3 and 4
	Single/multi-cycle procs	
	Input/Output system	Lab 5
2	The pipelined processor	Labs 13-18
3	The cache hierarchy	Labs 22-A and 22-B
4	SoC and Embedded	SoC - Starter Tutorial

Table IV shows the lab schedule. Obviously, it is essential to explain the theoretical concepts associated with each lab before the corresponding lab session. Thus, according to the schedule (Table IV), we review the MIPS ISA before the 2^{nd} lab session, the input/output system before the fourth session, and we explain the pipelined processor and the memory system before the 6^{th} and 9^{th} sessions, respectively.

In the first lecture, held several days before the first lab session, we distribute the Nexys4 DDR FPGA boards and the Bus Blaster probes and ask the students to download the MIPSfpga GSG package from [2], install the software tools on their laptops as explained in the GSG document (specifically, Windows users complete Appendices B and D, and Linux users complete Sections 2 and 3 of Appendix G), and test the installation by completing the corresponding sections of the GSG document (specifically, Windows users complete Sections 4.1, 4.2.1, and 7.4, and Linux users complete Section 4 in Appendix G). In this first lecture, we also ask the students to begin studying the Verilog hardware description language (HDL) on their own, using Chapter 4 of [5], as their prior HDL knowledge is in VHDL.

TABLE IV
LAB SCHEDULE

Session	Description
-	Homework: Before the first session, the students must install MIPSfpga as explained in the GSG
1	Finish MIPSfpga installation + **Lab 1**
2	**Labs 2** and **3** (C and Assembly Programming) + **Lab 4** (Image Transformation)
-	Homework: Complete Lab 4
3	Extra exercise Lab 4 + Test
4	**Lab 5** (7-Segment Displays)
-	Homework: Complete Lab 5
5	Extra exercise Lab 5 + Test
6	**Lab 13** (Performance Counters)
-	Homework: Complete Lab 13
7	Extra exercise Lab 13 + Test
8	Teamwork preparation (**Labs 14-18**)
-	Homework: Complete Labs 14-18 in teams
9	**Lab 22-A** (Hit management)
-	Homework: Complete Lab 22-A
10	**Lab 22-B** (Miss management)
-	Homework: Complete Lab 22-B
11	Extra exercise Lab 22 + Test
12	**MIPSfpga SoC** - Advanced Starter Tutorial

Each lab is completed during 1-2 lab sessions. For example, Lab 13 is completed in sessions 6 and 7: in session 6, the students start working on Lab 13. They complete the lab at home using the homework slot between the 6^{th} and the 7^{th}) sessions. Then, in the second of these sessions (the 7^{th} session), the students complete an additional exercise and a test that poses questions about the lab.

Labs 14-18 are completed in teams with 3-4 members per team. Each of the five teams completes one of the five labs at home and then explains that lab to the other students in 1 hour. After the presentations, the students ask questions and discuss their opinions for 30 minutes.

All students that attended the lab sessions were able to complete most of the labs. Actually, even for the most complex exercises (such as the ones proposed at the end of labs 14-17, in which the students must include new instructions in the core), more than 60% of the students were able to complete them with minimal or no guidance at all.

3) Final grade: The final grade of the course was computed as follows: 0.5*FE + 0.3*LM + 0.2*TM; where FE is their final exam grade, LM their lab sessions grade, mainly based on the tests specified in Table IV, and TM is their grade obtained for teamwork.

4) Opinions about the course: We would like to thank the students that took part in this course (P. Fernandez, M. Sanchez, G. Diaz-Tejeiro, C. Oliver, J. Alvaro, A. Villarin, M. Cereceda, M. Perez, A. Dorda, P.M. Teba, F.J. Oliva, E.I. Quezada, D. Fernandez, J.A. Canadas, A. Menendez, J. Martin, I. Diaz) for their revision of this report and their valuable feedback. Specific student feedback includes the following comments: "The course, especially the lab, gives a comprehensive understanding of the architecture of a computer" (G. Diaz-Tejeiro); "The lab sessions have allowed us to learn how a commercial processor works" (M. Sanchez); "The texts of the labs facilitate the learning due to their good and complete writing" (J. Martin); "At the beginning, it was really hard to get used to the new way of learning. However, once we got the main skills, we began to enjoy the subject, obtaining a perfect knowledge about how the core, cache, I/O, and other elements work. In my opinion this is the best way to learn a subject like this" (A. Menendez); "The labs allowed us to implement the concepts explained during the lessons" (P. Fernandez); "The labs have shown the application of computer architecture concepts to the real world" (A. Villarin).

B. Course at the Technical University of Darmstadt

During the second semester of 2015/2016, we taught a course entitled Fundamentals of Processor Architecture and Memory-mapped I/O at the Technical University of Darmstadt (TUD) in Germany. The 14-week course consists of weekly lectures and weekly or bi-weekly lab assignments. Table V lists an overview of the eight labs and associated lectures. The first three labs are provided with the textbook Digital Design and Computer Architecture [5] and the remaining labs (4-8) are from the MIPSfpga Labs distribution (Table I).

TABLE V
PROCESSOR ARCHITECTURE & MEMORY-MAPPED I/O LABS

Lab	Description	Duration
1	MIPS Assembly Program	1 Week
2	MIPS Single-Cycle Processor	1 Week
3	MIPS Pipelined Processor	2 Weeks
4	MIPSfpga Tutorial	1 Week
5	MIPSfpga Memory-Mapped I/O - Buzzer	1 Week
6	MIPSfpga Memory-Mapped I/O - LCD	2 Weeks
7	MIPSfpga DMA Engine	2 Weeks
8	MIPSfpga DES Encryption	2 Weeks

The first three lectures and labs use material from chapters 6 and 7 of [5]; they introduce the MIPS instruction set architecture (ISA) and processor architecture. The remaining lectures and labs are based on slides and labs provided with the MIPSfpga Labs package from Imagination Technologies. This course's lab 4 (MIPSfpga Tutorial) combine Labs 1-3 of MIPSfpga Labs. The four remaining labs are used directly from MIPSfpga Labs (corresponding to MIPSfpga Labs 7, 9, 11, and 12) .

Lectures 1-3 cover MIPS ISA, processor datapath and control, and pipelined processor architecture. The next two lectures introduce the MIPSfpga processor, system, and development environment. The last four lectures conclude with (1) introducing memory-mapped I/O and showing how to interface peripherals with memory-mapped I/O in MIPSfpga, (2) giving an overview of serial interfaces and liquid crystal displays (LCDs), (3) discussing direct-memory access (DMA) and how to use the AHB-Lite bus to implement it, and (4) discussing the data encryption standard (DES). The last four weeks of the semester contain no lectures; the students focused on completing the last two labs. However, an instructor could lecture on additional topics during that time, as desired.

The students receive the required hardware (Nexys4 DDR FPGA board, Bus Blaster probe, buzzer, LCD, wires, and capacitors) during the first week of class, and they install the tools (based on the MIPSfpga GSG) during Lab 1. A lab TA is available for 2-4 hours per week to assist students, although many students complete the labs at home on their own and attend the lab sessions only as needed for help with debugging. Student feedback for the course was positive, and it earned notably high enrollment and completion among hardware labs at the TUD. All 42 students who enrolled in the course also completed it.

C. Seminars in Russia, Ukraine, and Kazakhstan

During 2015/2016 we held a series of short MIPSfpga seminars in nine top universities in Russia, Ukraine, and Kazakhstan (National Research University of Electronic Technology, Lomonosov Moscow State University, National Research Nuclear University MEPhI, ITMO University, Samara State Aerospace University, Moscow Institute of Physics and Technology State University, Almaty Management University, Nautech Corporation, and the National Technical University of Ukraine). In addition, MIPSfpga was presented during four events in Russia and Kazakhstan (Microchip Masters Russia, a conference at Kazakh National Research Technical University, SECR conference, and Nanometer ASIC lectures at the National University of Science and Technology MISiS). These locations and events had a broad range of participants, ranging from students with some experience in embedded programming but no experience in hardware design, to sophisticated researchers who design ASICs and look for inexpensive FPGA-based platform to experiment with hardware-software codesign using a commercial core with a mature software toolchain.

During 2015 seminars, we confirmed that the introduction of MIPSfpga works best after students learn the basics of digital design, computer architecture, and microarchitecture.

Thus, before MIPSfpga, students should learn these principles using a simplified MIPS core implemented in a few lines of Verilog (such as the implementation provided in [5]). The concepts of processor structure, pipelining, stalls and forwarding are learned first using this minimal core. Once the basics are established, students can turn to MIPSfpga to experiment with the industrial core, observing the pipeline in action, the work of caches, adding custom coprocessors, integrating peripherals with SoC fabric and building multi-core systems. This approach was well-received in universities that already taught students how to design simple cores in an HDL (such as MIET, ITMO and KPI). The feedback from the seminars included a suggestion from the researchers in Moscow State University that MIPSfpga code can be used as a comprehensive test for new EDA tools. Some of the ideas discussed were incorporated into MIPSfpga 2.0, such as the ability to slow the system clock down to a very low (1 Hz) frequency, the ability to upload programs onto the synthesized system via a USB-to-UART converter, thus eliminating the need to use the Bus Blaster probe, and the incorporation of new labs that emphasize the processor internals and the CorExtend feature.

During 2016 seminars we combined MIPSfpga with other courses. In Almaty Management University (Kazakhstan), we put exercises with MIPSfpga at the end of an introduction into HDL-based design [7], exposing beginning HDL users to industry-grade designs. In Nanometer ASIC Seminars, we used MIPSfpga to prototype an SoC and then we described a path to create an ASIC based on MIPS M5150 (a close relative of the MIPS microAptiv UP offered by Europractice for ASIC design). Finally, in the National Technical University of Ukraine, we organized a hackathon [8] where teams of students integrated MIPSfpga with the sensors that use SPI, I2C, and other protocols. The event proved a success and the challenge of the task was just right: half of participants completed working solutions and learned about RTL coding and system integration while doing it.

As a result of these MIPSfpga seminars, a large group of MIPSfpga users in Russia and Ukraine exist who are contributing to the emerging MIPSfpga community. For example, two groups created custom CorExtend coprocessors ([9] and a group in MIET Zelenograd). Also, Stanislav Zhelnio, crated multiple MIPSfpga extensions, including an interface to an SDRAM controller, an interrupt controller, a debug interface with Visual Studio Code integration and UART16550 support, with the aim to port Linux to MIPSfpga-based system running on Terasic's DE10-Lite board. Some of these projects will be presented at the NGC-17 conference at Tomsk, including a project integrating MIPSfpga with the popular Wishbone system bus (MIET) and a project that adds a VGA peripheral (Boris Ivashinnikov from Komsomolsk-na-Amure State Technical University). In addition, several educators, including Ilya Kudryavtsev from Samara National Research University, will discuss the integration of such projects into the mainstream university curriculum.

IV. Related Work

In this section, we analyze other soft-core processors available today and describe other materials available for computer architecture education; each of these alternatives is briefly compared with the MIPSfpga 2.0 infrastructure.

A. Other soft-core processors

Xilinx and Altera offer their own soft-cores, Nios/Nios II [10] and MicroBlaze [11], respectively, specifically configured for their FPGAs. These alternatives however have disadvantages: they are not open-source, which limits their use substantially; they are neither based on industrial/commercial cores nor support a commercial ISA; and they lack teaching-oriented documentation. ARM also provides a non-open-source alternative, the Cortex M0 Design Start [12], which is a basic (8K gates), low performance soft-core. It is completely obfuscated and has primitive debug support because it does not include e-JTAG. Also, it does not provide a route to silicon for academia and it provides few teaching materials.

Several open-source soft-cores also exist. Two well-known alternatives, both supporting a SPARC RISC ISA, are the OpenSPARC family [13] (developed by Oracle and Sun Microsystems) and the LEON family [14] (developed by Aeroflex Gaisler and the European Space Agency). Although they are interesting, they lack good teaching materials and they implement an ISA not as widely used in academia as MIPS. Two other open-source alternatives worth mentioning are RISC-V [15], started at the University of California, Berkeley, and openRISC, developed by opencores.org [16]. These cores do not implement commercial ISAs and, as in the previous cases, provide few teaching materials.

MIPSfpga, on the other hand, addresses all constraints mentioned above. It is an industrial-level open-source soft-core that is completely unobfuscated and that is used in important commercial devices including Microchip's PIC32MZ. It implements an ISA (MIPS32r3) widely used in academia and with a wide range of existing documentation and support. MIPSfpga also provides extensive documentation, including a large amount of teaching materials and labs that, as analyzed in this paper, align perfectly with typical computer architecture courses. MIPSfpga also provides support across FPGA platforms, including both Xilinx and Altera FPGAs, and it is easily extendible to other FPGAs.

B. Other materials for computer architecture education

The work in [17] presents the HIP environment to show students how a pipelined processor works. It uses a simple 5-stages soft-core similar to some early MIPS processors, and connects the FPGA on which the design is loaded to a user GUI on the PC that shows the state of the core in each step. The ISA of the core used in this work consists of 52 instructions, far from the support for the full MIPS32r3 ISA of MIPSfpga. Also, the environment does not include a prepared set of labs to be used along with it, and the book where the core is described is only available in Slovene. On the other hand, MIPSfpga does not provide a GUI as the one in this environment, but lecturers and students have

unrestricted access to the RTL of the core, which allow them to explore the value and contents of every signal and register in the processor by inspecting the waves in an RTL simulator.

The authors in [18] introduce a computer architecture simulator called BZK.SAU. The ISA of this simulator consists of 59 instructions. Simple cores such as these allow students to experiment with some of the concepts they learn, but they do not bridge the gap between what they see in the textbooks and actual industrial-level processors. Additionally, the latter work is exclusively based on simulation, meaning that students cannot download the completed design onto an FPGA and experiment with it working on actual hardware.

The CNP laboratory presented in [19] puts together a simple environment to teach the basics of computer architecture, compilers, and networking. MinIPS, the pipelined processor at the core of this laboratory, is a minimized MIPS ISA, and the compiler that goes along with it, Tiny C, is designed to comply to this reduced set of instructions. Although the labs in MIPSfpga are not targeted at courses in compilers or networking, nothing stops lecturers in these topics from using the contents of the MIPSfpga Getting Started package as hands-on materials for their courses. The MIPS32r3 ISA is well known, extensively documented, and lecturers may implement their own compiler or choose among the different existing compilers that support the MIPS ISA, such as Codescape MIPS SDK, gcc, or LLVM. Also, as the MIPSfpga-SoC package shows, it is perfectly possible to connect the MIPSfpga core to an Ethernet adaptor.

Finally, the authors in [20] introduce a course on advanced multi-core architectures. This type of course is beyond the initial scope of MIPSfpga but, as stated previously, lecturers can adapt the GSG package to their own needs, as in [6].

V. Conclusions

In this paper we have analyzed the latest version of MIPSfpga, released in June 2017, and its application to several courses and seminars. As analyzed throughout this paper, this teaching infraestructure aligns perfectly with the concepts studied in typical computer architecture courses. Moreover, by using MIPSfpga in computer architecture education, students must deal with problems similar to those that a computer hardware engineer must resolve, such as integrating peripherals into an SoC system, including new instructions by using a standard interface (such as CorExtend) or by modifying a commercial core (such as microAptiv), evaluating various cache configurations and policies using performance counters, or building a complete SoC system, based on a commercial core and Xilinx IPs, that runs Linux. Although simpler approaches (such as the one presented in [5], based on a highly simplified core) are possible and perhaps more suitable for the initial undergraduate courses, an industrial-level approach (such as the one described in this paper) has the benefit of tackling projects much closer to the ones that the students will face in the professional world, thus making MIPSfpga well-suited for upper-division undergraduate and masters-level courses.

Acknowledgment

The authors would like to acknowledge the contributions from the Imagination University Program, the University of Nevada, Las Vegas, Imperial College London (UK), the ArTeCS group at University Complutense of Madrid (UCM), the Spanish government research contracts TIN2015-65277-R and TIN2015-65316-P, Munir Hasan (IMG UK), Prashant Deokar (IMG India), Mahesh Firke (IMG India) Parimal Patel (Xilinx), Kent Brinkley (IMG USA), Rick Leatherman (IMG USA), Chuck Swartley (IMG USA), Sean Raby (IMG UK), Michio Abe (IMG Japan), Bingli Wang (IMG China), Sachin Sundar (IMG USA), Alex Wong (Digilent Inc.), Matthew Fortune (IMG UK), Jeffrey Deans (IMG UK), Laurence Keung (IMG UK), Roy Kravitz (Portland State University), Dennis Pinto (UCM), Tejaswini Angel (Portland State University), Christian White, Gibson Fahnestock, Jason Wong, Cathal McCabe (Xilinx), and Larissa Swanland (Digilent).

References

[1] 'CE2016 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering' IEEE and ACM, 2016.
[2] 'Imagination University Program - Resources', https://community.imgtec.com/university/resources
[3] Harris, S., Harris, D., Chaver, D., et al.: 'MIPSfpga: Using a Commercial MIPS Soft-Core in Computer Architecture Education.'. IET Circuits, Devices and Systems, 2017.
[4] Imagination Technologies Ltd., 'MIPS32 microAptiv™ UP Processor Core Family Datasheet', July 31, 2013
[5] Harris, D., and Harris, S., 'Digital Design and Computer Architecture' (Elsevier Science and Technology, 2007, 2nd edn. 2012)
[6] Kumar H B, C., Ravi, P., Modi, G., Kapre, N..: '120-core microAptiv MIPS Overlay for the Terasic DE5-NET FPGA board', Int. Symp. on Field-Programmable Gate Arrays, Monterey, USA, February 2017
[7] 'Intro HDL design', http://www.almau.edu.kz/en/9891
[8] 'NTUU Hackathon', https://www.imgtec.com/blog/ukraine-mips-fpga-hackathon-success
[9] 'CorExtend Interface', http://zatslogic.blogspot.com/2016/01/using-mips-microaptiv-up-processor.html
[10] 'Altera - NIOS-II Processor', https://www.altera.com/products/processors/overview.html, February 2017
[11] 'Xilinx - MicroBlaze Soft Processor Core', http://www.xilinx.com/products/design-tools/microblaze.html, accessed February 2017
[12] 'ARM - Cortex M0 Design Start', http://www.arm.com/products/designstart/index.php, accessed February 2017
[13] 'Oracle - OpenSPARC', http://www.oracle.com/technetwork/systems/opensparc/index.html, Feb-2017
[14] 'Aeroflex Gaisler - LEON series Softcores', http://www.gaisler.com/, accessed February 2017
[15] Waterman, A., Lee, Y., Patterson, D.A., et al., 'The RISC-V Instruction Set Manual, Volume I: User-Level ISA', version 2.0, 2014
[16] 'OpenCores OpenRISC', http://opencores.org/or1k/Main_Page, accessed February 2017
[17] Bulić, P., Guštin, V., Šonc, D., and Štrancar, A,.: 'An FPGA-based integrated environment for computer architecture', Computer Applications in Engineering Education, 2013, 21, (1), pp. 26-35
[18] Oztekin, H., Temurtas, F., Gulbag, A.: 'BZK.SAU: Implementing a hardware and software-based computer architecture simulator for educational purpose'. Proc. 2nd Int. Conf. Computer Design and Applications, Qinhuangdao, China, June 2010, pp. 490-497
[19] Abe, K., Tateoka, T., Suzuki, M., Maeda, Y., Kono, K., Watanabe, T.: 'An integrated laboratory for processor organization, compiler design and computer networking', IEEE Trans. Education, 2004, 47, (3)
[20] Petit, S., Sahuquillo, et. al: 'A research-oriented course on Advanced Multicore Architecture: Contents and active learning methodologies', Journal of Parallel and Distributed Computing, Elsevier, 2017