RECKON – A Reconfigurable Prototyping Kit for Engineering and IT Laboratory Based Courses

Eduardo Bezerra [1] <eduardob@acm.org>, **Marianne Pouchet** [2] <M.Pouchet@sussex.ac.uk>, **Elias Stipidis** [2] <E. Stipidis@sussex.ac.uk>, **Fernando Moraes** [3] <moraes@inf.pucrs.br>, **Ney Calazans** [3] <calazans@inf.pucrs.br>, **Augusto Einsfeldt** [4] <aee@terra.com.br>

[1] Space Science Centre, School of EIT, University of Sussex, England, UK
[2] Communications Research Group, School of EIT, University of Sussex, England, UK
[1, 3] GAPH Group, Informatics Faculty, Catholic University (PUCRS), Porto Alegre, Brazil
[4] AEE Engenharia Eletronica, Joinvile, Brazil

Abstract

This work describes an educational kit developed at the University of Sussex, UK. The kit is based on reconfigurable computing technology, targeting its use in different laboratory based courses, with minimal modifications of the hardware components. The paper describes the software and hardware modules of the system, and the main stages of the project.

I. Introduction

Putting into practice the concepts taught in technological courses is an essential complement to a good understanding of the theory. The constant changes and advances in technology, result in challenges that universities and technical colleges have to deal with to keep their laboratories updated. The managers of modern courses in this highly competitive market have to find ways to provide students and tutors with appropriate tools, taking into consideration the usually modest available budget. A prototyping kit based on reconfigurable computing technology is a suggestion for the solution of this problem. This solution has been implemented at the University of Sussex, UK, and is used in one module of the Data Communications course for MSc. students at the School of Engineering and Information Technology. The implementation of the project has become feasible due to the falling costs of large Field Programmable Gate-Arrays (FPGAs) devices, which are the most widely used components in the design of reconfigurable computers.

The students use a tool (user interface), developed according to the course requirements, to observe and to control the execution of their experiments on the prototyping boards. It is important to highlight here that, depending on the course, the use of reconfigurable computing technology is completely transparent to the students. For example, in the Data Communications module the goal is to teach protocols and standards for instance RS-232 and RS-485, and to show how to program and use a traditional USART. In this case the students do not even have to know about the existence of the FPGA on the prototyping board. In other situations as in a "VHDL for synthesis" course or in a hardware/software co-design course, additional tools such as commercial synthesis can be used by the students to generate configurations for the FPGA.

The main objective of this paper is to introduce to the Reconfigurable Computing community some of the features of the prototyping kit developed at Sussex. The paper is organised as follows: Section II briefly describes a laboratory based course of a Brazilian university and a similar one taught in a British university; in Section III there is a description of the system architecture; Sections IV and V describe the main components of the system;

Section VI presents the case study; and Section VII discusses conclusions and future directions.

II. Laboratory Based Courses

In order to obtain a better understanding of the motivation for the development of *Reckon*, laboratory based courses of a Brazilian and a British University are introduced in this Section.

In the Computer Organization course of the Brazilian University [1], both lecture and laboratory courses start with a traditional, schematic-based approach to processor core design. Later, the whole design work is redone with modern, HDL-based tools. There are three main reasons for doing this. Firstly, to show that it is possible to design at high levels of abstraction and still obtain competitive implementations, due to improvements in quality of current CAD tools. Secondly, to provide students with a comprehensive insight into the panoply of hardware design methods and tools, by comparing these two significantly distinct approaches. Thirdly to make it clear to students that HDLs are not programming languages, which is achieved by the mapping of schematic symbols and structures to the syntactic and semantic structures of the chosen HDL. This last reason makes instructors insist to students that they must "think hardware" when using HDLs, even if the language they use allows them to view hardware as software. The Computer Organization Laboratory course is divided into three units. In Unit 1, entitled Classical Digital Circuit Design, students review basic concepts acquired in the previous Digital Circuits course, working with schematic capture and simulation tools. They implement basic combinational and sequential blocks such as adders, ALUs, counters and finite state machines. Next, in Unit 2, Classical CPU Design, they employ the same set of tools, to implement the case study of the lecture course, step by step, using RTL schematic modules. Finally, in Unit 3 - HDL Hardware Design - an HDL language is introduced and used to describe hardware at the behavioural and structural levels, contrasting the complexity of the circuits they can handle with that of the previous approach. The target HDL chosen is VHDL. The lab course comprises around 15 2-hour practical sessions. Examples are available for the students, from simple ALUs up to small load-store processors. Both for their work with schematic capture and with VHDL synthesis, they have to implement their circuits on available prototyping boards (XS40/Xstend). The use of *Reckon* will introduce more flexibility for the lecturer in defining the case studies, and it will provide the students with higher levels of observability and controllability when running their experiments.

In the British university, the *Reckon* system is used in the Data Communications module to demonstrate to students different serial communication protocols. The *Reckon* motherboard is connected to a daughter board consisting of an 8251a USART and RS232/RS422 drivers and receivers. In a sequence of three laboratories, issues involved in the data-link layer of serial data transmission are demonstrated allowing students to choose design settings and observe results. Transmission and reception can be conducted either on the same *Reckon* system or by using two separate systems – one as transmitter and the other as receiver. RS232, RS422 and Manchester Encoding are all used as physical layer protocols throughout the labs, and a simple Java GUI is used as well, for user interaction and to send simple control commands to the hardware.

In the first laboratory, the asynchronous transmission of characters and frames is practically demonstrated and the concepts of baud-rate, parity and Block Sum Check are introduced. Here the FPGA receives the appropriate user-defined USART settings from the GUI and is responsible for the control and timing of the USART, serial to parallel conversions, block sum check calculations and baud rate generation. The second lab introduces the concept of synchronous data transmission. Here the FPGA has the added

functionality of Manchester Encoding/Decoding and the generation of CRC error detection codes in order to further demonstrate to students the issues involved in serial data transmission. Finally, Data Link Protocols are introduced in the third lab. Idle RQ is again added to the FPGA functionality using a user controlled timing procedure to clearly display the step-by-step execution of this protocol. The advantages of the *Reckon* system over the previous laboratory kit used in this course are numerous. Firstly, the reduction in complexity and size obtained in the use of a single FPGA as opposed to discrete hardware components is phenomenal. Secondly the flexibility of the reconfigurable FPGA allows for easy expandability should the course co-ordinator decide to add further modules (additional protocols perhaps) or adjust existing ones. The Java interface as well allows the execution of the laboratories to be platform-independent and even potentially web-based for optional distance use.

III. Prototyping Kit Architecture

The kit comprises of four main components: the control board; the motherboard; the daughter board; and the user interface. As shown in Figure 1, students use the software module (user interface) to set up the mother and daughter boards, and to control the execution of their laboratory assignments. In the current version of the system, the host computer is connected to the prototyping platform via serial port. The use of a USB port is planned for future versions.

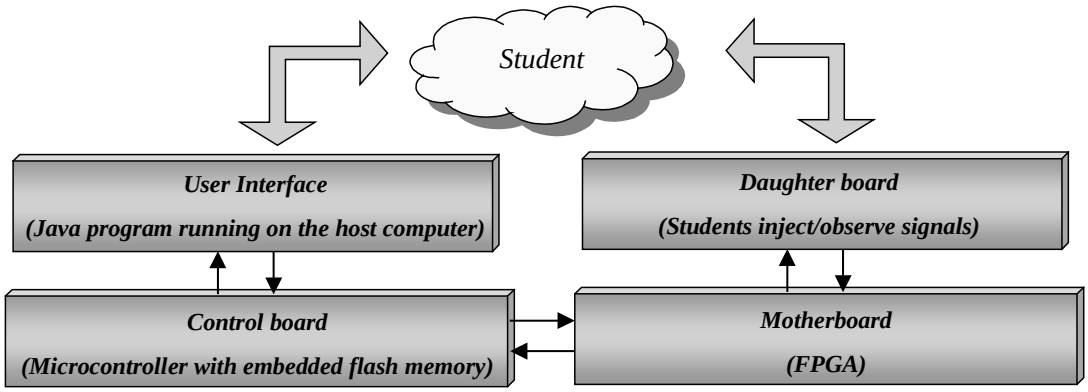

Fig. 1. Block diagram of *Reckon*

The **control board** consists, basically, of a microcontroller [2] with embedded flash memory and serial port interfaces. The microcontroller runs an embedded operating system used to configure the FPGA and to manage the communication between the FPGA and the host computer. The software module running on the host computer is designed to use the services provided by the operating system.

The **motherboard** consists of a Xilinx Virtex FPGA [3], an oscillator, and a power regulator module responsible for supplying the power for all boards. There is a Finite State Machine (FSM) running on the FPGA. The main function of the FSM is the provision of the interface between the daughter board and the control board (and consequently, the user interface). Using an external oscillator chip in the motherboard to generate a clock signal to the circuit implemented within the FPGA, provides the flexibility of allowing different oscillators with different clock frequencies to be used for different applications.

The **daughter board** is where the students run their experiments. Although some courses may require more than one daughter board, usually one daughter board per course/kit is sufficient. Some courses may not need daughter boards at all. The user interface is designed in order to generate signals that can be observed from the daughter board output pins. The system as a whole is designed in modules in order to facilitate its adaptation to different

courses. From the hardware point of view (i.e. chips and boards), only a new daughter board may be necessary for a new course. Figure 2 shows the three boards components of *Reckon*.

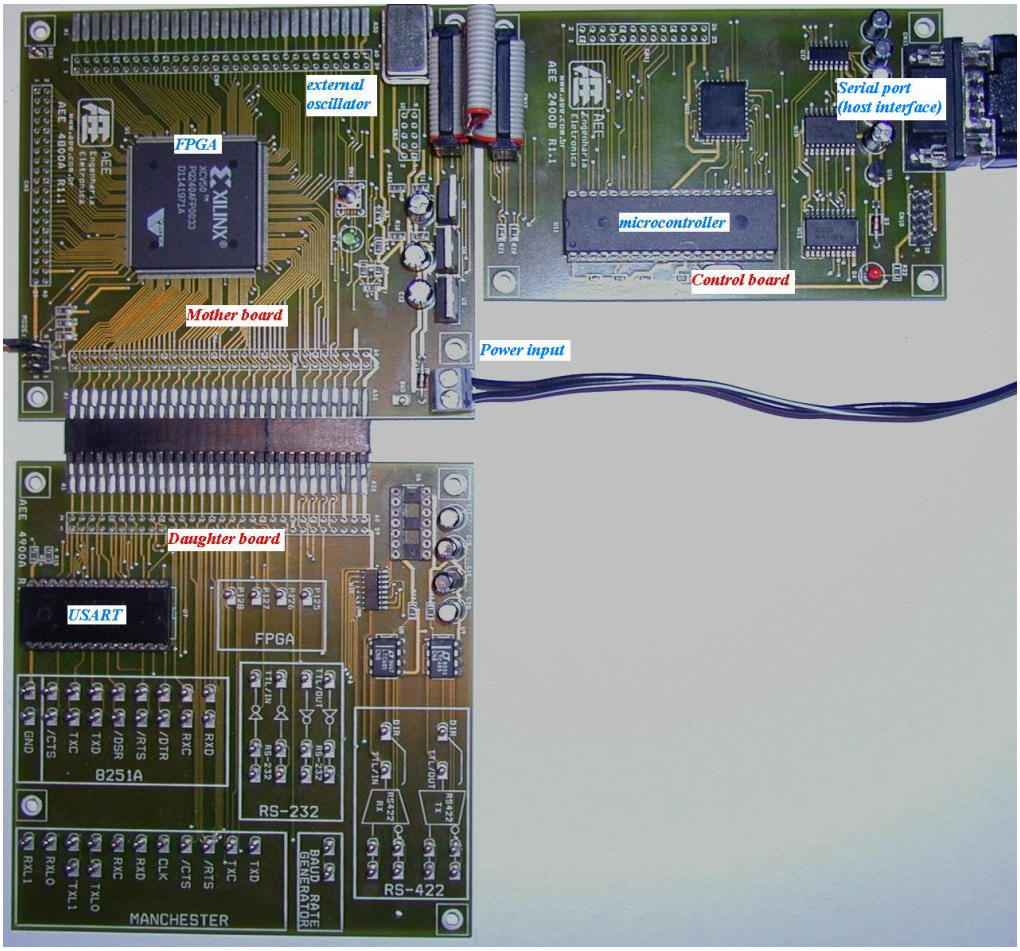

Fig. 2. *Reckon:* control board, motherboard and daughter board.

IV. The Software Module (Host Computer)

The user interface has been written in Java [9] in order to run on different platforms. As shown in Figure 1, the user interface is connected to the control board by means of the Java Communications API *commapi* [4]. This Sun's API has been designed to help in the development of platform-independent communications applications, but at the time this paper was written there were versions available only for MS-Windows and Solaris.

In order to use *commapi*, the software module (host computer) creates an object to manage the serial communication: *connection = new SerialConnection(mainframe, parameters, systemStatus);*. The next step is to open the serial port in the host computer where the Control board is connected: *connection.openConnection();*. After that the prototyping kit is accessed by using the *sendByte* method from *commapi*. The embedded operating system running on the Control board has a set of services used by the user interface. For example, as shown in Table 1, in order to obtain the version of the prototyping kit and the amount of available memory on the microcontroller, the pair of commands used is: *connection.sendByte(0x40);* and *connection.sendByte(0x55);*. Service 0x47 provides a way for the host computer to exchange data with the FPGA. Considering the daughter board used as a case study in this work (Section VI), in order to reset the 8251A USART, the sequence of commands is *connection.sendByte(0x47); connection.sendByte(0x20);* and *connection.sendByte(0x00);*. In this example, service 0x47 sends to the FPGA the bytes 0x20

and 0x00, and the FSM running on the FPGA understands that it is the command to reset the USART chip on the daughter board.

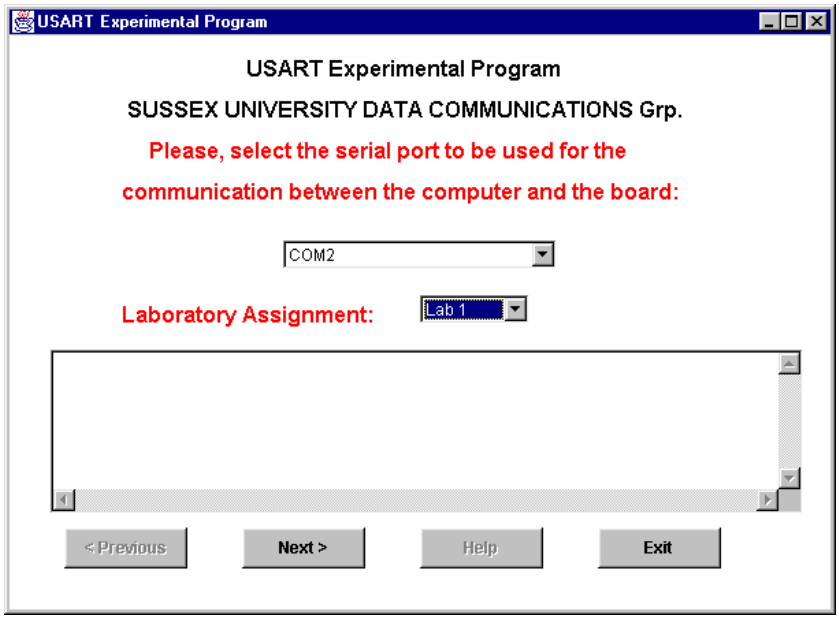

Fig. 3. GUI of the software module

The Graphical Unit Interface (GUI) of the software module has been designed to be as simple as possible in order to keep the students' attention on the subject of the laboratory session. As shown in Figure 3, the students select few parameters per screen and press "Next" to proceed to the next screen. After three or four screens, depending on the experiment, the students are able to observe and compare the obtained results against the expected ones.

V. The Control board and the Motherboard

In the original design there was no need for the Control board. The idea was to have a microcontroller core embedded into the FPGA, interfacing the prototyping boards to the host computer. However, a reliable and inexpensive microcontroller core has not been found and, in order to keep the motherboard as simple as possible, an FPGA configuring board with a PIC microcontroller has been acquired from the Brazilian company AEE Engenharia Eletronica. The embedded operating system running on the PIC microcontroller provides the services listed in Table 1.

Table 1. List of services of the embedded operating system running on the Control board.

Command	Description	Data	Response
0x40	Read version and memory size	55	13 bytes + 0x0D + 0x0A
0x41	Set 32 bits WR address	4 bytes, MSB first	address 4 bytes, MSB first
0x42	Erase sector (flash memory)	Sector (30 to 3F)	'E': erased; 'F': fail
0x43	Command	30: disable FPGA 31: enable FPGA	'0': success '1'..'9': fail
0x45	Write byte and inc. address	Value	Value complemented
0x46	Read byte and inc. address	52	Data read
0x47	RD/WR 2 bytes from/to FPGA	2 bytes, MSB first	2 bytes read, MSB first

Once the FPGA configuration bitstream has been sent to the Control board, the microcontroller stores it into the embedded flash memory. Every time the system is powered up, the microcontroller (Control board) uses the stored bitstream to configure the FPGA (motherboard). The services listed in Table 1 can be used to send a new configuration bitstream to the microcontroller, to read a stored bitstream from the flash memory, and to send/receive bytes to/from the FPGA. The embedded operating system in use has been written in assembly language, and the FSM for the FPGA has been designed using schematic diagrams. The versions in use of the operating system and the FSM have been implemented by AEE. The version of the FSM developed at Sussex was written in VHDL [7][8], but it is not in use because of scheduling problems (see Figure 4 for project details).

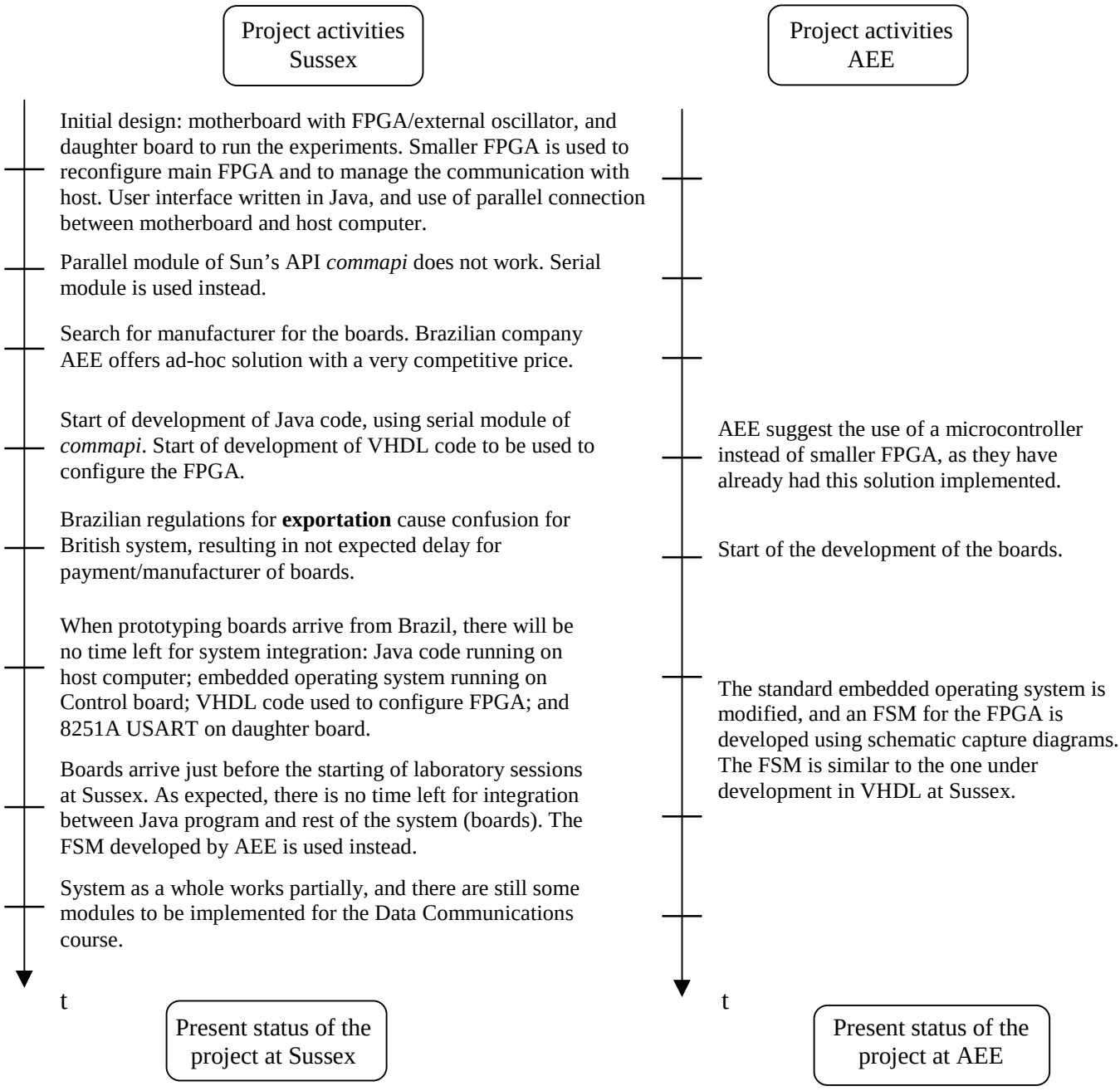

Fig. 4. The stages of the project

VI. Case Study: Targeting the Data Communications course

As an example of the functionality of the system, for the Data Communications course (British university) the daughter board has an 8251A USART [5], RS-232/RS-422 converters and several test pins, as shown in Figure 2. The students can use test and protocol analyser equipment to verify the functionality of their designs. As stated before, there is no need for an 8251A chip in the daughter board. A USART core could have been implemented in the FPGA itself, but at the time the system was designed such cores were not full compatible with the 8251A USART at an affordable cost. In the Sussex version, the FPGA has three main components, all of them implemented in VHDL: a manager kernel, a baud rate generator and a Manchester encoder/decoder. The kernel is responsible for the control of all the components, and also for programming and controlling the USART in the daughter board. The students use the Java program on the host computer to program the USART and to send/receive characters and complete frames either synchronous or asynchronous.
In the AEE version, the FSM running on the FPGA programs the USART, generates the baud-rate and controls the exchange of bytes between the daughter board and the host computer. A GUI has to be designed for each new course. The first screen of the GUI for the Data Communications course is shown in Figure 3. Figure 4 shows the different stages of the project.

VII. Final Remarks

This work briefly describes an educational kit based on reconfigurable computing technology. The kit has been designed for its reusability in different laboratory based courses. Just to exemplify the flexibility of the kit, in the case of a microprocessor or computer organisation course, with minimal effort, a new user interface can be implemented by reusing the existing objects. A microprocessor core could be used in the FPGA, and the daughter board would have, mainly, test pins. The possibility of reusability of Intellectual Property (IP) cores and Java objects is an important point for the success of this project.

The biggest advantage of this system, from a financial point of view is that the daughter board, in most of the cases, is a very simple and, consequently, inexpensive component. In case of changes or updates to course content, there will be no need for further investments in extra hardware. Only new daughter boards may be necessary, with an upgrade in the FPGA configuration and in the user interface.

Considering the low cost of the system, about €250.00 each, some students may decide to buy the kit. The configuration file and the user interface will be available on the Lecturer's web site for students to download and use at home. An example of improvement in this project is the use of a core of a microcontroller on the FPGA, avoiding the need for the Control board. This will reduce the final cost of the kit.

Acknowledgements

This research is mainly supported and funded by the Teaching and Learning Development Unit and the School of EIT, University of Sussex, UK. This research is also supported by CNPq – Brazilian Council for the Development of Science and Technology, and PUCRS – Pontific Catholic University of Rio Grande do Sul, Brazil.

References

[1] Calazans, N. L. V.; and Moraes, F. G. “Integrating the Teaching of Computer Organization and Architecture with Digital Hardware Design Early in Undergraduate Courses”, IEEE Transactions on Education, vol. 44, no. 2, May 2001, pp. 109-119.

[2] Microchip Technology Inc. “PIC 16F87X databook”, Microchip, USA, 1999, 200p.

[3] Xilinx “The Programmable Logic Data Book”, Xilinx, San Jose, 1999.

[4] Sun Microsystems “Java™ Communications API – *commapi”* Java Developer Connection (JDC) web-site <http://developer.java.sun.com>, 2001.

[5] Intel Corporation “Microprocessor and Peripheral Handbook”. Intel, Santa Clara, 1987.

[6] Electronics Industry Association (EIA) "Interface between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange", EIA, Washington, D.C., 1985.

[7] Xilinx “Synthesis and Simulation Design Guide”. Xilinx, 1998, 314p.

[8] Synplicity “Synplify Better Synthesis – User Guide release 5.0” Synplicity, 1998.

[9] Sun Microsystems, “Java™ 2 SDK, Standard Edition Documentation”, Sun, 2001. <http://java.sun.com/j2se/1.3/docs/index.html>