IET Circuits, Devices & Systems

Research Article

IET Journals
The Institution of Engineering and Technology

MIPSfpga: using a commercial MIPS soft-core in computer architecture education

ISSN 1751-858X
Received on 30th September 2016
Revised 19th February 2017
Accepted on 27th February 2017
doi: 10.1049/iet-cds.2016.0383
www.ietdl.org

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

[1]*Department of Electrical and Computer Engineering, University of Nevada Las Vegas, Las Vegas, NV, USA*
[2]*Department of Engineering, Harvey Mudd College, Claremont, CA, USA*
[3]*Department of Computer Architecture and Automation, Complutense University of Madrid, Madrid, Spain*
[4]*Imagination Technologies Ltd., Kings Langley, Hertfordshire WD4 8LZ, UK*
✉ *E-mail: Sarah.Harris@unlv.edu*

Abstract: In this study, the authors introduce MIPSfpga and its accompanying set of learning materials. MIPSfpga is a teaching infrastructure that offers access to the non-obfuscated Register-Transfer Level (RTL) source code of the MIPS microAptiv UP processor. The core is made available by Imagination Technologies for academic use and is targeted to a field-programmable gate array (FPGA), making it ideal for both the classroom and research. The supporting materials and labs focus on hands-on learning that emphasises computer architecture, system on chip (SoC) design and hardware–software codesign. Among other things, students learn to set up the MIPS soft-core processor on an FPGA, run and debug programs on the core in simulation and in hardware, add new peripherals to the system, understand the microarchitecture and extend it to support new features, experiment with different cache sizes and content management policies, add new instructions using the CorExtend interface available in MIPS processors, and understand SoCs in embedded systems and how they are designed and built up in layers to run complex software such as Linux.

1 Introduction

The recent availability of MIPSfpga, a teaching infrastructure based on a soft-core processor provided by Imagination Technologies under a free license, enables students to learn about computer architecture and hardware–software codesign on a commercial MIPS core. Students can use the same hardware tool, a field-programmable gate array (FPGA), to experiment with both digital design and computer architecture.

While soft-core processors have been available for several decades, MIPSfpga is the first commercial MIPS core openly available to academics. Many colleges and universities, including ours, use MIPS to teach computer architecture. The availability of the MIPSfpga core bridges the gap between existing curricula, which include toy MIPS processors, and industrial-level work with a real MIPS processor and its supporting tools.

The MIPSfpga project encompasses three main sets of materials, available through [1], which we present in the remainder of the paper: MIPSfpga Getting Started Guide (GSG), MIPSfpga Labs, and MIPSfpga-SoC. We begin with an overview of the MIPSfpga core as described in the GSG package (Section 2), given that it is the foundation of the other two sets of materials. We then introduce the MIPSfpga Labs package (Section 3), which includes 25 labs. These labs explain, among other things, how to set up the microAptiv core on an FPGA and how to run and debug programs on the core, how to expand MIPSfpga to add new peripherals and communicate with them via interrupts or direct memory access (DMA), how to expand the core with new instructions, both using the CorExtend feature and by modifying the core's hardware itself, and how to use the Performance Counters for experimenting with different memory hierarchies, cache sizes, associativities and content management policies. Once students are familiar with the functioning of the industrial-level core, the final set of materials, MIPSfpga-SoC (system on chip), shows how to design, synthesise and load an SoC design around the MIPSfpga core and port and run the Linux kernel on it (Section 4). In the final sections, we discuss other existing soft-core options (Section 5) and conclude (Section 6).

2 MIPSfpga overview

MIPSfpga offers access to an unobfuscated commercial MIPS soft-core processor targeted to an FPGA. This soft-core is provided as a set of Verilog hardware description language (HDL) files as part of the MIPSfpga Getting Started package [1], which is freely available to academics from Imagination Technologies after registration. The package also includes the installers for the programming and debugging tools (Codescape MIPS SDK (software development kit) Essentials and OpenOCD), a thorough GSG, and a set of scripts and examples. This section gives an overview of the MIPSfpga core and system and describes the hardware and software required to run MIPSfpga.

2.1 MIPSfpga core

The MIPS soft-core used in MIPSfpga is a version of the microAptiv UP core used in the popular Microchip PIC32MZ microcontroller. The soft-core is composed of a set of Verilog HDL files that implement the MIPS32r3 instruction set architecture (ISA) in a 5-stage pipeline [2]. The released core includes a memory management unit (MMU) with translation lookaside buffer, instruction and data caches, and several interfaces (such as EJTAG), as shown in Fig. 1. The bus interface unit supports the Advanced Microcontroller Bus Architecture (AMBA) 3 Advanced High-performance Bus (AHB)-Lite protocol [3]. Detailed specifications of the full core can be found in the datasheet [2].

2.2 MIPSfpga system

The MIPSfpga system [4], as shown in Fig. 2, includes the MIPS core and peripherals that communicate with the core via the AHB-Lite Interface. As stated above, the MIPSfpga core is provided in Verilog HDL (VHDL) only. However, given that many universities around the world opt to use VHDL in their curricula, the GSG includes both Verilog and VHDL versions of MIPSfpga system top-level modules.

The peripherals include memory, implemented as block RAM on the FPGA, and general-purpose I/O (GPIO) that interacts with

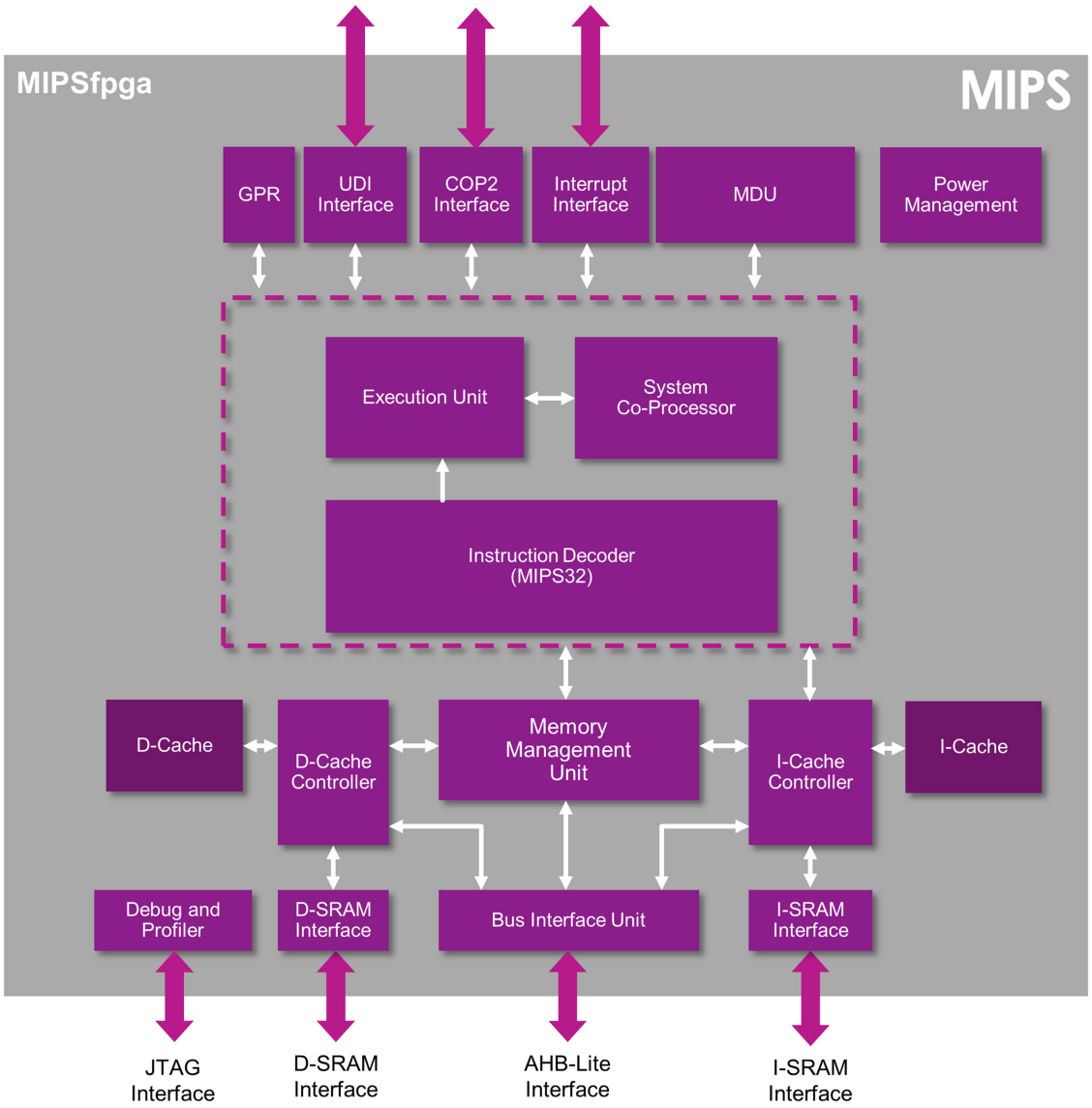

Fig. 1 *MIPSfpga core*

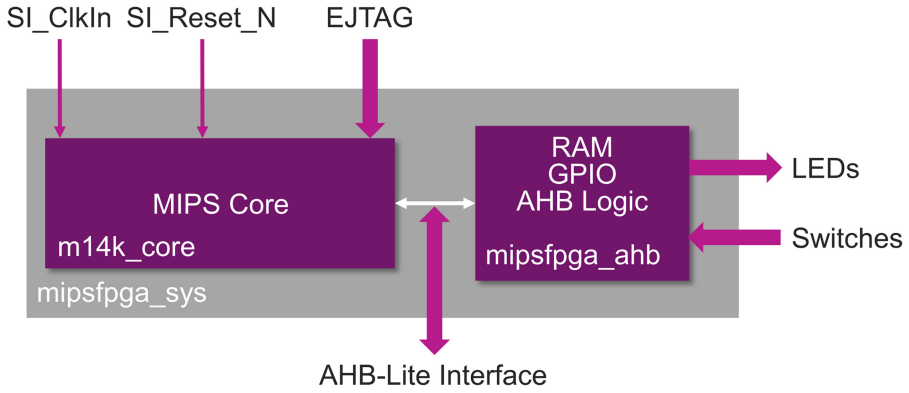

Fig. 2 *MIPSfpga system*

the LEDs and switches on an FPGA board. The system requires at a minimum a system clock (`SI_ClkIn`) and a low-asserted reset signal (`S_Reset_N`) to run. While not fundamentally required, the EJTAG interface facilitates development and testing of the system by enabling users to easily download and debug programs on the MIPSfpga system.

The physical memory map for the MIPSfpga system has two populated blocks, one starting at address `0×0` and the other starting at address `0×1fc00000`, as shown in Fig. 3. Upon reset, the MIPS processor begins fetching instructions at physical address `0×1fc00000`. So, at a minimum, memory at that address must be populated. Typically, instructions starting at that address contain boot code that initialises the system and then jumps to the user code located in the lower memory block, labelled Code/Data RAM in Fig. 3. However, as described in the MIPSfpga GSG, simple programs that do not take advantage of system features such as caching can be placed directly at physical address `0×1fc00000` and run immediately upon reset.

The MIPSfpga system has 1 KB of boot RAM (labelled Reset RAM in Fig. 3) and 256 KB of program RAM (labelled Code/Data RAM) by default. This amount of memory fits well when the MIPSfpga system is targeted to FPGA boards such as the Nexys4

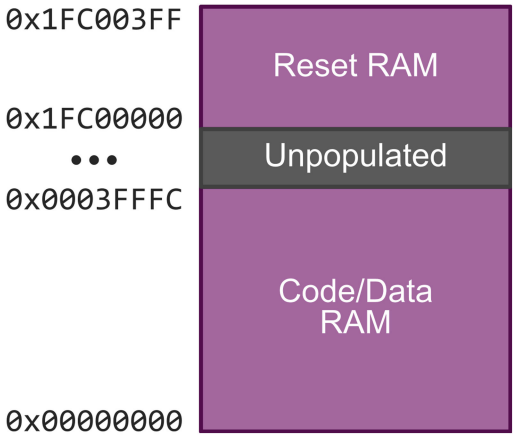

Fig. 3 *MIPSfpga physical memory map*

DDR or DE2-115 FPGA boards. However, the system can also run with different amounts of memory, e.g. smaller amounts of code/data memory when targeting smaller FPGAs (such as Basys3 and DE0), as described in the next section.

Table 1 MIPSfpga hardware requirements

Name	FPGA board options		
	FPGA	Cost	Website
Nexys 4 DDR	Xilinx Artix-7	$159 (academic) $320 (non-academic)	www.digilentinc.com
Basys 3	Xilinx Artix-7	$79 (academic) $149 (non-academic)	www.digilentinc.com
DE2-115	Altera Cyclone IV	$309 (academic) $595 (non-academic)	www.de2-115.terasic.com
DE0	Altera Cyclone III	$81 (academic) $119 (non-academic)	www.de0.terasic.com
Name	MIPSfpga programming probe		
	Cost	Website	
bus blaster	$43.95	www.seeedstudio.com/Bus-Blaster-V3c-for-MIPS-Kit-p-2258.html	

Table 2 MIPSfpga software requirements

Name	CAD tools	
	Description	Website
Xilinx Vivado Webpack	Software for simulation and programming Xilinx-based FPGA boards. (cost: free)	www.xilinx.com/support/download.html
Altera Quartus II Web Edition	Software for simulation and programming Altera-based FPGA boards. (cost: free)	dl.altera.com/?edition=web
Codescape MIPS SDK Essentials	Software development toolkit provided by Imagination Technologies for programming and debugging MIPSfpga. It includes gcc and gdb. (cost: free)	Installer available as part of the MIPSfpga Getting Started package.
OpenOCD	Open Source in-circuit debugger that enables loading and debugging programs on the MIPS core. (cost: free)	Installer available as part of the MIPSfpga Getting Started package.

2.3 Hardware requirements

While MIPSfpga could be explored in simulation only, understanding the MIPSfpga system is most effective when using a combination of simulation and hardware implementation. The hardware required to complete the labs are an FPGA board and a Bus Blaster probe (Table 1 lists details about this hardware).

Among the four FPGA boards shown in Table 1, the Nexys4 DDR [5] and the DE2-115 [6] are the ones used as the example FPGA targets. The other two boards included in the table (Basys3 and DE0), constitute lower-cost alternatives that are, however, large enough to hold the MIPSfpga system. A detailed guide is provided for those who wish to retarget the MIPSfpga system to such smaller boards (actually, the process described may be applied to any other FPGA board as well). The base MIPSfpga system uses 15, 42, 13 and 98% of the logic on the Nexys4 DDR, Basys3, DE2-115 and DE0-CV FPGA boards, respectively. The remaining logic may be used for custom hardware or extensions to the MIPSfpga system.

The Bus Blaster probe, available from SEEED Technology, enables downloading and debugging programs in hardware on the MIPSfpga system, as described in the MIPSfpga GSG. As an alternative to the Bus Blaster probe, programs may also be downloaded using a USB to Universal Asynchronous Receiver-Transmitter (UART) connection such as an existing FPGA programming cable or an FTDI cable. This alternative reduces the hardware requirements and thus lowers the costs for using MIPSfpga. Note, however, that debugging programs in real-time on the MIPSfpga system requires the Bus Blaster probe.

2.4 Software requirements

The software required to simulate and run the MIPSfpga system is: (i) A Windows or Linux-based operating system (OS), (ii) a CAD tool for simulating and loading the MIPSfpga system onto an FPGA, (iii) programming tools and (iv) software for loading and debugging programs on the core, as shown in Table 2. Vivado or Quartus II, respectively, are Xilinx or Altera's CAD tools for loading a hardware design, in this case the MIPSfpga system, onto an FPGA. Imagination Technologies provides the Codescape SDK for compiling and debugging C and MIPS assembly programs. OpenOCD is an open-source in-circuit debugger used with the Bus Blaster probe to load and debug programs on the MIPSfpga system. As an alternative to the Bus Blaster probe, programs can be downloaded onto the MIPSfpga system using the existing programming cable or an inexpensive FTDI cable.

In addition to the main instructions available for a Windows platform, the MIPSfpga GSG shows how to run MIPSfpga under a Linux OS. This extended capability can be useful not only for teachers and students using a Linux OS, but also for MacOS/OS-X users, who can run MIPSfpga on a virtual machine with Linux. The GSG shows how to install and use all of the software and drivers needed to run and use MIPSfpga.

3 MIPSfpga Labs

The second package within the MIPSfpga infrastructure uses the system described in the previous section to teach students computer architecture and SoC design through hands-on learning. Some prior knowledge of digital design, computer architecture, and the MIPS ISA, e.g. the topics taught in [7], is required. As a suggested option, students could also build their own minimally functional MIPS processor using the labs accompanying [7] before completing the exercises found in MIPSfpga Labs. Prior software programming experience is useful, but it can be taught concurrently if necessary.

After studying the digital design and computer architecture topics in [7] and completing the Getting Started package, the students complete 25 labs that guide them through the underlying MIPSfpga setup and increasingly complex interactions with and extensions of the MIPSfpga core and system. The labs guide

students to, first, set up the MIPSfpga hardware and program and debug the MIPS soft-core processor (Part 1 – Introduction, Labs 1–4). They then extend the system to interact with new peripherals (Part 2 – Input/output, Labs 5–13). The third group of labs delves into the microarchitectural details of the 5-stage pipeline of the microAptiv core at the heart of MIPSfpga (Part 3 – Pipeline and Instructions, Labs 14–19). Finally, the memory hierarchy is analysed and modified (Part 4 – The Memory System, Labs 20–25). Table 3 gives a brief description of each lab, and further details are given below.

3.1 Part 1 – Introduction and programming MIPSfpga

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. The availability of the Verilog source files for the MIPSfpga system offers students an unobfuscated view of the design and the ability to modify, extend, and probe inside the MIPS soft-core processor. 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 core running on an FPGA. Moreover, MIPSfpga enables the use of I/O functions such as printf to assist debugging. Lab 4 provides optional exercises for additional programming practice.

3.2 Part 2 – input/output (I/O) in MIPSfpga

Part 2 begins with five memory-mapped I/O exercises (Labs 5–9) for interfacing MIPSfpga with different peripherals. Then, Labs 10–12 analyse advanced I/O topics (interrupts and DMA). Finally, Lab 13 explains how to use the Performance Counters available in microAptiv.

Lab 5 is the first lab showing how to use memory-mapped I/O to add peripherals to MIPSfpga and build an SoC design. It shows how to memory-map signals needed to drive the eight 7-segment displays on the FPGA boards, modify and extend the MIPSfpga hardware to support these memory-mapped addresses, and write C or MIPS assembly code to drive the 7-segment displays.

Labs 6–9 guide the user in adding increasingly complex peripherals: a memory-mapped counter for timing, a buzzer to play

Table 3 MIPSfpga Labs

Part	Lab	Description
part 1 (programming)	1	Vivado or Quartus II Project: Create a project for the MIPSfpga system using Vivado (for the Nexys4 DDR board) or Quartus II (for the DE2-115 board)
	2	C Programming: Learn how to write, compile, debug, and run C programs on the MIPSfpga system
	3	MIPS Assembly Programming: Learn how to write, compile, debug, and run MIPS assembly programs on the MIPSfpga system
	4	More Programming Practice (optional): Write two C programs that implement a pocket hypnotiser and a memory game
part 2 (I/O)	5	Memory-Mapped I/O – 7-Segment Displays: Expand the MIPSfpga system to add access to the eight 7-segment displays using memory-mapped I/O
	6	Memory-Mapped I/O – Counter: Add a memory-mapped millisecond counter to the MIPSfpga system
	7	Memory-Mapped I/O – Buzzer: Add a memory-mapped buzzer to the MIPSfpga system and write a program that plays a song using the buzzer
	8	Memory-Mapped I/O – SPI and LCD: Add a memory- mapped serial peripheral interface (SPI) port to the MIPSfpga system to drive a liquid crystal display (LCD)
	9	SPI Light Sensor: Add a memory-mapped serial peripheral interface (SPI) port to the MIPSfpga system to drive a light sensor
	10	Interrupt-driven I/O: Interact with peripherals using interrupts
	11	Direct-memory-access (DMA): Build a DMA engine to drive interactions between peripherals
	12	DES Encryption with DMA: Build a Data Encryption Standard (DES) encryption engine
	13	Performance Counters: Learn how to configure and use the Performance Counters in microAptiv and test the performance of different programs
part 3 (the core)	14	Basic Instruction Flow through the Pipeline. ADD instruction: Learn how an ADD instruction passes through all the stages of the core and experiment with related arithmetic instructions
	15	Basic Instruction Flow through the Pipeline. AND instruction: Analyse an AND instruction and perform related exercises
	16	Basic Instruction Flow through the Pipeline. LW instruction: Analyse a LW instruction and perform related exercises
	17	Basic Instruction Flow through the Pipeline. BEQ instruction: Analyse a BEQ instruction and perform related exercises
	18	The Hazard Unit: Learn how the hazard logic is implemented in microAptiv
	19	CorExtend. Adding new Instructions to MIPSfpga: Learn how to use the CorExtend interface and employ it for adding several user defined instructions (UDIs) to the MIPS32 ISA
part 4 (memory system)	20	Basic Caching: Introduction to the caches available in the MIPSfpga microprocessor (microAptiv UP)
	21	Cache Structure: Analyse the structure of the data cache in detail and implement and test new configurations
	22	Cache Controller. Hit and Miss analysis: Learn about the theory and practice behind hit and miss management within the cache controller via simulation and exercises
	23	Cache Controller. Management policies: Learn and test the allocation, write, and replacement policies available in microAptiv, and implement new policies
	24	Cache Controller. Store Buffer and Fill Buffer: Learn how these two buffers work and experiment with different access patterns
	25	Scratchpad RAM implementation: Implement an Instruction Scratchpad RAM in microAptiv

music, and two SPI devices (a liquid crystal display and a light sensor). These labs, along with Lab 5, emphasise SoC design and hardware–software codesign. An instructor may choose to include all or a subset of these four labs in the course.

A few extra components are needed to complete Labs 7–9. Table 4 provides some affordable recommendations. Basic components typically already available in a hardware lab (wires, capacitors and breadboards) are also needed.

Lab 10 explains the basic usage of interrupts in MIPS CPUs. The lab also demonstrates how the interrupts can offload the processor from constantly polling I/O ports, which increases the number of cycles available for computation and other non-I/O tasks. The next two labs analyse 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. This valuable resource can be used to test new functionality and find bottlenecks in a system. 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.

3.3 Part 3 – MIPSfpga pipeline and instructions

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 14–17) dive into the implementation of the microAptiv core and its pipeline. Students learn how ADD, AND, Load Word (LW) and Branch if EQual (BEQ) instructions are handled by the pipeline. The labs first introduce the stages of the microAptiv pipeline, showing how the analysed 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 included in the Verilog code (RTL). Finally, students are asked to analyse 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. Two scenarios are analysed: a RAW hazard between arithmetic-logic instructions, where the pipeline does not need to stall, and a RAW hazard between a LW instruction and a subsequent arithmetic-logic instruction, where a 1-cycle bubble is necessary between the load and the dependent instruction. 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 behaviour in real time by connecting system signals, e.g. pipeline, hazard control or cache eviction signals, to LEDs.

The final lab (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 specialised 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, and the interaction of the UDI unit with the microAptiv pipeline. The lab also shows how to use CorExtend by providing an example instruction implemented with this interface: SELEQZ, a new instruction introduced in MIPS32 ISA R6. Finally, students are asked to implement their own repertoire of UDIs, with increasing complexity. Students begin by implementing simple logical instructions, NAND and SEQ (an instruction that is similar to SLT). The advanced exercises implement more complex instructions that require a floating-point unit (FPU) and a digital signal processing (DSP) unit.

Table 4 MIPSfpga Labs extra hardware requirements

Name	Cost	Website
buzzer (Lab 7)	$2	www.digikey.com/product-search/en/audio-products/buzzers/720967?k=102-1153-ND
LCD (Lab 8)	$16	www.digikey.com/product-search/en?keywords=1481-1063-ND
light sensor (Lab 9)	$10	http://store.digilentinc.com/pmod-als-ambient-light-sensor/

3.4 Part 4 – MIPSfpga memory system

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) and by describing the cache structure (Lab 21). Lab 20 is an overview lab that uses the FPGA board's LEDs to analyse the number of hits and misses in several example programs. Lab 21 analyses the various cache arrays that make up the cache used by microAptiv: the Data Array, which stores the instructions/data, the Tag Array, which holds part of the physical addresses corresponding to the instructions/data stored in the Data Array, and the Way Select Array, which holds information about whether cache blocks have been written (i.e. whether they are dirty) and replacement policy state of the cache blocks, also called cache lines. The lab describes both the array interfaces and their internal implementation. After these detailed explanations, the students are asked to implement and test new cache configurations and to test several code optimisation techniques using the performance counters.

The next three labs (22–24) delve into the cache controller. Lab 22 analyses the management of cache hits and misses. The lab describes the main stages, structures, and signals involved in a cache hit/miss. Then, several simulations are provided to illustrate the described concepts. Finally, proposed exercises explore the cache system by, e.g. 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 the different allocation and write policies available and also to implement a First-In First-Out (FIFO) replacement policy sometimes used in embedded processors. Finally, Lab 24 explains the operation of two important structures included in the data cache controller: 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. The Store Buffer removes writing to the cache from the critical path, allowing store instructions to be non-blocking. Similarly, the Fill Buffer removes block filling after a miss from the critical path, allowing the pipeline to resume execution before the missed line has been copied into the cache.

The final lab (Lab 25) shows how to add an Instruction Scratchpad RAM to MIPSfpga. A Scratchpad RAM is managed by the programmer or through compiler support, as opposed to the cache memory which is transparent to the user, and is usually aimed at storing critical blocks of code that need to be retrieved with a small and predictable latency. The basic MIPSfpga configuration includes an interface (I/D-SRAM Interface in Fig. 1) but no actual Scratchpad RAM. This lab shows how to add the Scratchpad RAM block and how to communicate with it through the I/D cache controller.

4 MIPSfpga-SoC

The final package in the MIPSfpga materials is the MIPSfpga-SoC package [8], which shows how to extend MIPSfpga to build a Linux SoC system and load the open source OS. This section provides an overview of Linux itself and the hardware requirements to run it and then describes the system-level design practices that enable the rapid development of an SoC, centred around MIPSfpga, capable of running Linux. This section also describes the software port of the Linux kernel. MIPSfpga-SoC targets Digilent's Nexys4 DDR board and uses Xilinx's Vivado IP Integrator. The MIPSfpga-SoC package gives a detailed view of how the SoC in embedded systems is designed and built up in layers to run complex software.

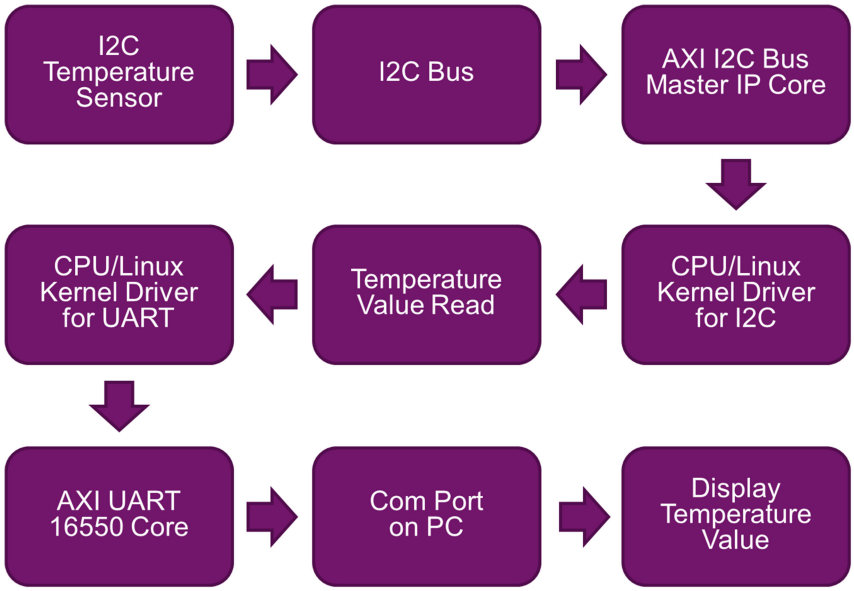

Fig. 4 *Simplified flow of data when reading from a temperature sensor*

4.1 Linux

Linux is one of the most popular and scalable open source OSs used in embedded systems. Linux can be divided into two parts: the Linux Kernel and the Linux Userspace. The Kernel interacts directly with hardware and provides a layer of abstraction. The Userspace interacts with the hardware via standardised system calls provided by the Linux Kernel. Due to the abstraction by the Linux Kernel, many different types of Linux Userspace can run on the same hardware. MIPSfpga-SoC uses Buildroot, one of the most scalable Linux Userspace flavours.

Table 5 Soft-SoC requirements for running Linux

Minimum requirements for Linux			
Linux requirement	Soft-SoC IP block	Type	Comments
CPU with MMU	MIPS microAptiv	Synthesised hardware	Wired internally
Interrupt controller	MIPS microAptiv	Synthesised hardware	Wired internally
Timer interrupt	MIPS microAptiv	Synthesised hardware	Wired internally
UART	Xilinx AXI UART16550	Synthesised hardware + FTDI Chip on Nexys4 DDR	Wired to USB-Serial chip on Nexys4DDR for connection to PC
Memory	Xilinx Memory Interface Generator (MIG)	Synthesised hardware + DDR Chip on Nexys4 DDR	DDR controller wired to DDR2 chips on Nexys4 DDR
e-JTAG	MIPS microAptiv	Synthesised hardware	JTAG wired to external probe such as BusBlaster
Extra functionality			
Ethernet for connecting to Internet	Xilinx Ethernet MAC	Synthesised hardware + PHY on Nexys4 DDR	MAC and PHY wired to RJ45 Ethernet port on Nexys4 DDR
GPIO for switches/LEDs	Xilinx GPIO controller and Custom GPIO controller	Synthesised hardware	GPIO controllers I/O wired to switches/leds on Nexys4 DDR
I2C for temperature sensor	Xilinx I2C bus master	Synthesised hardware for bus master	I2C bus connected to temperature sensor chip on Nexys4 DDR

4.2 SoC building blocks

A Linux kernel can be implemented with the minimum hardware support listed in Table 5. The table also lists extra features added for increased functionality. To run Linux, the system must have a processor with an MMU, interrupt controller, timer interrupts, UART, memory, and an EJTAG interface. Physical memory in embedded systems can be addressable in a fragmented manner. The MMU is necessary to provide a consistent virtual memory map to the CPU which then maps onto the physical memory in the hardware. The interrupt controller is needed to receive interrupts from multiple peripherals in the system. The timer interrupt provides a consistent clock to Linux to schedule various processes running in the OS. UART and e-JTAG interfaces are the primary tools used in embedded systems for both debug and to access the console. The Ethernet, GPIO, and I2C temperature sensor interfaces are added for extended functionality.

MIPSfpga-SoC uses hardware, Xilinx IP (synthesised hardware), and software to complete the connection between the MIPS core and its peripherals. An example can be seen in Fig. 4. The temperature sensor is a physical chip on the Nexys4 DDR board that connects to the MIPS core through a Xilinx I2C bus to the AXI I2C Bus Master IP core. The kernel software driver initialises and configures the I2C bus master IP block to read the temperature value. Another Linux kernel driver controls the console display via the AXI UART 16550 IP core. The console is accessible via a com port on the PC. The temperature value is written on that port for the user to see. The MIPS core together with the Xilinx IP becomes the SoC when the design is either targeted to an FPGA, as in this case, or fabricated on a chip.

4.3 System-level design

The Linux SoC is built with the MIPS core as the master controlling peripherals, acting as slaves, across the AHB-Lite bus. The slaves are implemented mostly using Xilinx IP blocks. A custom GPIO module is also added to demonstrate custom peripheral integration. Using existing IP blocks, in this case supplied by Xilinx, 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 MIPS core with the Xilinx-supplied IP.

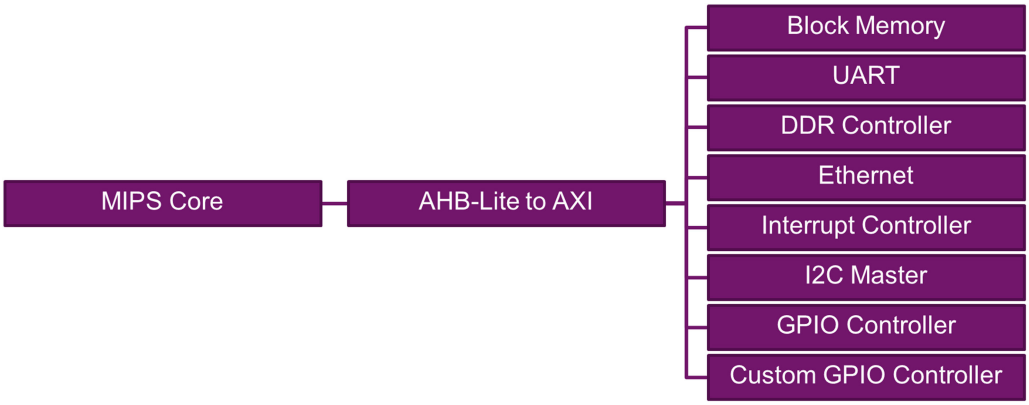

Fig. 5 *Simplified block diagram for MIPSfpga-SoC*

Fig. 5 shows a block diagram of the system. The MIPS core connects to the peripherals through the AHB-Lite to AXI bridge and AXI interface. The AXI interconnect allows the MIPS master to connect to several slaves using memory-mapped I/O. The slave peripherals are block ram controllers, Ethernet MAC, AXI Interrupt controller, an AXI GPIO, UART16550, an AXI IIC master, a Memory Interface Generator which serves as the DDR controller, and a Custom GPIO Controller supplied by Imagination. All of the blocks except the MIPS core and the custom GPIO module are Xilinx IP blocks. This design fulfils the requirements listed in Table 5 to run Linux.

4.4 Kernel and Buildroot port

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 kernel running on the MIPSfpga-SoC. The resulting system can run applications relying on the GNU standard C libraries.

4.5 MIPSfpga boot process

The Linux Kernel expects hardware to be in an initialised state before being loaded. In traditional systems, this is accomplished by the first stage bootloaders. The first stage bootloaders are stored in static ROM on the SoC. They initialise the CPU cache and other peripherals. They load the secondary stage bootloader or kernel in memory from an attached storage medium and then set the program counter register to the start address of the kernel to execute it.

In MIPSfpga, the first stage bootloader is similarly preloaded in block memory as part of the bitstream that loads the MIPS SoC onto the FPGA. However, the kernel is not read from any storage medium. The kernel is loaded via e-JTAG. After the bootloader initialises the system, the CPU is halted via e-JTAG and the kernel is loaded in memory.

4.6 Custom blocks versus industry-supplied IP

MIPSfpga-SoC additionally bridges the gap between academia and industry by showing how to use both custom blocks and industry-supplied IP, in this case Xilinx IP, in a design. Custom design enables students to understand the entire system they are building from the ground up; whereas using industry-supplied blocks decreases development time. To facilitate practising both methods of design, MIPSfpga-SoC uses both types of blocks, notably adding a custom GPIO block with an AXI interface. This simple block 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.

5 Related work

Many soft-core processors, in addition to MIPSfpga, are available today. The most notable is the soft-cores offered by the major FPGA providers, Xilinx and Altera, and existing open-source soft-cores. This section describes these other soft-core options and compares them with MIPSfpga.

Xilinx and Altera offer their own soft-cores, Nios/Nios II [9] and MicroBlaze [10], 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 [11], which is a basic (8 K 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 only limited teaching materials.

Several open-source soft-cores also exist. Two well-known alternatives, both supporting a SPARC RISC ISA, are the OpenSPARC family [12] and the LEON family [13], developed by Oracle (and originally by Sun Microsystems) and by Aeroflex Gaisler (and originally the European Space Agency), respectively. Although they are interesting open-source alternatives, they lack good teaching materials and they implement an ISA not as widely used in academia as MIPS or ARM. Two other open-source alternatives worth mentioning are RISC-V [14], started at the University of California, Berkeley, and openRISC, developed by opencores.org [15]. 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 are available in five languages. MIPSfpga also provides support across FPGA platforms, including both Xilinx and Altera FPGAs, and it is easily extendible to other FPGAs. The system has also been extended with an advanced package, MIPSfpga-SoC, which includes a Linux SoC design centred on the MIPS core with interfaces such as memory (DDR), UART16550, I2C, Ethernet, and an interrupt controller.

Other teaching packages for computer architecture are also available, including the HIP environment [16], the BZK.SAU simulator [17], the CNP laboratory [18] and the advanced multi-core architectures (AMA) course [19], among others. These packages are discussed and compared with MIPSfpga here.

The work in [16] 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. Instead, the MIPSfpga teaching resources, which include guides and labs, are available in five different languages. 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 [17] introduce a computer architecture simulator called BZK.SAU. Similar to the work shown previously, the ISA of this simulator consists of 59 instructions. Simple cores such as these allow for 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. In addition, the latter work is exclusively based on simulation, meaning that students cannot download the completed design onto an FPGA and experiment with it works on actual hardware.

The CNP laboratory presented in [18] 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 minimised MIPS ISA, and the compiler that goes along with it, Tiny C, is designed to comply with 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.

Other simulators suitable for computer architecture courses are compared in [20]. An interesting aspect of this survey is that the evaluation criteria are established using the IEEE Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering [21]. These guidelines establish six main knowledge units: Fundamentals of computer architecture, Memory system organisation and architecture, Interfacing and Communication, Device subsystems, Processor systems design, and Organisation of the CPU. As shown in Table 3, all of these topics are covered to some extent by the MIPSfpga Labs.

The authors in [19] introduce a course on AMA. This type of courses is beyond the initial scope of MIPSfpga but, as we stated previously, lecturers have the freedom to adapt the contents of the Getting Started package to their own needs. As an example, the authors in [22] 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.

For some research purposes, fabricating the MIPS core in silicon is useful. For these cases, 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-classes are superset extensions 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 an FPU, DSP, microMIPS Instruction set and Hardware Virtualisation for enhanced security. These agreements position MIPSfpga as a one-of-a-kind collection of teaching resources, providing materials from the first courses in Computer Architecture up to advanced architectural topics for masters courses and a route to silicon for researchers.

6 Conclusions

MIPSfpga offers a transparent platform for learning principles of computer architecture, digital design, SoC design, and hardware–software codesign. The two packages, MIPSfpga Getting Started and MIPSfpga Labs, available to academics by Imagination Technologies [1], offer an introduction to MIPSfpga and a set of labs to teach the above principles. In the process, students learn how to use FPGAs and peripherals and how to work with a commercial processor, deeply analysing its microarchitecture and its memory system. For instructors who want to build on this knowledge, a third package, MIPSfpga-SoC, also available from Imagination Technologies [1], shows how to run Linux on the MIPSfpga core and how to integrate industry-supplied interfaces into the system.

Once the students complete studying all the materials included in these packages, they are ready to develop more ambitious projects, such as adding interfaces to drive additional peripherals (e.g. I^2C or UART), adding new features to the core (such as a hardware prefetcher) or to the memory system (such as a second cache level or a way predictor), or any other project that the instructor may choose.

7 Acknowledgments

The authors acknowledge the contributions from the Imagination University Program, the University of Nevada, Las Vegas (USA), Imperial College London (UK), 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 (University Complutense of Madrid), Tejaswini Angel (Portland State University), Christian White, Gibson Fahnestock, Jason Wong, Cathal McCabe (Xilinx), and Larissa Swanland (Digilent).

8 References

[1] 'Imagination University Program – Resources', https://community.imgtec.com/university/resources, accessed February 2017
[2] Imagination Technologies Ltd.: 'MIPS32 microAptiv™ UP Processor Core Family Datasheet', 31 July 2013
[3] ARM: 'AMBA 3 AHB-Lite Protocol Specification', 2006
[4] Harris, S., Owen, R., Sedano, E., *et al.*: 'MIPSfpga: hands-on learning on a commercial soft-core'. 11th European Workshop on Microelectronics Education (EWME), Southampton, England, May 2016, pp. 1–5
[5] Digilent Inc.: 'Nexys4 DDR™ FPGA Board Reference Manual', 11 September 2014
[6] Terasic Inc.: 'DE2-115 User Manual', 2013
[7] Harris, D., Harris, S.: '*Digital design and computer architecture*' (Elsevier Science and Technology, 2007, 2nd edn. 2012)
[8] Kakakhel, Z., Harris, S., Harris, D.: 'MIPSfpga: an unobfuscated commercial MIPS core and SoC that runs Linux". Embedded World 2016, Nuremberg, Germany, February 2016
[9] 'Altera – NIOS-II Processor', https://www.altera.com/products/processors/overview.html, accessed February 2017
[10] 'Xilinx – MicroBlaze Soft Processor Core', http://www.xilinx.com/products/design-tools/microblaze.html, accessed February 2017
[11] 'ARM – Cortex M0 Design Start', http://www.arm.com/products/designstart/index.php, accessed February 2017
[12] 'Oracle – OpenSPARC', http://www.oracle.com/technetwork/systems/opensparc/index.html, accessed February 2017
[13] 'Aeroflex Gaisler – LEON series Softcores', http://www.gaisler.com/, accessed February 2017
[14] Waterman, A., Lee, Y., Patterson, D.A., *et al.*: 'The RISC-V Instruction Set Manual, Volume I: User-Level ISA', version 2.0, 2014
[15] 'OpenCores – OpenRISC', http://opencores.org/or1k/Main_Page, accessed February 2017
[16] Bulić, P., Guštin, V., Šonc, D., *et al.*: 'An FPGA-based integrated environment for computer architecture', *Comput. Appl. Eng. Educ.*, 2013, **21**, (1), pp. 26–35
[17] 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
[18] Abe, K., Tateoka, T., Suzuki, M., *et al.*: 'An integrated laboratory for processor organization, compiler design and computer networking', *IEEE Trans. Educ.*, 2004, **47**, (3), pp. 311–320
[19] Petit, S., Sahuquillo, J., Gómez, M.E., *et al.*: 'A research-oriented course on advanced multicore architecture: contents and active learning methodologies', *J. Parallel Distrib. Comput.*, 2017
[20] Nikolic, B., Radivojevic, Z., Djordjevic, J., *et al.*: 'A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization', *IEEE Trans. Educ.*, 2009, **52**, (4), pp. 449–458

[21] The Joint Task Force on Computing Curricula, IEEE Computer Society/ ACM: '*Computer engineering 2004 – curriculum guidelines for undergraduate degree programs in computer engineering*' (IEEE Computer Society, 2004)

[22] Kumar, H.B.C., Ravi, P., Modi, G.*, et al.*: '120-core microAptiv MIPS Overlay for the Terasic DE5-NET FPGA board'. Int. Symp. on Field-Programmable Gate Arrays, Monterey, USA, February 2017