*Int. J. Engng Ed.* Vol. 24, No. 4, pp. 833–842, 2008
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 2008 TEMPUS Publications.

# A Teaching Strategy for Developing Application Specific Architectures for FPGAs*

JOÃO M. P. CARDOSO
*Universidade Técnica de Lisboa (UTL), Instituto Superior Técnico (IST), Department of Informatics Engineering, INESC-ID, Lisboa, Portugal. E-mail: jmpc@acm.org*

*This paper presents an approach to teaching design of non-programmable application-specific architectures using VHDL, logic and physical synthesis tools and FPGAs. The approach relies on mini-projects that resemble typical problems that students may face in real-life concerning the design of application-specific architectures. The teaching approach presented in this paper supports the incremental learning of both VHDL and the tools used, as the projects are being developed, i.e., students are motivated to acquire skills at the pace at which those skills are required to advance project development. The results so far are very encouraging. Even students with little knowledge of hardware design and embedded systems have succeeded in their assignments. Feedback obtained from students reveals the suitability of certain aspects of the approach and the major difficulties they have faced.*

**Keywords:** FPGAs; VHDL; application-specific architectures; digital systems; education

## INTRODUCTION

APPLICATION SPECIFIC ARCHITECTURES can be used to achieve certain requirements (e.g., performance demands, energy savings), unlikely to be otherwise met. Therefore, skills on how to design application specific architectures are becoming very important. Years ago, the design of those architectures was mainly carried out by hardware experts with deep understanding of low-level details. Those architectures were mostly implemented as ASICs (Application-Specific Integrated Circuits) [1], whose design efforts and skills required were typically mastered by electrical and computer engineering graduates. This is still the case even with the massive use of hardware description languages (e.g., Verilog [2] and VHDL [3]) and more powerful software tools to synthesize the hardware structures from textual descriptions (similar to programming) [4]. However, with the advent of FPGAs (Field-Programmable Gate Arrays) [5]—seen as the softening of hardware [6]—making it possible to implement complex architectures with the potential of being used as computing engines [7], this scenario seems to be changing, i.e., computer science students might play an important role in programming hardware based layers of FPGA based computing systems.

In certain computing systems (e.g., embedded systems) an FPGA might be used as the core component. That FPGA may integrate one or more on-chip microprocessors (in softcore or hardcore modalities) and dedicated components, some of them to globally accelerate the target application or to meet, besides performance, other requirements (e.g., energy savings, fault-tolerance, and security) [8]. Programmers of the emerging systems may also need the knowledge in order to specify the organization of the system architecture, since even that can be defined by programming the FPGA [9]. The hardware structures of FPGAs are new synergies available to system architects and programmers. Programmers may also need to know how application specific architectures may meet certain requirements and how those architectures can be designed so that programmers are able to go ahead with the development of certain systems. Since FPGAs are being more widely used, it might be possible in coming years to witness the mainstream use of FPGAs by computer science graduates without low-level hardware design expertise. This opinion has been also recently manifested in [10] and success requires that computer science graduates have knowledge of RTL (Register Transfer Level) design. This is even true when software programming languages (e.g., C) are used for FPGA programming [11].

Although many academic and industrial efforts have been made in order to make the automatic path from software code to hardware a reality (the subject started in the 80's with the advent of "silicon compilers" [12]), there are many unsolved issues that still hinder a generic tool to efficiently generate application specific architectures from pure software programs (e.g., coded using C language). This is one of the main reasons why

* Accepted 12 April 2007.

computer science students may need to acquire the knowledge of how to develop application specific architectures properly to execute a given algorithm in FPGAs.

Concerning embedded systems, a number of authors have already addressed the knowledge needed by future computer science graduates [13] and have also discussed several open issues [9]. They have pointed out the need to teach FPGA programming, among other important topics. Owing to their increasing importance, embedded systems have been the ultimate focus of several teaching efforts [14]. Those systems benefit most from using FPGAs. Note also that many authors use reconfigurable hardware platforms (using an FPGA as the main device) to teach embedded systems [15, 16], computer architecture and networks [17] embedded systems and mobile robotics [18], digital hardware design [19, 20], etc. Others use special courses on reconfigurable systems [21]. However, few of them strictly focus on FPGA programming bearing in mind the execution of algorithms as the main goal. An exception is the approach presented in [22], which uses mobile robotics as a motivational target to teach embedded systems and reconfigurable computing.

Many courses on how to teach hardware design using logic synthesis and hardware description languages (HDLs) for electrical and computer engineering programs have been proposed (see [23] and [24], just to name a few). In computer science programs, such courses frequently use the same methodologies, which most often address circuit design rather than architecture-specific design to execute a given algorithm. This paper shows the details of an approach to teach how to program FPGAs, in order to implement dedicated hardware engines for executing algorithms (image processing algorithms are mainly used). Note, however, that this approach assumes students know the basics about computer architecture and digital system design. Student's feedback based on collected responses to questionnaires, answered anonymously by students, shows experimental evidence of the effectiveness of the approach. The approach seems to have motivated students to acquire the required skills. This has been also proven by the fact that some of them, although at first attracted to software programming, computer networks, and information systems, have chosen final year engineering projects[1] using FPGAs.

The remainder of this paper is structured as follows. The next section presents the approach. Following that, the most relevant mini-projects students are required to implement are presented. Then some results of two implementations performed by students are given. The penultimate section presents some feedback from students and overall comments. Finally, the last section gives the conclusions.

[1] An engineering project in the last year of the engineering program that usually requires one or two semesters in full-time or part-time (with courses), respectively.

## METHODOLOGY

The approach presented in this paper has been tested in a single-semester course, named "Hardware/Software integration", in the 5th-year of the Informatics and System Engineering program at the University of Algarve in Portugal. The course has been taken by students from informatics, computer networks and embedded systems branches. All of them had single-semester courses on imperative and object-oriented programming languages (typically C and Java), algorithms and data-structures, mathematics (analysis and algebra), computer architecture (based on the Hennessy and Patterson book [25]), digital systems (covering Boolean functions, combinational and sequential circuits, finite state machines, arithmetic operations, etc.), operating systems, circuit analysis and signals and systems. "Hardware/Software integration" was the first course where students encountered FPGAs and VHDL.

### *Tools and devices*

There are a number of tools and devices that can be used to assist the learning process. In the context of the course presented in this paper, Xilinx ISE WebPack [26] was used for synthesis and place and route and ModelSim SE from Mentor Graphics [27] was used for VHDL simulation. Both are freely available, which is important for students to test their designs at home. As for the FPGA board, the Xilinx Spartan-3 Starter Kit Board [28], has been the one most used. It is a low cost board (approximately $90) with a Xilinx Spartan-3 FPGA (XC3S200FT256-4). In addition, the board includes a number of important devices such as: SRAM, Flash, 3-bit (8-color) VGA display port, RS-232 serial port, expansion connectors, LEDs, buttons, interrupts, and 7-segment displays.

### *Course organization*

The part of the course dedicated to VHDL, FPGAs and application-specific architecture design consists of PowerPoint$^{TM}$ presentations and laboratory lessons. A hands-on approach is used in the lab lessons. Students are able to try out their ideas and to learn from their own mistakes. The help of a faculty member is always important in areas where they are having more problems during those experiments. In the first part of the laboratory lessons students acquire the knowledge needed to be prepared for the mini-project (second part).

Figure 1 shows the flow of topics and examples used until students start their mini-project. In this flow some steps are dedicated to learning important aspects related to the boards used, FPGA

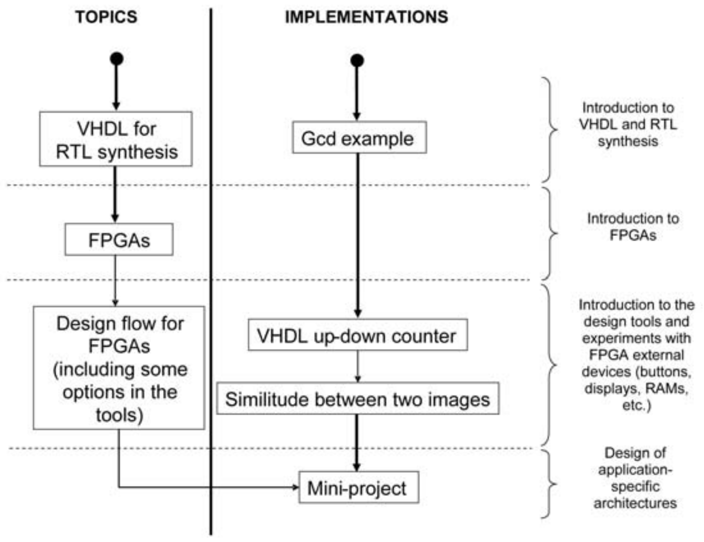

TOPICS
IMPLEMENTATIONS
VHDL for RTL synthesis
Gcd example
Introduction to VHDL and RTL synthesis
FPGAs
Introduction to FPGAs
Design flow for FPGAs (including some options in the tools)
VHDL up-down counter
Similitude between two images
Introduction to the design tools and experiments with FPGA external devices (buttons, displays, RAMs, etc.)
Mini-project
Design of application-specific architectures

Fig. 1. Flow of topics and examples used.

design flow (from VHDL to download of bitstreams), main options of the synthesis and place and route tools, VHDL simulation at different levels (functional, post place and route, etc.), and to some details (e.g., assigning FPGA pins to VHDL entity ports, interfacing to buttons and 7-segment displays, etc.). To test the board and the interface with displays and buttons a simple up/down counter is used.

### *VHDL*

A subset of VHDL, enforcing synthesizable constructs, which can easily be understood by students, is adopted. Synthesizable VHDL code is something they have to master. Sometimes because they could be coding something impossible to implement with the library of hardware components, other times because an architectural synthesis[2] tool would be needed. The greatest common divisor (gcd) algorithm presented in Fig. 2(a) shows students at the very beginning of the limitations of RTL and logic synthesis. Some of the students ask why the solution is not simply to code the algorithm in VHDL using a process instead of having to design a datapath and a control unit. Thus, students are exposed to the difference between architectural and RTL synthesis from the very beginning. A special focus on imaging the inferred hardware one may expect for each VHDL statement or group of statements is privileged. This enforces a connection to synthesis from the start and is usually well accepted by students.

In this phase, it is of extreme importance for students settle their doubts concerning the concurrent mechanism of VHDL and the delta cycle semantics [3]. Experience has shown that when doubts about the concurrency model of VHDL persist, they become the source of most problems students face when coding VHDL.

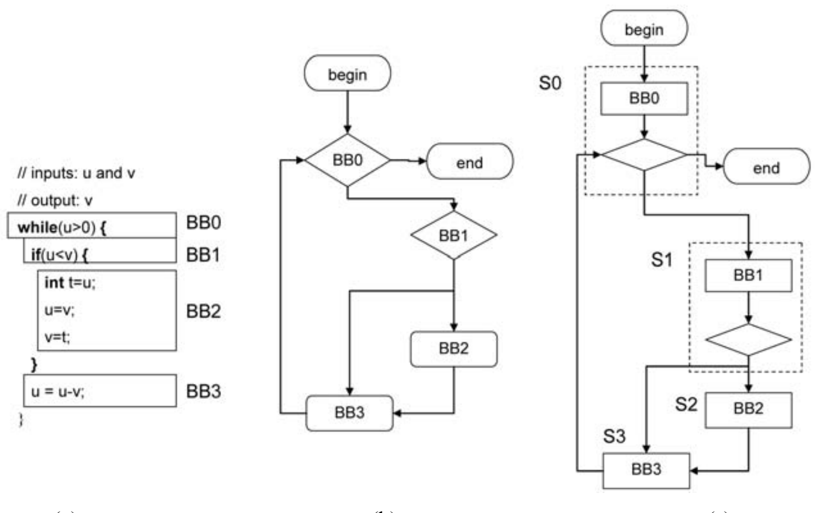
To guide students, the following VHDL tips are applied.

- Use process constructs as much as possible (code inside processes is more similar to the imperative model to which they are used).
- Use few VHDL lines per process (this reduces the potential for coding either non-synthesizable VHDL or erroneous functional descriptions).
- Use only "for loops" and with iteration bounds known at compile time (i.e., possible to be fully unrolled).
- Think as if one needs to construct the system based on components such as registers, ALUs (Arithmetic Logic Units), memories, FSMs (Finite State Machines), etc.
- Use a separate entity for each component of the architecture (this enhances modularity and helps test).
- Perform synthesis for each VHDL entity (this always gives students the perception of what hardware structures they are specifying).
- Use functional simulation to test each VHDL entity alone and integrated with other components early.

### *Synthesis topics*

This methodology assumes students do not have previous experience with high-level synthesis and they have only limited experience with datapath and control units in the one-semester courses: digital design and computer architecture. Those courses are not focused on design but mainly on analysis, assembly and on computer architecture concepts. When guiding them to application specific architectures, emphasis on a first assumption that the FSM of the control unit of the specific architecture they are supposed to design can be

[2] Also known as "behavioral synthesis" and "high-level synthesis". It is related to the capability of the synthesis tool to create a specific circuit with a datapath and a control unit from an algorithm.



```
// inputs: u and v
// output: v
while(u>0) {
  if(u<v) {
    int t=u;
    u=v;
    v=t;
  }
  u = u-v;
}
```

Fig. 2. (a) Algorithm to calculate the greatest common divisor (gcd) between two integer values; (b) control flow graph (CFG) of the algorithm; (c) state transition graph (STG) directly based on the CFG.

directly based on the control flow graph (CFG) of the algorithm (see the gcd example in Fig. 2). The CFG is a single-entry, single-exit, directed graph where each node represents a basic block (a sequence of instructions in the original code where the flow always starts on the first instruction and only leaves on the last instruction). From the CFG of a given algorithm they can easily achieve a first specification of a control unit: the state transition graph (STG), representing the states and transitions between states of the control unit, can be an image of the CFG. Later, they may think about optimizing the STG by including additional states (when scheduling the operations on each basic block) or by removing states. A dataflow graph is used in order to expose the data-dependences between operations to schedule the operations in each basic block, thus explicitly showing operations to be executed in sequence and those that can be executed in parallel.

Based on this approach, they also easily recognize the needed datapath hardware structures and the signals from the FSM to implement the iteration control of each loop, for instance.

*Initial examples*

The gcd example (see Figs 2 and 3) is used to continue the learning process about the flow to design an application-specific architecture and the simulation levels. In this example, students implement the architecture based on the algorithm and the datapath block diagram provided at the beginning. Afterwards, a design contest with two possible awards, one for the gcd architecture with the best execution time for a set of inputs previously given, and the other for the gcd architecture using the minimum FPGA resources (number of slices), is usually held. With regard to performance, they acquire sensitivity towards the balance between the number of clock cycles to determine the gcd and the maximum clock frequency achieved to reach the minimum execution time.

An example is then used so that students can gain experience with memories (on-chip block

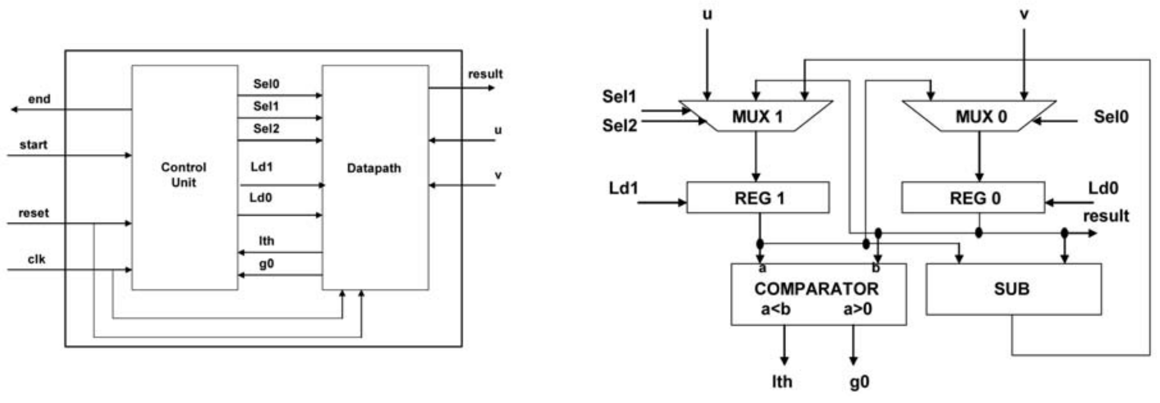


Fig. 3. The greatest common divisor (gcd): (a) block diagram of the application-specific architecture; (b) block diagram of the datapath.

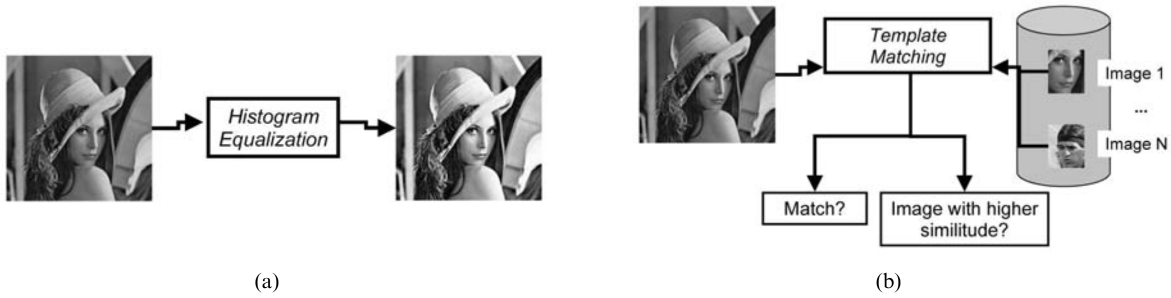


Fig. 4. Input/output results for two mini-projects: (a) image histogram equalizer; (b) image template matching.

RAMs are used, in this case). They design an architecture to calculate the similitude (the sum of the square differences is used) between two vectors stored in RAMs.

These two types of examples have been of very important in the successful achievement of the goals of the approach discussed in this paper.

*Overall comments*

This approach has successfully enabled students to acquire the skills needed to design specific (non-programmable) architectures to be implemented in FPGAs. It mainly focuses on autonomous skills that students usually acquire hands-on, conducting mini-projects during a number of laboratory lessons. At the end of the semester, students present and discuss their work and are required to write a technical report about the architecture designed. The following section shows the most relevant mini-projects used in past courses.

## MINI-PROJECTS

The approach mainly uses image processing kernels as the algorithms for which students have to design a non-programmable specific architecture. Figure 4 shows the expected input/output results for two of the algorithms used. In projects of this kind, images are usually the main input/outputs of the system to be implemented. Examples of image processing algorithms that have been used: are as follows.

- The image smoothing algorithm is an image filter used in image pre-processing tasks to reduce noise in images.
- The image equalization based on a histogram (the code of the algorithm is presented in Fig. 5) is used to enhance contrast in images.
- The template matching algorithm is an algorithm to find the closest image in a given database of images to an input image. It considers database images of the same size (smaller than or equal to the input image) and working by moving each image in the database through the input image, calculating the similitude (the sum of the square differences is used) at each position. At the end, the image in the database leading to a similitude value below a certain threshold is considered to be the matching one. LEDs in the board are used to indicate the result of the algorithm after execution.

*Main characteristic of the examples*

Besides other properties, those examples have loop nested structures and array variables, important computing and data constructs to achieve the goals of our approach, because they expose features that exist in the most interesting applications and are able to gain from implementations using application-specific architectures. Note that our goals include programming FPGAs being aware of the hardware resources available in order to achieve efficient non-programmable application-specific architectures.

*Work methodology*

During the course students are invited to search VHDL components available on the Internet (e.g., repositories of IP cores) and to give them in-house cores such as: a combinatorial VHDL integer divider, a VGA interface core, VHDL cores to inference Block RAMs, etc. In addition, students receive the code for each of the algorithms and input test images. In this way, they have an executable and correct specification of the problem, so they have a reference whenever they need to compare VHDL or FPGA results with those obtained by executing the algorithm using software. This has also been important with examples where students need to transform floating- to fixed-point representations, because they can measure the accuracy obtained after the transformation and can explore the number of bits of the fixed-point representations before starting the design of the architecture. They also receive auxiliary programs in order to help them in the design process. One of the programs reads images (jpeg and gif formats) and translates them into VHDL representations in order to initialize RAM content.

Students are encouraged to design architectures with parameterization capabilities so that the hardware description of the design can be adapted to different characteristics (e.g., image sizes considering maximum sizes of $1024 \times 740$ pixels).

*FPGA Resources*

FPGA on-chip memories (Block RAMs) have

```
// L = 256; number of gray values
// short Xsize = 64, Ysize = 64;
// byte image[Xsize][Ysize]; (input image)
// int histogram[L]; (histogram)
// int gray_level_mapping[L]; (gray mapping)
// byte out_image[Xsize][Ysize]; (output image)

for (int i = 0; i < L; i++) histogram[i] = 0;

// Compute the image's histogram
  for (int i = 0; i < Xsize; i++)
    for (int j = 0; j < Ysize; j++)
      histogram[image[i][j]] += 1;

// Compute the mapping from the old to the new gray levels
Float cdf = 0.0;
Float pixels = (float) (Xsize*Ysize);
for (int i = 0; i < L; i++) {
  cdf += ((float)(histogram[i])) / pixels;
  gray_level_mapping[i] = (int) (255.0 * cdf);
}

// generate the new image
for (int i = 0; i < Xsize; i++)
  for (int j = 0; j < Ysize; j++)
    out_image[i][j] = gray_level_mapping[image[i][j]];
```

Fig. 5. Java code for the image histogram equalization algorithm (based on [29]).

been used to store images and auxiliary arrays, if needed. This might constrain the dimensions of the images to low sizes but has the convenience of simplifying tests, because this solution does not need to transfer image data to SRAMs on the board. The images are transferred along with the FPGA programming bitstreams[3]. To test the architectures, input/output interfaces are usually employed such as: LEDs, VGA monitors, or specific hardware components added to the design. One of those components uses a Block RAM having the expected content and an auxiliary circuit to compare contents of RAMs. It traverses the RAM content output by the architecture and the correct content stored in the added Block RAM and flags possible mismatches.

*Optimizations*

Algorithms using floating-point representations are first translated to fixed-point by students. In this step, they have to study the accuracy to obtain acceptable fixed-point representations (as is the case for the example shown in Fig. 5 for the code statements referring to the scalar variable cdf). In some of the mini-projects students also observe the optimizations that can be carried out when divisions and multiplications by constants are needed (operator strength reduction) and they acquire more awareness to some code transformations (e.g., loop unrolling, scalar replacement).

*Comments on the mini-projects*

Other mini-projects include microprocessor and microcontroller cores (simple MIPS and PIC cores). However, these kinds of projects are usually less motivating to computer science students since they do not see the practical advantage, besides learning hardware design, of designing a microprocessor. Conversely, they easily see the advantage in being able to implement a specific machine (e.g., to accelerate an algorithm). Also note that mini-projects based on microprocessor and microcontroller cores are the ones usually used in most teaching approaches. The fact that image processing algorithms appear to capture students' attention better also seems to be relevant. This may be comparable to the motivation students show when applications for mobile robotics are used (see, e.g., [18]]).

*Illustrative example*

As an illustrative example of the options students may have to face, two implementations of the image histogram equalization algorithm carried out by two groups of two students (identified here as Group A and Group B) are illustrated. Table 1 shows the main design options faced by students and their decisions. The following paragraphs illustrate some of the main design decisions that students made.

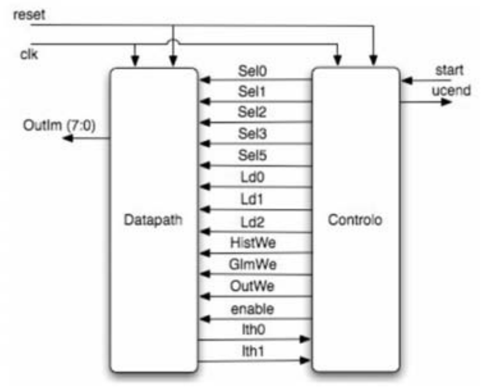
- Group A implemented an architecture that is able to compute images of 95×96 in 73 315 clock cycles (maximum frequency of 74.6 MHz). This group parameterized the number of bits used for the fractional part of the computations (when using 8 bits instead of 16 the maximum clock frequency is about 100 MHz). They decided to use a single hardware structure responsible for loop iterations and to share it among all the loops in the program. They also decided to implement the required divider as a datapath and control unit.
- Group B designed an architecture able to compute images of 67×80 pixels in 49 533 clock cycles (maximum frequency of 13.3 MHz). They used a combinatorial implementation of the divider and thus obtained a lower clock frequency (higher clock frequencies would require the insertion of pipelining stages or additional FSM states) and needed a higher number of FPGA resources. This group did not consider sharing hardware structures. As an example, Fig. 6 shows three views of the architecture: (a) the interconnections between datapath and control units, (b) the state transition graph of the control unit, and (c) the block diagram of the datapath (RTL).

Figure 7 shows the number of FPGA resources used (Slices, Flip-Flops, 4-LUTs and BRAMs) for the two architectures selected. As shown, very different solutions have been achieved by the two groups of students. Results of this kind are important to show and discuss with students, so that they

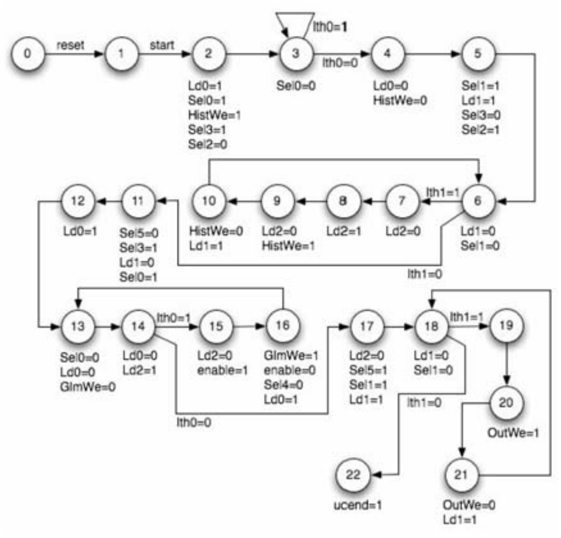
[3] Binary data to program the FPGA hardware resources.

Table 1. Main options and those taken by each group

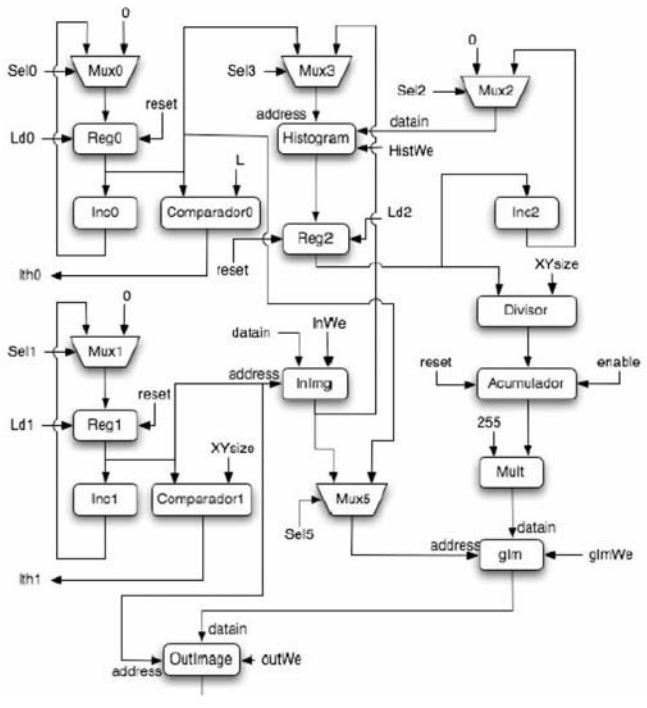
| Design options | Group A | Group B |
|---|---|---|
| Division | Sequential (control + datapath) | Combinatorial |
| Fractional bit-widths | 8 or 16 | 8 |
| Sharing of functional units | Sharing of a single hardware structure to control loop iterations among all the loops | No |
| Sharing of internal memories | Same memory to store input and output image | Different memories |



(a)



(b)



(c)

Fig. 6. Image histogram equalization architecture implemented by Group B: (a) top level of the architecture; (b) state transition graph for the control unit; (c) block diagram of the datapath unit.

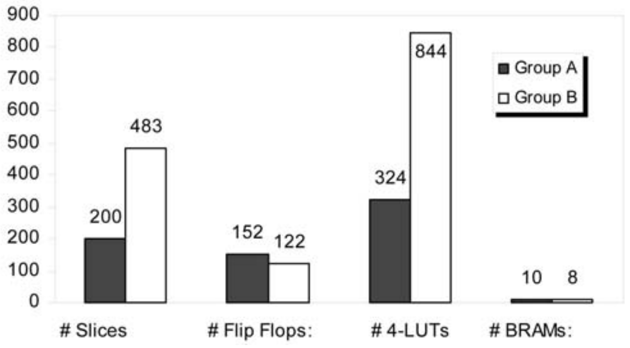


Fig. 7. Resources used for two implementations of the image histogram equalization algorithm done by students.

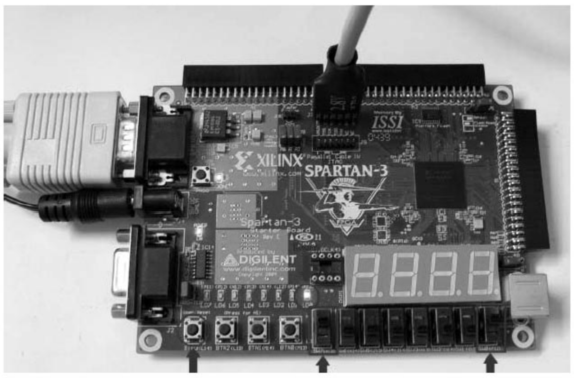


Fig. 8. FPGA board used in the experiments.

observe the substantial differences on the implemented architectures.

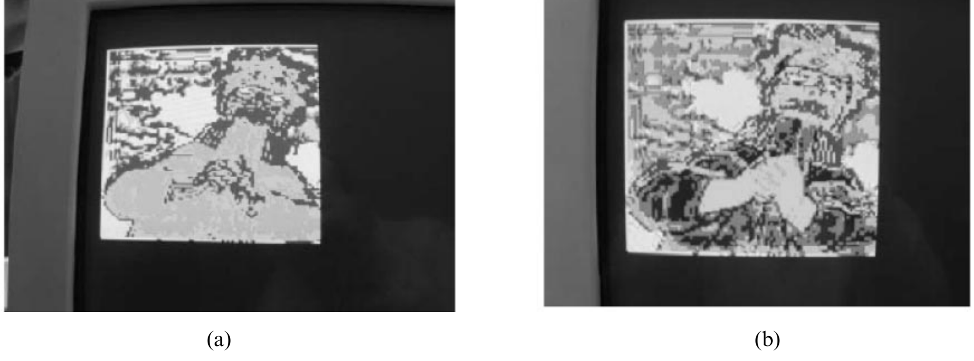
Figure 8 shows the FPGA board used by students to implement the image histogram equalization architecture. The cables used to connect the board to the VGA monitor and to the JTAG interface for programming via a PC are shown. The arrows at the bottom point to one push-button and two switches used for hardware reset, to start the execution of the image histogram equalization, and to select the image seen in the VGA monitor (between the input and the output image), respectively. Figure 9 shows an example of the pictures they obtained in real-time connecting the FPGA board to a VGA monitor (using 3-bit color representation).

## OVERALL COMMENTS

Students' opinions were collected in the 2004/2005 and 2005/2006 academic years using questionnaires. Approximately 62% of the students consider the projects either very or just interesting. Although students do not consider the projects simple, they do not judge them as more difficult than the most difficult ones they have done in other undergraduate courses. The semantics and especially the support to VHDL concurrency have been mentioned as the most difficult topics when learning the language. Students mentioned advanced digital systems, computer architecture and applications (image processing, signal processing, control systems, etc.) as the most necessary skills to perform the projects.

As for the biggest difficulties they faced, they claimed these were the problems they had using the tools (especially the simulator). This is not surprising, since for most of them this was the first experience with an HDL simulator. VHDL learning has also been chosen by most students as one of their most problematic topics.

Experience has revealed that groups of two students are preferable. A student alone may become frustrated in certain phases of the project since they are unable to consult with other students involved in the same problem. Groups of more than two students seem to lead to inadequate task distribution, which in the end usually results that



(a) (b)

Fig. 9. VGA monitor showing results obtained with the hardware implementation of the image histogram equalization: (a) the input Naruto image with 3-bit color representations; (b) the resultant image.

one of the members contributed much less to the solution. Note that two students per group also creates the potential to experiment with modern software engineering methods such as pair-programming [30], which can also be successfully applied to VHDL coding.

A choice of different types of projects allows students to be able to select the one that more highly motivates them. This seems to be very important, especially when they have to face learning challenges, which are easier to undertake with high levels of motivation.

## CONCLUSIONS

This paper presents an approach to teaching FPGA programming to informatics engineering undergraduate students. The goal is that they acquire the necessary skills in order to program FPGA devices to execute dedicated computing engines. This is a challenging task since informatics engineering students do not usually master low-level hardware details. From these experiences the approach seems to allow students to acquire the necessary knowledge to design non-programmable application-specific architectures for FPGAs successfully. Such knowledge includes the learning of a VHDL language subset, synthesizable constructs, FPGA commercial tools and relevant FPGA internal structures. Strong evidence has shown us that image processing algorithms not only expose the most relevant architectural issues, but also add greater motivation for students to achieve these goals. The fact that these system inputs and/or outputs images can be seen on a VGA monitor connected to the FPGA board strongly contributes to the extra motivation.

The approach is not tied to a specific undergraduate degree in computer science or informatics engineering and only requires students to master traditional topics taught in digital systems, computer architecture, and programming languages. Using mostly image processing algorithms does not make the approach less effective with other kinds of applications. Note, however, the design of complex and sophisticated interface circuits may require students to master lower levels of hardware design.

Future enhancements should include topics about loop pipelining to raise students' awareness of the performance improvements that can be obtained using this optimization. In addition to this, creating a higher level of abstraction to help on teaching architectural design appears to be important. A graphic tool where components can be defined and interconnected (component-based approach), each one implementing a VHDL entity with only one process would discipline and may avoid many of the problems faced by students when developing synthesizable VHDL code.

*Acknowledgments*—The author acknowledges the donation of boards and software licenses from Xilinx, Inc. The ideas presented in this paper have been tested and evaluated during the author's affiliation with the University of Algarve, Portugal. A warm acknowledgement is given to all the students who contributed to the validation of the ideas presented. The author especially acknowledges students João Bispo, Ricardo Avó, José Coimbra and Ruben Gomes for the mini-project implementations and results shown.

## REFERENCES

1. Michael J. S. Smith, *Application-Specific Integrated Circuits (The VLSI Systems Series)*, 1st edn, Addison-Wesley Professional, (1997).
2. Donald E. Thomas and Philip R. Moorby, *The Verilog® Hardware Description Language*, Springer, 5th edn, (2002).
3. Gray Armstrong, *VHDL Design Representation and Synthesis*, Prentice-Hall, (2000).
4. Pong P. Chu, *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability*, Wiley-IEEE Press, 2006.
5. Stephen Brown and Jonathan Rose, FPGA and CPLD architectures: a tutorial, in *IEEE Design & Test of Computers*, **13**(2), 1996.
6. F. Vahid, The softening of hardware, *IEEE Computer*, **36**( 4), 2003, pp. 27–34.
7. Maya Gokhale, and Paul S. Graham, *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*, Springer, 1st edn, 2006.
8. Patrick Lysaght and Wolfgang Rosenstiel (Eds), *New Algorithms, Architectures, and Applications for Reconfigurable Computing*, Springer, 1st edn, 2005.
9. João M. P. Cardoso, New challenges in computer science education, in *10th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'05)*, *Universidade Nova de Lisboa*, Lisbon, Portugal, June 27–29, 2005, ACM Press, pp. 203–207.
10. Kevin Morris, Death of the hardware engineer: A dirge for the digital designer, *FPGA and Structured ASIC Journal*, April 18, 2006 [http://www.fpgajournal.com].
11. David Pellerin and Scott Thibault, *Practical FPGA Programming in C*, Prentice Hall, 2005.
12. D. Gajski (Ed.), *Silicon Compilers*, Addison–Wesley, Reading, MA, (1987).
13. R. Hartenstein, The changing role of computer architecture education within CS curricula, invited talk in *Workshop on Computer Architecture Education (WCAE'04)*, June 19, 2004, at *31st Int. Symposium on Computer Architecture*, Munich, Germany, June 19–23, 2004.
14. W. Wolf, and J. Madsen, Embedded systems education for the future, in *Proceedings of the IEEE*, **88**(1), 2000, pp. 23–30.
15. Falk Salewski, Dirk Wilking and Stefan Kowalewski, Diverse hardware platforms in embedded systems lab courses: A way to teach the differences, *SIGBED Review: Special Issue on the First Workshop on Embedded System Education (WESE)*, **2**(4), October 2005.

16. Stephen A. Edwards, Experiences teaching an FPGA-based embedded systems class, in *Proceedings of the workshop on embedded systems education (WESE)*, Jersey City, NJ, September 2005.
17. Kôki Abe, Takamichi Tateoka, Mitsugu Suzuki, Youichi Maeda, Kenji Kono and Tan Watanabe, An integrated laboratory for processor organization, compiler design, and computer networking, *IEEE Transactions on Education*, **47**(3), 2004, pp. 311–320.
18. V. Bonato *et al.*, "Teaching embedded systems with FPGAs throughout a computer science course, in *Workshop on Computer Architecture Education (WCAE 2004)*, June 19, 2004, at *31st Int. Symposium on Computer Architecture*, Munich, Germany, June 19–23, 2004, pp. 8–14.
19. N. Calazans and F. Moraes, Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses, *IEEE Transactions on Education*, **44**(2), 2001, pp. 109–119.
20. J. Amaral, P. Berube and P. Mehta, Teaching Digital Design to Computing Science Students in a Single Academic Term, *IEEE Transactions on Education*, **48**(1), 2005 pp. 127–132.
21. Valery Sklyarov and Iouliia Skliarova, Teaching reconfigurable systems: Methods, tools, tutorials, and projects, *IEEE Transactions on Education*, **48**(2), 2005, pp. 290–300.
22. Jorge Silva, Marcio M. Fernandes, Vanderlei Bonato, Ricardo Menotti, João M. P. Cardoso and Eduardo Marques, Using mobile robotics to teach reconfigurable computing, *The 1st International Workshop on Reconfigurable Computing Education (RC-Education)*, March 1, 2006, Karlsruhe, Germany.
23. G. Puvvada, and M. A. Breuer, Teaching computer hardware design using commercial CAD tools, *IEEE Transactions on Education*, 36(1), 1993, pp. 158–163.
24. Tsai Chi Huang, R.W. Melton, P.R. Bingham, C.O. Alford and F. Ghannadian, The teaching of VHDL in computer architecture, in *International Conference on Microelectronics Systems Education (MSE'97)*, (1997), p. 0133.
25. D. A. Patterson and J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kaufmann, 3rd edn, (2004).
26. http://www.xilinx.com
27. http://www.mentor.com
28. Xilinx Inc., *Spartan-3 Starter Kit Board User Guide*, UG130 (v1.1), May 13, 2005.
29. P. M. Embree and B. Kimble, *C Language Algorithms for Digital Signal Processing*, Prentice-Hall, (1991).
30. Pair Programming, an Extreme Programming practice, http://www.pairprogramming.com

**João M. P. Cardoso** finished his Ph.D. in Electrical and Computer Engineering at the IST, Lisbon, Portugal, in 2001. He has been Assistant Professor in the Department of Informatics Engineering at the *Instituto Superior Técnico* (IST), Technical University of Lisbon since April 2006. He is also a senior researcher at the INESC-ID in Lisbon. Previously, he was a faculty member in the Faculty of Sciences and Technology, at the University of Algarve, Portugal. In 2001/2002 he worked for PACT XPP Technologies, Inc., Munich, Germany. There he participated in the research and development of the C compiler for the eXtreme Processing Platform (XPP). He was program chair of ARC'05 and general co-chair of ARC'06, the International Workshop on Applied Reconfigurable Computing. He serves as a Program Committee member for various conferences (IEEE FPT, FPL, ARC, SAMOS VI, ACM SAC-EMBS, etc.). His research interests include reconfigurable computing, compilation techniques, application specific architectures and design automation of embedded systems. He is a member of the IEEE, the IEEE Computer Society and the ACM.