Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 952560, 14 pages
doi:10.1155/2011/952560

Research Article

Experiment Centric Teaching for Reconfigurable Processors

Loïc Lagadec, Damien Picard, Youenn Corre, and Pierre-Yves Lucas

Université de Brest, CNRS, UMR 3192 Lab-STICC, ISSTB, 20 avenue Le Gorgeu, 29285 Brest, France

Correspondence should be addressed to Loïc Lagadec, loic.lagadec@univ-brest.fr

Received 22 July 2010; Accepted 17 December 2010

Academic Editor: Michael Hübner

This paper presents a setup for teaching configware to master students. Our approach focuses on experiment and leaning-by-doing while being supported by research activity. The central project we submit to students addresses building up a simple RISC processor, that supports an extensible instructions set thanks to its reconfigurable functional unit. The originality comes from that the students make use of the Biniou framework. Biniou is a research tool which approach covers tasks ranging from describing the RFU, synthesizing it as VHDL code, and implementing applications over it. Once done, students exhibit a deep understanding of the domain, ensuring the ability to fast adapt to state-of-the-art techniques.

1. Introduction

Innovative lectures and lab courses are required to offer high quality training in the field of configware. Being either an electrical engineer (EE) or a computer scientist (CS) expert will not be enough to meet the needs we foresee in terms of interdisciplinary for the future. As teachers, our goal is not to output Computer-Assisted-Design (CAD) end-users but highly educated experts, who will easily self-adapt to new technologies.

Our contribution to this in-depth rethinking of curricula goes through providing cross expertise training centered around CAD environments design. CAD tools embed the full expertise both from an architectural and from an algorithmic point of view. Affording the design of CAD environments ensures a full understanding of the domain.

As teachers, we make use of some research tools we have developed, that offer a full design suite for reconfigurable accelerators. The key principle behind this is to let students design and implement simple schemes (processors, processor-to-accelerator coupling, etc.) while taking advantage of research tools that promote high productivity. After students have manipulated these toys examples, they show a promising learning curve when addressing state-of-the-art technology (processor soft cores, Xilinx design suite, FSL Fast Serial Links, etc.). This second stage is when performances issue arises. At this point, some discussions happen: fine-versus coarse-grained accelerators, compiler friendly architecture, reconfigurable functional unit versus coprocessor, and so forth.

Splitting the learning activities in such a way emphasizes simplicity. A first consideration is that a simple design always takes less time to finish than a complex one, exhibits more readability, and offers a better support for further refactoring. Another thing about simple designs is that they require knowledge to recognize. Knowledge is different from information. Information is what you get as a student, when gaining access to a lecture. However, you can have plenty of information and no knowledge. Knowledge is insight into your problem domain that develops over time. Our teaching approach aims at accompanying students from information to knowledge.

This paper reports this experience. The rest of the paper is organized as follows: Section 2 introduces the lecture's context along with the experiment centric approach we followed. Section 3 focuses on the project we submit to students. Section 4 shifts from the toy example to a more realistic scope. Section 5 summarizes the benefits of our approach.

2. Experiment Centric Teaching

2.1. Local Scope

2.1.1. Local Curriculum. The master curriculum "Software for Embedded Systems" opened two years ago at the

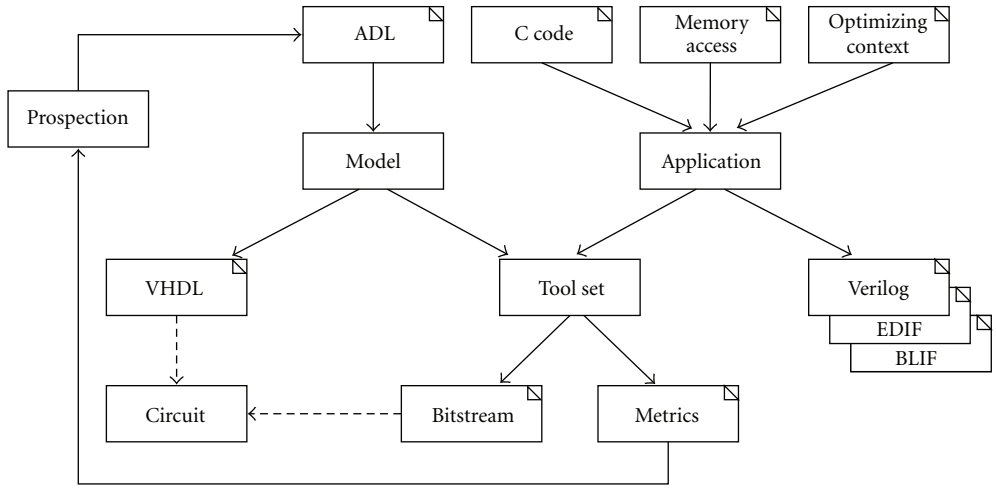

Figure 1: Overview of the Biniou flow.

university of Brest. This curriculum addresses emerging trends in embedded systems and highly focuses on reconfigurable embedded systems, with a set of courses for teaching hardware/configware/software codesign. The master gathers students from both CS and EE former curricula. The current master size is 12, coming from half a dozen countries. Half of the students are former local students hence own a sound background in terms of CS but suffer from lacks in electronic system design.

The reconfigurable computing courses are organized around two main topics covering the hardware (architectures) and software (CAD tools and compiler basics) aspects. These courses enable students to build from their previous knowledge a cross-expertise giving a complete vision of the domain.

A strength of this teaching approach is to partially rely on a research environment rather than purely on Xilinx hands-on tutorials. This offers the opportunity to exercise internal changes on algorithms and architectures, and to address both state-of-the art concepts and both some more prospective topics such as innovative—and still confidential in the industry—architectural trends.

First an overview of the reconfigurable computing (RC) landscape is introduced. Both industrial and academic architectural solutions are considered. This course is structured in three parts:

(i) overview of RC for embedded systems (2 sessions),

(ii) virtualization techniques for RC (2 sessions),

(iii) Modeling and generation of reconfigurable architectures (1 session). The second item addresses both state-of-the-art tools and algorithms in one hand as well as locally designed tools in another hand. The key idea is that students tend towards learning classical (or vendors's) tools so that they can bring a direct added-value to any employer of the field, hence get in an interesting and well-paid job.

However, tools obviously encapsulate the whole domain-specific expertise, and letting students "open the box" closes the gap between "lambda users" and experts. This takes up the challenge of providing a valuable and innovative curriculum. Obviously a single class is not wide enough to address all the-above mentioned items, but this course is closely integrated with some others such as "Numeric and Symbolic synthesis" or "Test & Simulation".

2.1.2. Legacy CAD Development. The research group behind this initiative is the Architectures & Systems team from the Lab-STICC (UMR 3192). This group owns a legacy expertise in designing parallel reconfigurable processor (the Armen [1] project was initiated in 1991) but has been focusing on CAD environment developments (Madeo framework [2]) for the past 15 years. The Madeo framework is an open and extensible modeling environment that allows to represent reconfigurable architectures then acts as a one-stop shopping point providing basic functionality to the programmer (place&route, floorplanning, simulation, etc.). The Madeo project ended in 2006 while being integrated as facilities in a new framework. This new framework, named Biniou, embeds additional capabilities such as, from the hardware side, VHDL export of the modeled architecture and, from the software side, wider interchange format and extended synthesis support. Biniou targets reconfigurable System-On-Chip (SOCs) design and offers middleware facilities to favor a modular design of reconfigurable IPs within the SOC.

Figure 1 provides an overview of Biniou. In the application side (right) an application is specified as C-code, memory access patterns and some optimizing contexts we use to tailor the application. This side outputs some post-synthesis files conforming to mainstream formats (Verilog, EDIF, BLIF, PLA). Results can be further processed by the Biniou Place and Route (P&R) layer to produce a bitstream. Of course the bitstream matches the specification of the underlying reconfigurable target, being the target modeled using a specific Architecture Description Language (ADL). A model is issued on which the P&R layer can operate as previously mentioned, and a behavioral VHDL description

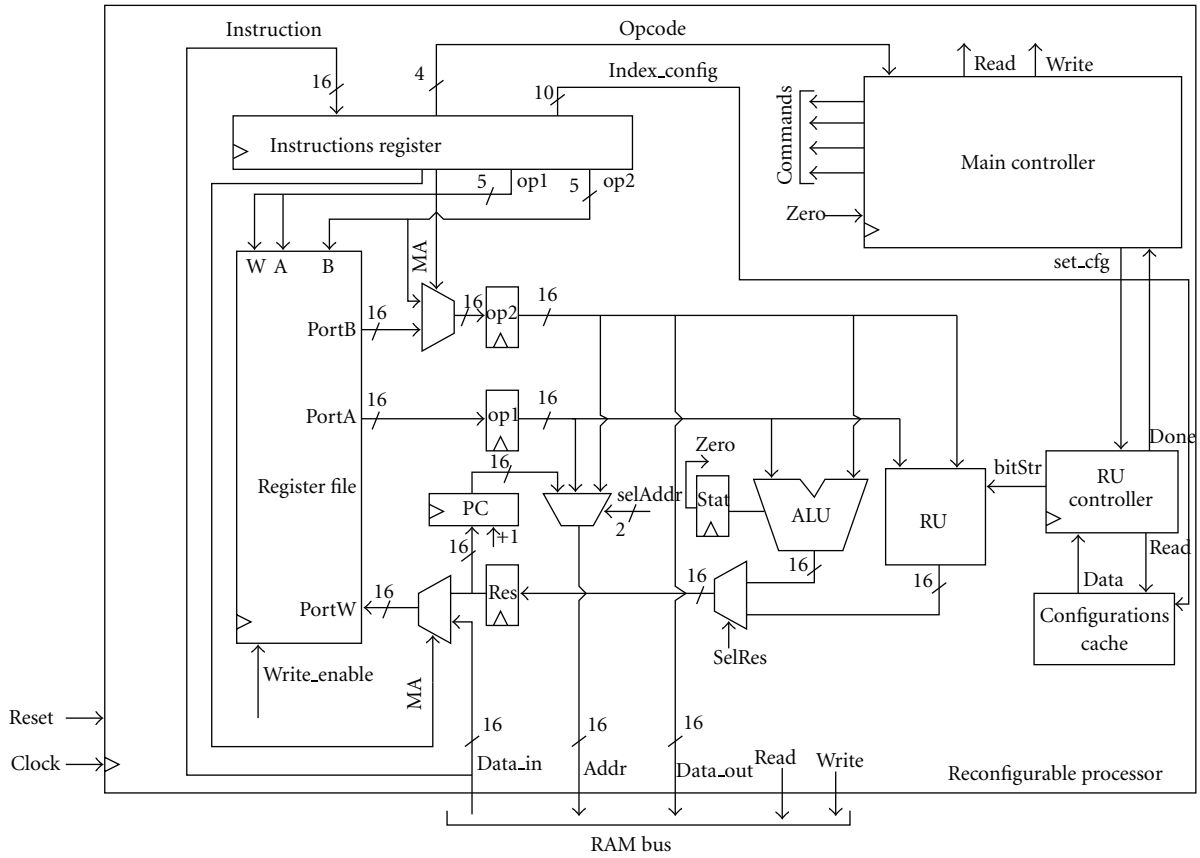

Figure 2: Schematic of the entire reconfigurable processor.

of the target is generated for simulation purposes and FPGA implementation.

Once a bitstream is generated out of the application specification, the designer can download it to configure its platform.

Also some debugging facilities can be added either in the architecture itself or as parts of the application [3].

2.1.3. From Research to Teaching. Biniou has been exercised as a teaching platform for Master 2 students. This happened in a reconfigurable computing course. In addition to lectures, students practice reconfigurable computing through practical sessions and exercise their new skills through a project. This project covers VHDL hand-writing, reconfigurable architecture modeling and programming, code generation, and modules assembly in order to exhibit a simple processor with a reconfigurable functional unit. This extra-unit allows to extend its instructions set.

2.2. Practical Sessions. Practical sessions are organized as three activities. The first activity is to gather documentation and publications related to a particular aspect of the course; the students have to present their short bibliographic study individually in front of the whole class.

The second activity is centered around algorithms used to implement applications over a reconfigurable architecture: point-to-point and global routers, floorplanners, placers. Some data structures such as Transitive Closure Graphs (TCG) are introduced later on in order to point out the need for refactoring and design patterns use [4]. This bridges the software expertise to the covered domain (CAD tools for reconfigurable architecture).

The third activity is related to tools and formats. Three slots are dedicated to VHDL that most of the students do not know. Manual description of fine grained reconfigurable architecture is introduced within this amount of time.

Some sessions are dedicated to practicing required tools; students manipulate logic synthesis tools (SIS [5], ABC), file formats conversion (Verilog, EDIF, BLIF, PLA), and behavioral synthesis according to some data access pattern (Biniou). We also offer a web-based tool [6] to output RTL netlist that students use to exercise several options for netlist generation.

Students create their own FPGA using Biniou, that is further reused in the project under a tuned up version.

2.3. Project Description. The project consists in designing a simple RISC processor, that can perform spatial execution through a Reconfigurable Functional Unit (RFU).

Coupling an RFU along with a processor to get a reconfigurable processor is one out of other alternatives for accelerating intensive tasks. The concept of instruction set metamorphosis [7] is defined and a set of architectures are described. For example, P-RISC [8], Garp [9], XiRISC [10], and Molen [11]. A specific focus is set on the Molen programming model and its architectural organization. The Molen approach is presented as a meeting point between the software domain (sequential programming and compiler) and the hardware domain (specific instruction designed in hardware).

Figure 2 illustrates the schematic view of the whole processor, including the RFU.

The processor supports a restricted instructions set, that conforms to a SET-EXECUTE-STORE Molen paradigm [11]. In order to keep the project reasonably simple, we restrict the use of the RFU to implementing Data Flow Graphs (DFGs) on one hand, and we provide students with the Biniou framework on the other hand. Restricting the use of the reconfigurable part as a functional units also mitigates the complexity of the whole design. However, this covers the need for being reachable by average students while preserving the ability to arouse's top students curiosity, by offering a set of interesting perspectives for further developments.

This project let students build and stress new ideas in many disciplines related to reconfigurable computing such as spatial versus temporal execution, architectures, programming environments, and algorithms.

2.3.1. Context. This project takes place during the fall semester, from mid October to early January. A noticeable point is that almost no free slots within the timetable are dedicated to this project, that overlaps with courses as well as with "concurrent" projects. This intends to stress students and make them aware of handling competing priorities.

2.3.2. Expected Deliverables. We define three milestones and three deliverables. The milestones are practical sessions in front of the teacher.

Three main milestones are as follows.

M1: RISC processor, running its provided test programs.

M2: RFU, with Galois Field-based operations implemented as bitstream.

M3: Integration, final review.

2.3.3. Schedule. The schedule is provided during the project "kick-off". To prevent students from postponing managing this project we use the collaborative platform to monitor activities, to specify time-windows for uploading deliverables, and to broadcast updates/comments/additional information. Reminders can be sent by mail when the deadline is approaching. Once the deadline expires, over-due deliverables are applied a penalty per extra half-day.

Table 1: Instruction layout.

15	14	13	12	11	10	9	8	7	6	5	4	···	0
Opcode				MA				OP1				OP2	

3. Project

3.1. Processor Soft-Core. Designing such a simple processor carries no extra value and several teaching experiments are reported [12]. However, keeping in mind that half the students have never exercised writing VHDL description, and given practice makes success, we decided to let students design their own processor. Although, a preliminary version with missing control structures was provided in order to ensure a minimal compatibility through the designs. Obviously, the matter here was to ease evaluation from a scholar point of view as well as to force students to handle kind of legacy system and refactoring rather than full redesign.

We also provided the instruction set and opcode. In an ideal world, and with a more generous amount of time to spend on the project, as the design is highly modular, building a working design by picking best-fit modules out of several designs would have also been an interesting issue.

3.1.1. Decoder. It outputs signals from input instruction according to the layout on Table 1. This information is provided to ensure compatibility as well as programmability (as no compiler support is considered).

3.1.2. Test Bench Program. Students are familiar with agile programming, test-driven development and characterization tests. When designing a processor, the same approach applies but at a wider granularity (program execution instead of unit test). Hence, we distributed some test bench programs. Analyzing at specific timestamps (including after the application stops) the internal states (some signals plus registers contents) leads to design scoring.

3.2. Reconfigurable RFU Design

3.2.1. Background. In order to give to students the main architectural concepts behind FPGAs, we first focus on a simple mesh of basic processing elements composed of one 4 entries Look-Up Table (LUT) each. Combination of the basic blocks (LUT, switch, buses, and topology) is presented as a template to be extended (in terms of routing structure and processing elements) for building real FPGA. A more realistic example from the industry (a Xilinx Virtex-5) is considered with a highlight on template basic blocks in Xilinx schematics. As a result, students are able to locate the essential elements for a better understanding of state-of-the-art architectures. Drawbacks of fine-grained architectures such as low computation density and routing congestion are highlighted to introduce coarse-grained architectures. This type of reconfigurable architecture is firstly presented as a specialization of FPGA suited for DSP application domain.

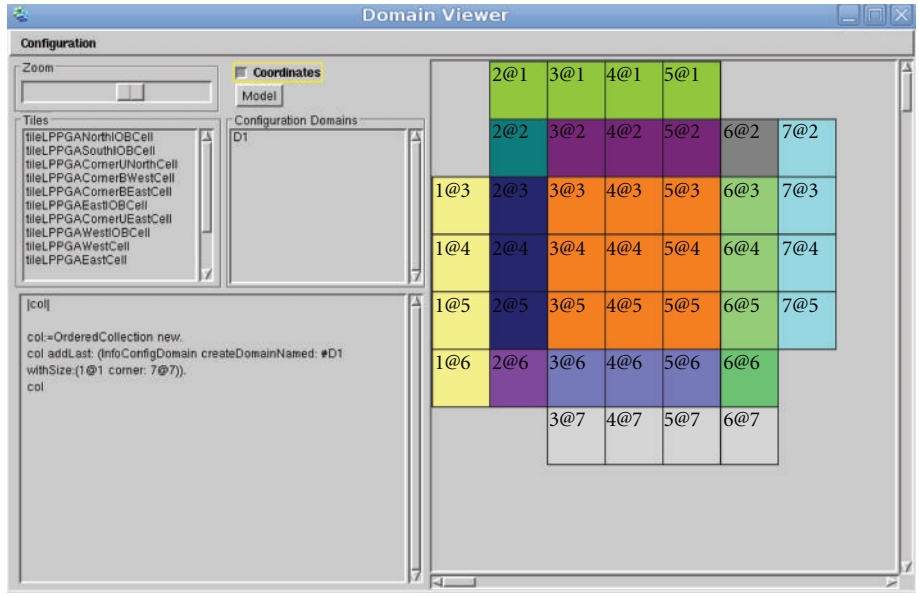

Figure 3: On the right, view of the different cell types composing the matrix (border cells, middle cells, IO cells). On the left, configuration domains are defined as a set of rectangular boxes. They can be reconfigured independently from each other.

Architectures presented are Kress-Array [13], Piperench [14], PACT XPP [15], and Morphosys [16]. Programming model issues are discussed with a comparison between software oriented approach (generally using subsets of C) and hardware approach (netlist based descriptions). A case study of the DREAM architecture is presented with an emphasis on the compiler friendly approach of the tools targeting the PiCoGA [17, 18].

3.2.2. Modeling. Before entering the generation phase, students learn to hand-design an FPGA. Every elements of a basic FPGA are detailed and a corresponding VHDL behavioral description is provided. The bottom-up description starts from atomic elements, such as pass gates, multiplexers, that are combined to form input/output blocks and configurable logic blocks. A daisy chain architecture is detailed as well as a configuration controller.

Then, the second part describes the Biniou generation of the architecture from an ADL description. An FPGA is described using an ADL increasing the level of abstraction compared to a VHDL description. The configuration plan is described as a set of domains to support partial reconfiguration. The approach relies on model transformation, with an automatic VHDL code generation from a high-level description.

3.2.3. RFU Structure. As a preliminary approach, students have to design an island style mesh architecture, what means sizing the matrix, defining a basic cell, and isolating border cells that deserve special attention. The basic cell is either used as is for the internal cells and tuned to generate the border cells because their structure is slightly different from the common template. Defining the domains appears as shown by Figure 3.

The basic cell schematic view is provided by Figure 4.

Ultimately, the full matrix appears as an array of N^2 cells as illustrated by the snapshot of the Biniou P&R layer (Figure 5).

3.3. Reconfigurable Functional Unit Integration. The reconfigurable functional unit (RFU) is composed of three main components: the reconfigurable matrix (RM) generated by Biniou, a configuration cache, and the RFU controller both hand-written (see bottom right in Figure 2).

Configuration is triggered by the processor controller which reacts to a SET instruction by sending a signal to the RFU controller. The RFU controller drives the configuration cache controller, which provides back a bitstream on demand.

The processor controller gets an acknowledgment after the configuration completed.

One critical issue about the processor-RFU coupling lies in data transfers to/from the RFU. Students have to design a simple adapter which connects a set of RFU's iopads to the processor registers holding input and output data (Op1, Op2, and Res in Figure 2).

Figure 6 gives a detailed view of the adapter.

3.4. Application Synthesis over the RFU. To let students figuring out the benefit of adding the RFU to the processor design, it is desirable that students can assess and compare the impact of several options. One classical approach lies in isolating a portion of the application to be further converted into an accelerated function. In this case, we

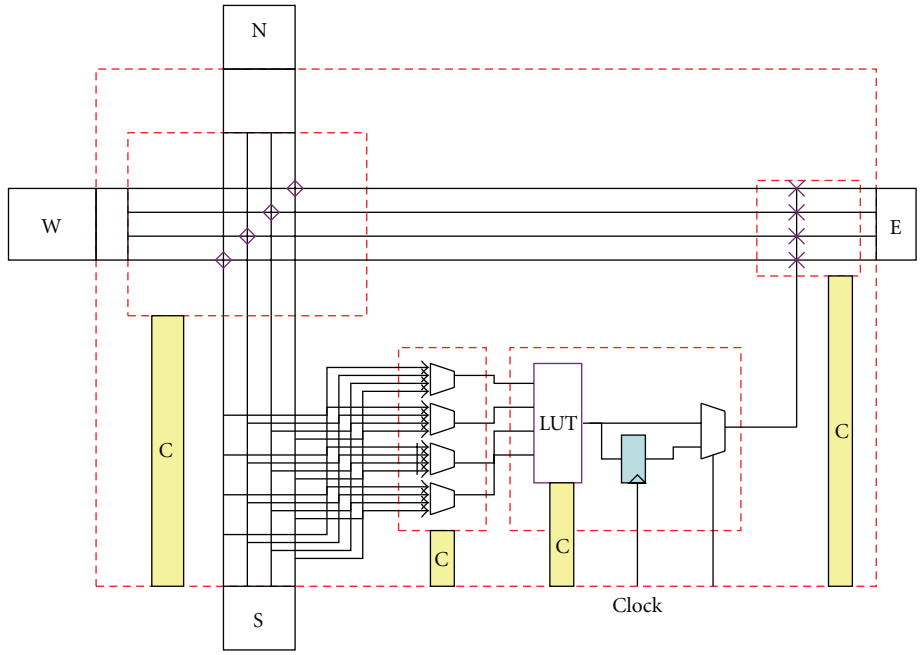

Figure 4: Structure of a basic cell (middle cell) within the RFU matrix.

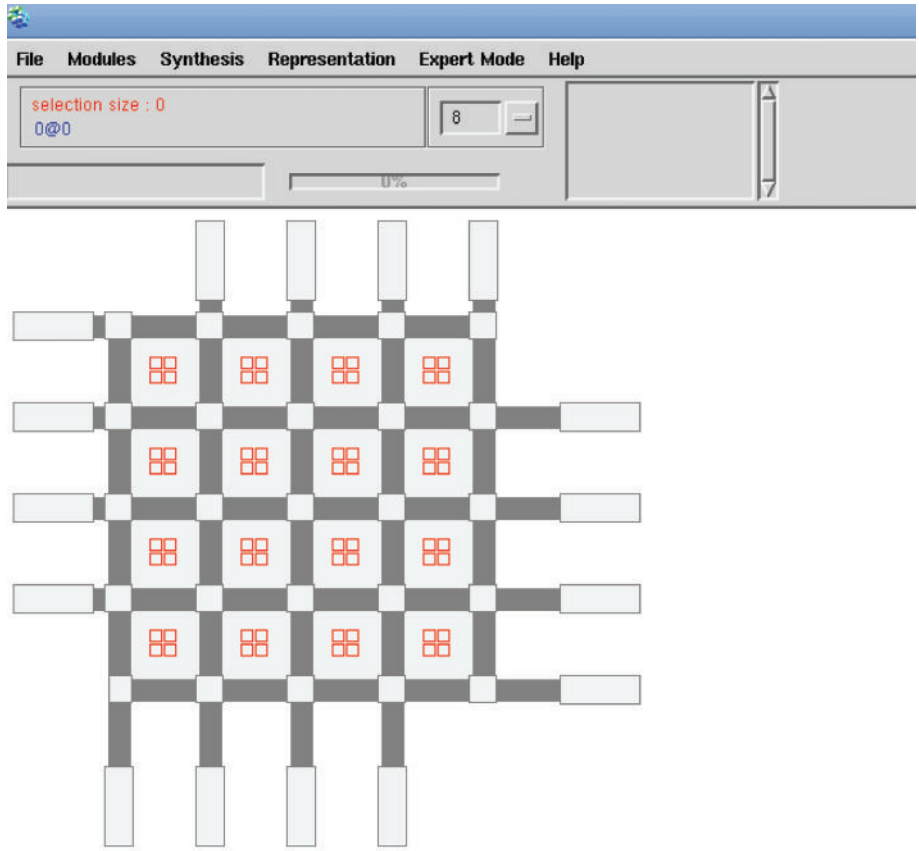

Figure 5: Whole view of the RFU.

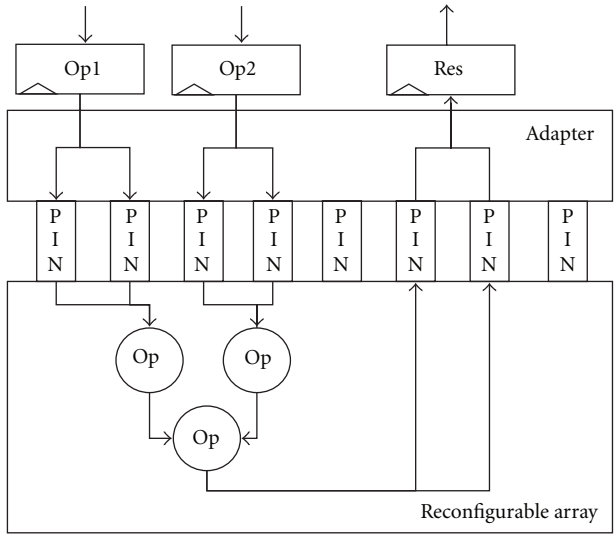

Figure 6: RFU is interfaced with the processor registers through an adapter.

implement a DFG to exhibit spatial execution. Another option consists in defining novel primitive operators. As an example, defining a multiplier instead of performing several processors instructions (addition, shifts, etc.) can make sense due to a high reuse rate.

In both cases, the RFU extends the instructions set.

Additionally, the underlying arithmetic can vary keeping the instructions set stable despite adding new variants for implementing these instructions. This goes through either a library-based design or dedicated synthesizers. Libraries are typically targeted to a reduced set of predefined macroblocks, and they are not easily customizable to new kinds of functions or use-cases.

We chose to focus on the second topics as this seems to carry extra added-value compared to classical flows, while reducing the need for a coding extra effort thanks to provided synthesis facility.

Figure 7 illustrates the Biniou behavioral application synthesizer. The optimizing context here is made up of typing as Galois Field GF16 values the two parameters. A so-called high-level truth table is computed per graph node for which values are encoded and binarized. The logic minimization [19] produces a context-dependent BLIF file.

This BLIF file is further processed by the Biniou P&R layer. As application is simple enough to keep the design flatten, no need exists for using a floorplanner. However, for modular designs, a TCG-based floorplanner [20] is integrated within Biniou.

Some constraints are considered, such as making some location immutable to conform to the pinout of the adapter (Figure 6) with regards to the ones assigned to the I/O of a placed and routed application (see Figure 8).

Once the P&R process ends, a bitstream is generated. Each element of the matrix both knows its state (used, free, which one out of N, etc.) and its layout structure. The full layout is gained by composing recursively (bottom up) these sub-bitstreams. An interesting point is that the bitstream structure can vary independently from the architecture by applying several generation schemes. As a result, in a partial reconfiguration scope, the students benefit from enriched architectural prospection capabilities. In the frame of the project an example of bitstream structure is provided by Figure 9.

3.5. Reports and Oral Defense. Students had to provide three reports, one per milestone. The reports conformed to a common template and ranged from 10 to 25 pages each. The last report embedded the previous ones so that the final document was made available straight after the project and students were given second opportunity to correct their mistakes.

Some recommendations were mandatory such as embedding all images as source format within the package, so that we could reuse some of them. As an illustration, more or less half of the figures in this papers come from students reports. The students had no constraints over the language but some of them chose to give back English-written reports. We selected some reports to be published on line as examples for next year students.

The last deliverable was made up of a report, working VHDL code and an oral defense. Students had to expose within 10 minutes, in front of the group, course teachers, and a colleague responsible for the "communication and job market" course.

Some students chose to center their defense around the project and the course versus project adequation, some others around the "product", that was their version of the processor.

3.6. Results Coming out of the Project. The simulation environment is ModelSim [21] as illustrated by Figure 10. The loader module—that loads up the program—was not provided but students could easily get one by simply reusing and adapting the generated test bench. Only one group out of five got it right.

This allowed to set a properly initialized state prior to execution's start. Of course, this was a critical issue, and students would have done well to fix it in an early stage as tracing values remained the one validation scheme. This was all the more important as the full simulation took a long time to complete and rerun had a real cost for students.

The simulation of the processor itself is time-affordable but the full simulation takes around 4 hours, including bitstream loading, and whole test bench program execution.

3.6.1. Optimizations. Students came to us with several policies to speed up the simulation. A first proposal is to let simulation happen at several abstraction levels, with a high rate of early error detection. Second, some modules have been substituted by a simpler version. As an example, by providing a RFU that only supports 8 bits ADD/SUB operations, the bitstream size is downscaled to 1 bit with no compromise on the architecture decomposition itself. This approach is very interesting as it confines changes to

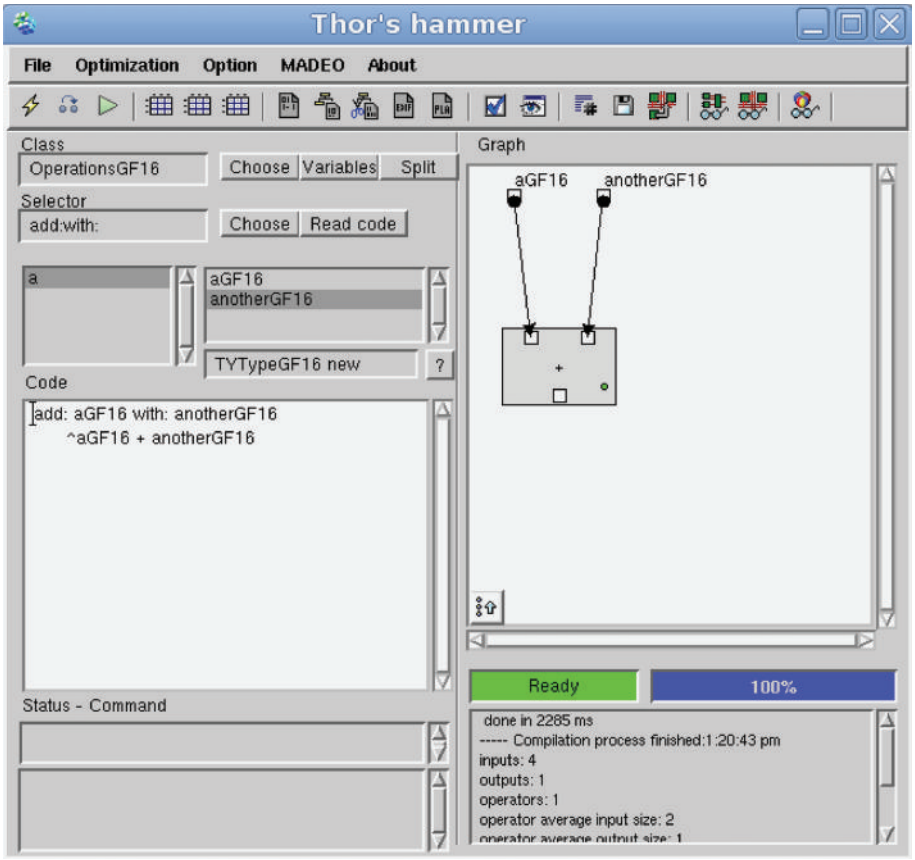

Figure 7: Specification of a GF16 adder.

the inside of the RFU while still preserving the application programming interface. In addition, it joins back the concern of grain increase in a general scope (i.e., balancing the computation/flexibility and reducing the bitstream size). Also this approach must be linked to the notion of "mock object" [22], software engineers are familiar with, when accelerating code testing.

Third, as the application is outputed as RTL code, the code can be used as a hard FU instead of using reconfigurable one. In this way, the students validated the GF-based synthesis. Grabbing these last two points, the global design can be validated very fast, being the scalability issue. This issue has been ignored during the project, but is addressed as the global design is given a physical implementation.

3.6.2. Analysis. The students sampling cannot be considered representative from a statistical point of view. However, some preliminary remarks seem to make sense.

Figure 11 shows that the deliverable 2 is harder to complete than the first one, but that more than half of the students got a success rate between 70% and 90%.

We chose to make students pair-achieve the project. In this way, beyond simply averaging the prerequisites matching so that the pairs are equally offered a chance to succeed, we intended to favor incidental learning as pointed out by chanck [23].

The increase of the standard deviation (Figure 12) highlights that one group failed in properly using the toolset (left border, Figure 11); another way to analyze this is that the toolset allowed to overcome the complexity of deliverable 2. Another interesting point is that the global understanding raises up during the full project, being the group who gave up after the first milestone (right border, Figure 11). The difference between regular and restricted lines is that restricted lines ignore this group. Finally, the standard deviation line points out that most homogeneous results came from integration, manual design of the processor, and last using the tool set.

4. Real Case Study

4.1. Experimentation Platform. The physical implementation was out of the scope of this project mainly due to some timetable hard constraints. Not all of the students proceeded in implementing their circuits. But the lessons we have learned are really inlined with the feedback we got from those of our students who applied for an internship in another lab.

The development platform we use for this demonstrator is a Virtex-5 FXT ML510 Embedded Development Platform from Xilinx.

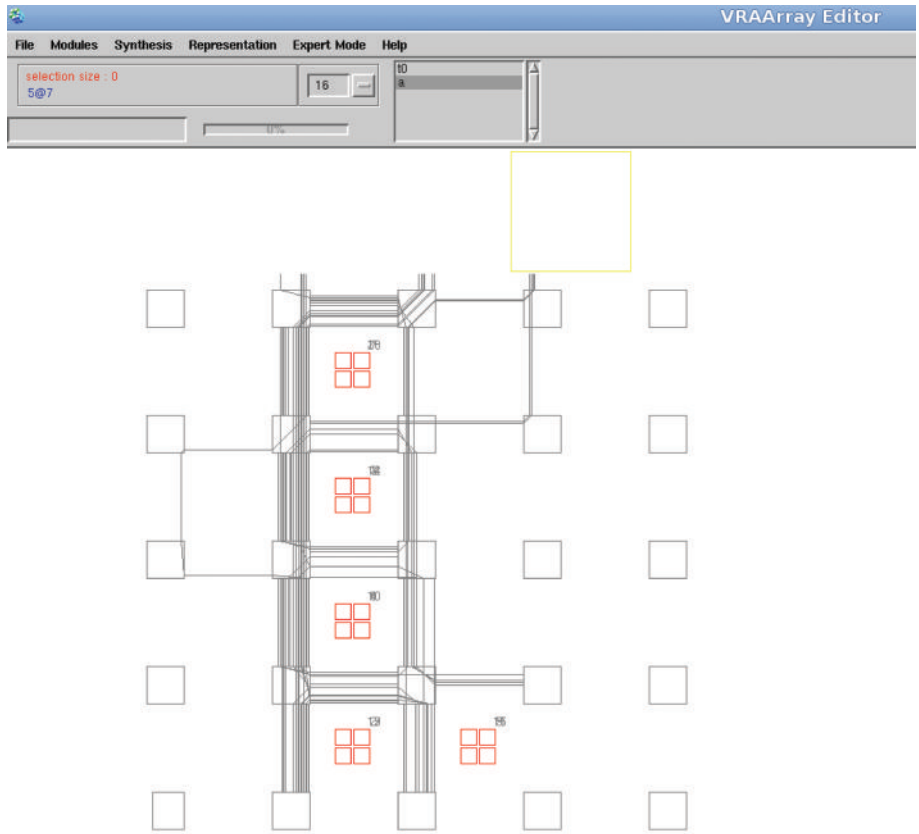

Figure 8: An application placed and routed over the RFU.

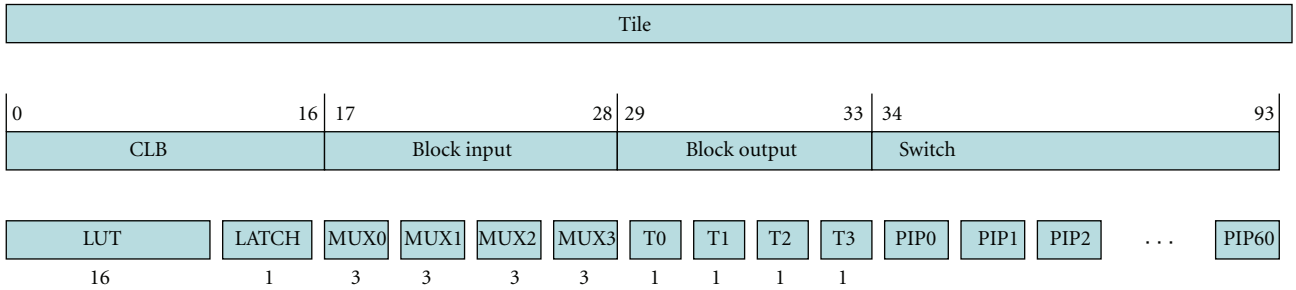

Figure 9: Example of a bitstream hierarchical organization.

4.2. Processor. A first noticeable difference between their former experience and the real case implementation lies in abandoning their hand written processor. Instead, the students had to instantiate a soft core.

4.2.1. Soft-Core. The soft-core processor is a Micro-Blaze and comes along with a full software environment.

4.2.2. Programmability. Not only, using this soft-core ensures a knowledge of state-of-the-art techniques but also it eases porting application. On the other hand, mixing soft and hard components within a single application is pretty clear to students who extended by hand the ISA of the toy processor.

4.2.3. Simulation. Another interesting features is the observability the simulation environment provides. On the opposite, gaining visibility during ModelSim simulation required to group/color/rename signals in the first processor. This is also important for performances extraction as scanning a *done* signal was used for time measurement.

4.3. Accelerator. The first version of the accelerator was a fine grained mesh. However, these architectures suffer from a long synthesis process, hence some coarser-grained architectures have been proposed in the literature to overcome this limitation. The second version reflects this architectural shift by exhibiting coarse-grained elements.

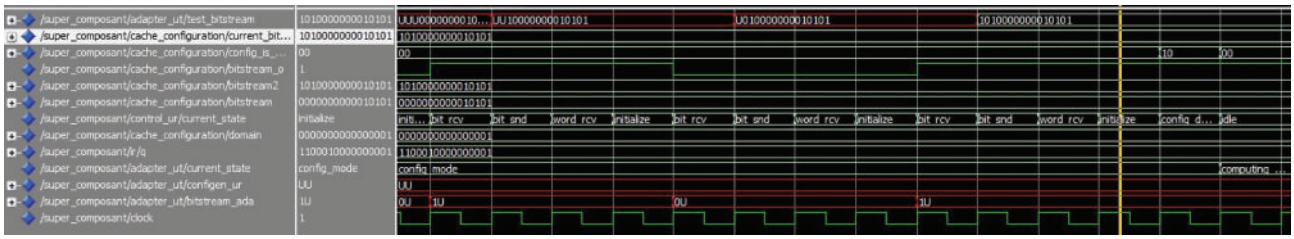

Figure 10: Modelsim simulation.

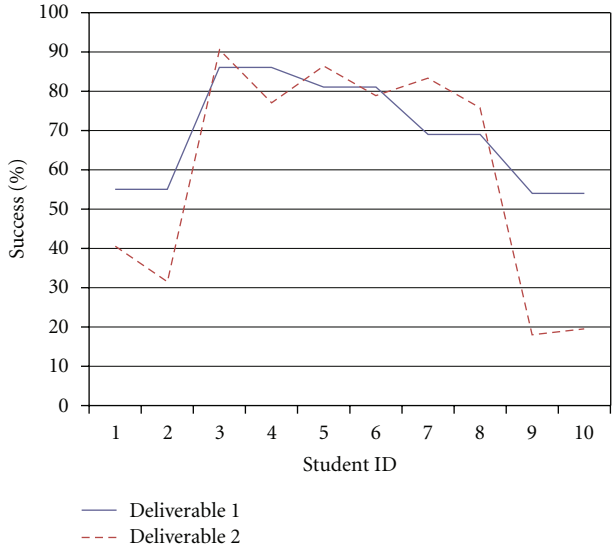

Figure 11: Results (% success) from milestones 1 and 2.

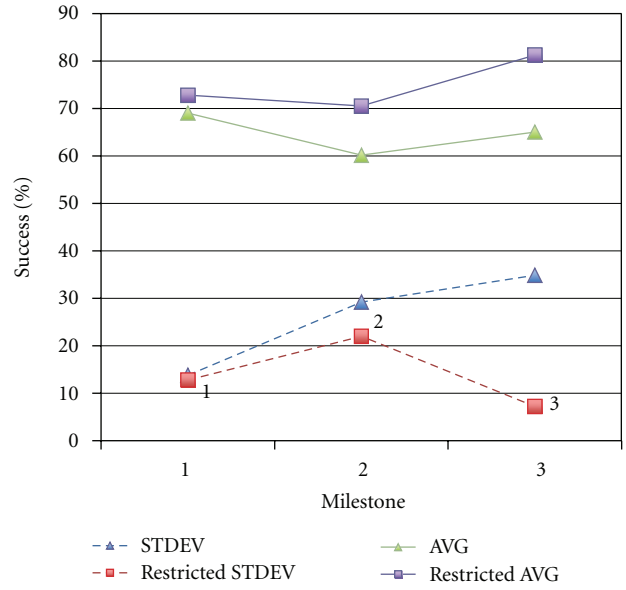

Figure 12: Global results.

4.3.1. Grain Considerations. In addition to these general considerations, students faced a performance issue when implementing fine-grained mesh over an FPGA. First, the synthesizer exhibited very low frequency. Secondly, the placement efficiency was unsurprisingly very poor.

At this point, another option emerged. A coarser grained architecture, inspired of PicoGA [24] but not as complex, was considered. The new architecture is organized as pipelined stripes. Logic elements are ALUs.

4.3.2. Impact over the Software Environment. The Biniou P&R relies on a Pathfinder [25] algorithm. The students got wrong configuration until we provided them a refactored version of the placer, that conforms to the stripe-based organization.

4.4. Processor-Accelerator Pairing. The third move between the project and the real case lies in changing the way the processor and the accelerator are connected to each other. The processor must support non blocking accelerated function calls which prohibits the former coupling scheme.

4.4.1. Coupling. Instead we asked the students to isolate the accelerator as an autonomous entity (coprocessor). The implementation was realized using FSLs, which is a classical option. Combining network concerns (FIFO, hand-shake, negotiation, etc.) with the simple adapter (Figure 6) made FSL a very natural concept to computer engineers.

4.4.2. Timing Constraints. Gaining high performances requires to force constraints when calling the ISE synthesizer.

4.4.3. Layout. Figure 14 illustrates a layout of a coarse-grained reconfigurable architecture (see Figure 13) acting as an accelerator for a Micro-Blaze.

4.5. Manual Domain Space Exploration. Once acquired a sound knowledge of the domain (architecture, platform, tools), students started to address Domain Space Exploration (DSE). First, this stage was kept manual still following the precept of "simplicity" and "just-fit approach".

4.5.1. Considered Cases. The first dimension for variability is the matrix sizing. Several instances have been designed ($5 * 2$, $5 * 4$, $5 * 10$, $40 * 40$). The second axis is the reconfiguration grain. For a similar matrix, several instances are issued with a different partial reconfiguration page size

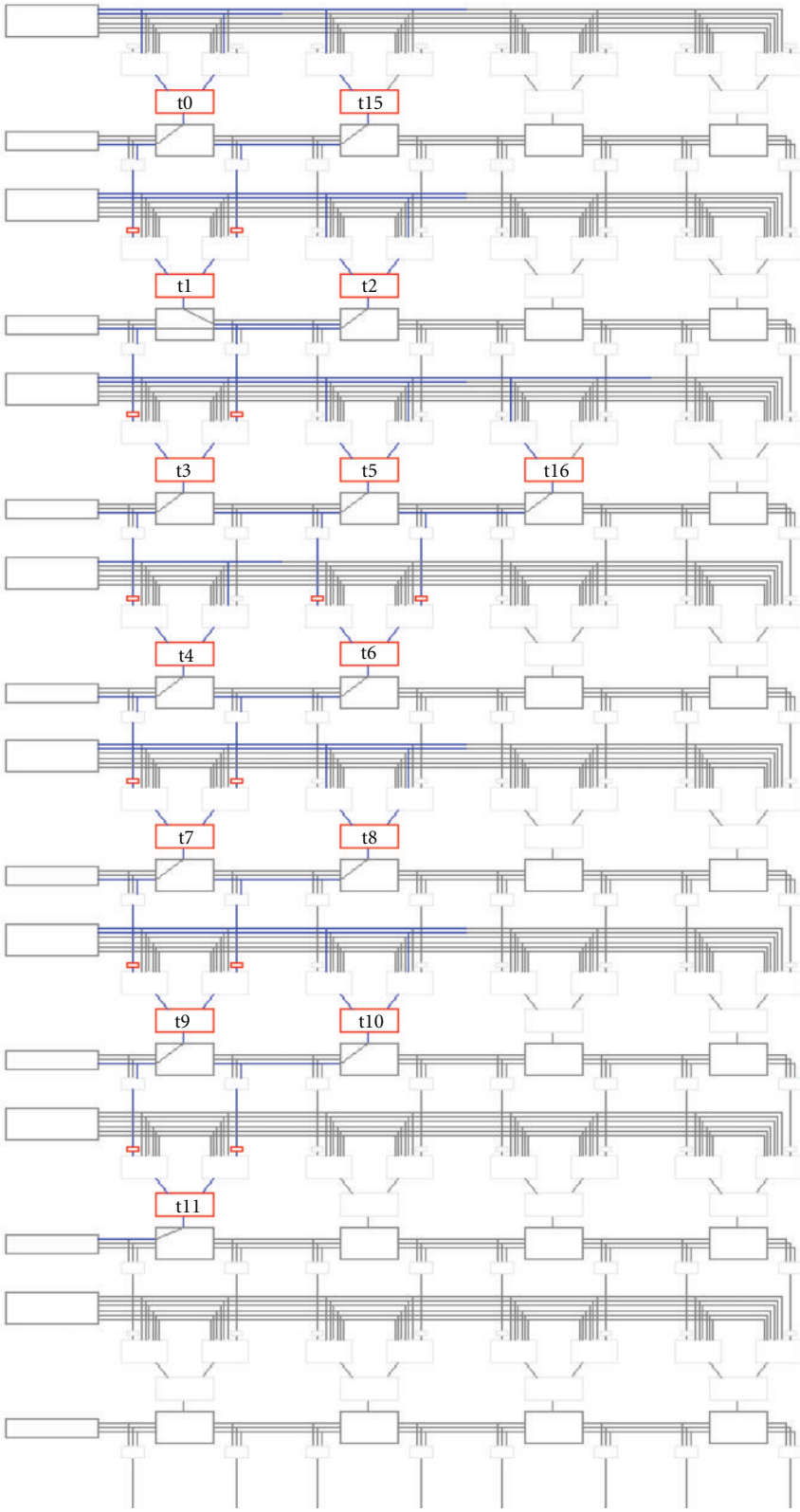

Figure 13: The view Biniou provides over a Coarse Grained Reconfigurable Architecture under use.

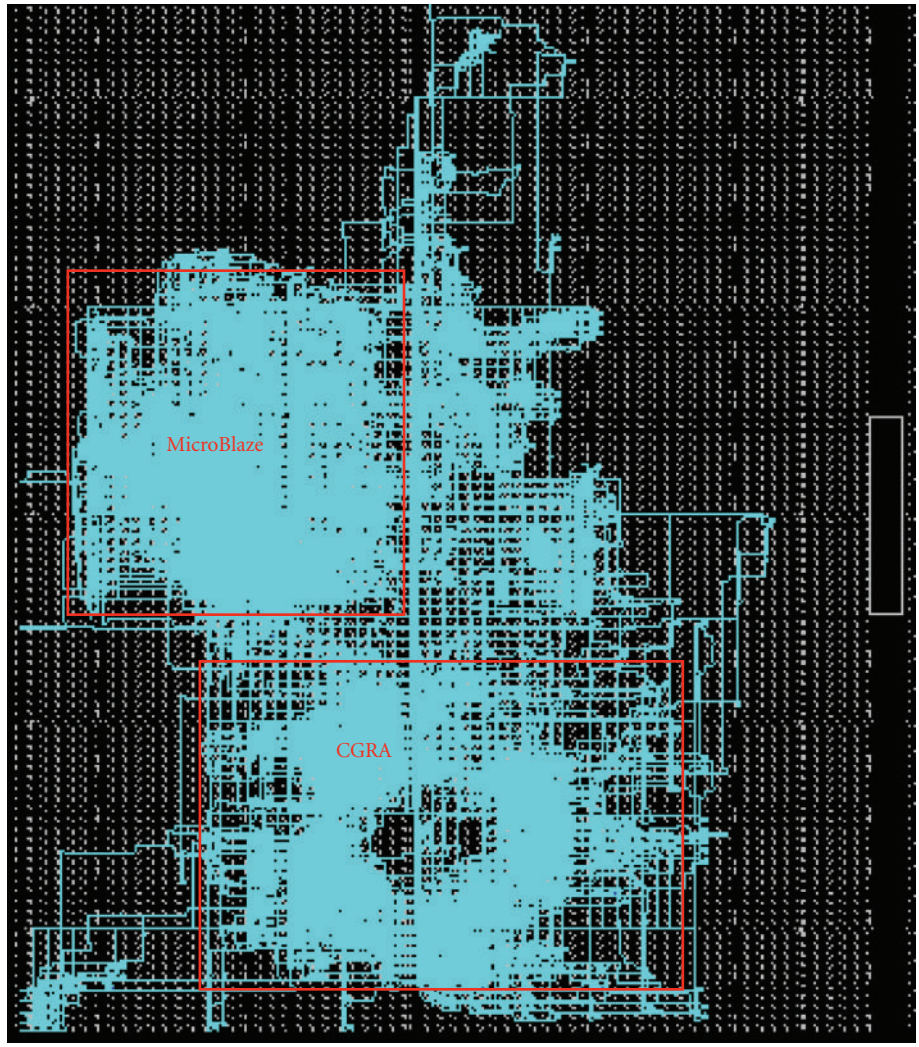

Figure 14: A resulting layout with a MicroBlaze connected to a coarse grained reconfigurable architecture through FSL.

each. Then, the last measured impact is related to the number of configuration contexts.

4.5.2. Metrics. It is important to measure the quality of solutions, especially the specific amount of a certain resource and architectural solution needs. Examples of such resources would be area, time, or memory storage.

4.5.3. Speed up Measurements. Computing a speed up requires two things: first, measuring an execution time, then comparing versus a reference execution time. A nonobvious point to students is how to make a fair measurement. As an example, the coarse grained architecture may affect the processor's frequency. Hence, two speed-up must be analyzed. The first one makes use of a pure software execution time whereas the second one considers the execution time of a full software variant running on a processor/coprocessor architecture.

Of course, this speed-up remains highly application dependent. A FIR execution has been considered as this was enough for teaching purposes; as an example, the speed up factor for an FIR with 8 coefficients and 6500 data hits 31.6.

4.6. Towards an Automatic DSE. Creating spike solutions helps to figure out answers to tough technical or design problems. A spike solution is a very simple program to explore potential solutions. Students are encouraged to design spike solutions to stress some hypothesis before any announcement. The spike must be built only to address the problem under examination and ignore all other concerns. The goal is to reduce the risk of a technical problem or to increase the reliability of their feelings and estimate.

Spike solutions are applied for grabbing synthesis information and scripting the design tool suite.

4.6.1. Synthesis Report Analysis. The synthesis reports provide a set of information for quality measurement. The first metric is the amount of used resources. This appears as used Luts/FlipFlops pairs, plus internal fragmentation. The students have no control over the algorithms, and some

Table 2: Sizing matrix impact over frequency, resources, and synthesis time.

Dimensions	5×2	5×4	5×10	40×40
Freq.	102.8	102.8	101.7	53.8
# Slices	492	969	2397	4999
Cpu	35	45	100	47355

Table 3: Multiple context impact over frequency, resources, and synthesis time.

dimensions	# Contexts	Freq.	# Slices	Cpu
5×4	1	102.8	969	45
	2	99.8	1105	55
	5	99.8	1423	83
	10	98.0	2184	128
5×10	1	101.7	2397	100
	2	99.7	2747	137
	5	100.2	3522	238
	10	99.2	5415	468

results are difficult to analyze. As an example, in Figure 12, the depopulated center of the coprocessor may reflect the torus nature of the coarse-grained architecture. Nevertheless, stressing the constraints change the topology at the expense of a frequency scaling down.

Frequency is the second metric that the students concentrated on, all the more so as violations can occur which invalidate the full design.

The students knew how to find the relevant information. Going further though would have required to write a parser, then to extract scoring out of generated reports. This would be an interesting step forward command/scoring the tool suite.

4.6.2. Xilinx SDK Scripting. In order to detect the system files that are involved in a potential scripting, a first design is done through the user interface. Then, all modified files are reported, and a `diff` command is issued to let the students precisely locate internal changes. Then, code generation happens and recompiling the projet results in refactoring the design.

4.6.3. DSE Results. Tables 2 and 3 summarize for illustration purposes some of the DSE results the students collected.

5. Conclusion

This paper presents an experience report of course setup for master students discovering configware. This course tends to overcome the information pick-up limit to offer a real knowledge to students. This goes through manual design of toy examples that forces students to emphasize simple designs. Once acquired such an insight, commercial design suite are introduced for up-to-date training. Beside, research tools support complex tasks such as reconfigurable platform design, and DSE in a general way.

5.1. Forces. One interesting point regarding this project lies in the change in the students feeling. When we presented at the first time the project, they thought they would never complete the goals. After the first milestone, one group gave up to avoid paying the over due penalty and bounded their work to the first deliverable. They finally reached 7 points out of 20. The other groups faced the challenge and discovered that the key issue lies in getting proper tools to free oneself from manually developing both architectures and application mapping. The final results were very likely acceptable and we collected several working packages.

With this experience in mind, students are now ready for entering a very competitive job market. They share a deep understanding of both hardware design over reconfigurable architecture, microprocessors, reconfigurable cross integration, and tools and algorithms development.

This effect has been clearly pointed out when migrating from a toy example to real design environment. This move has offered several dimensions for DSE: reconfigurable unit grain, processor, coupling, and so forth.

5.2. A Very Positive Feedback. The actual success of this teaching experience lies in the highly efficient learning curve we noticed when students started to experience Xilinx design Kit. Obviously, neither the test bench examples we first provided nor the students population size are sufficient to practice real metrics-based measurements. Exploring the benefits of this approach (e.g., measuring speed-up) requires an easy path from a structured programming language such as C to the processor execution. Hence, the application's change would carry no need for hand-written adjustments. From our point of view, such an add-on in the project would be a fruitful upgrade to the course, and would spawn new opportunities for cross H/S expertise; keeping in mind that the reconfigurable computing course intends to get out with highly trained students sharing skills in both area.

Developing a small compiler was out of the scope of this project due to some timing constraints, but remains one hot spot to be further addressed. This could benefit from some Biniou facilities such as the C-entry synthesizer.

An open option is then to benefit from another course and invited keynoters to fulfill the prerequisites so that adapting/developing simple C parser becomes feasible in the scope of our project, at the cost of around an extra week.

5.3. Going Further. The second very positive feedback we got is that students are ready for new experiences, even with research tools that do not offer the same QoS than commercial design suite. This offered a path to reconfigurable units design with a full high level synthesis support.

Now, an interesting option is to introduce more efficient RFU, by generating coarse-grained architectures that support virtualization. Applying virtualization techniques allows to leverage some well-known limitations of reconfigurable architectures: limited amount of resources, lack of high-level programming model, and nonportability of bitstream.

Biniou offers a smart framework for design-space exploration of reconfigurable IPs. Fine-grained architectures offer

a nice teaching testbed, but shifting from fine to coarse-grained architecture rather make sense for current technologies. This brings no extra cost as Biniou fully supports this architectural scheme. Instead, this carries extra value as it underlines the resulting shift from "hardware" netlist design to "software" operation graphs editing.

Ensuring students will get the appropriate strength to self-adapt to such changing environment remains our educational goal. Once done, hard-soft co-design and applicative needs adequation driven platform development are on their way.

References

[1] J. M. Filloque, E. Gautrin, and B. Pottier, "Efficient global computations on a processor network with programmable logic," in *Parallel Architectures and Languages Europe (PARLE '91)*, pp. 69–82, 1991.

[2] L. Lagadec and B. Pottier, "Object oriented meta tools for reconfigurable architectures," in *Reconfigurable Technology: FPGAs for Computing and Applications II*, Proceedings of SPIE, pp. 69–79, November 2000.

[3] L. Lagadec and D. Picard, "Software-like debugging methodology for reconfigurable platforms," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS '09)*, May 2009.

[4] S. R. Alpert, K. Brown, and B. Woolf, *The Design Patterns Smalltalk Companion*, Addison-Wesley, Boston, Mass, USA, 1998.

[5] E. M. Sentovich et al., "Sis: a system for sequential cirquit synthesis," Tech. Rep. UCB/ERL M92/41, Department of Electrical Engineering and Computer Science, Berkeley, Calif, USA, May 1992.

[6] "Madeo-web, the madeo+ web version," http://stiff.univ-brest.fr/MADEO-WEB/.

[7] P. M. Athanas and H. F. Silverman, "Processor reconfiguration through instruction-set metamorphosis," *Computer*, vol. 26, no. 3, pp. 11–18, 1993.

[8] R. Razdan, K. S. Brace, and M. D. Smith, "PRISC software acceleration techniques," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 145–149, October 1994.

[9] T. J. Callahan, J. R. Hauser, and J. Wawrzynek, "Garp architecture and C compiler," *Computer*, vol. 33, no. 4, pp. 62–69, 2000.

[10] F. Campi, R. Canegallo, and R. Guerrieri, "Ip-reusable 32-bit vliw risc core," in *Proceedings of the 27th European Solid-State Circuits Conference*, pp. 445–448, 2001.

[11] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. L. M. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN polymorphic processor," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1363–1375, 2004.

[12] V. Angelov and V. Lindenstruth, "The educational processor Sweet-16," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 555–559, August 2009.

[13] R. Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger, "Using the KressArray for reconfigurable computing," in *Configuralble Computing: Technology and Applications*, Proceedings of SPIE, pp. 150–161, November 1998.

[14] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Matt, and R. R. Taylor, "PipeRench: a reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, 2000.

[15] J. Becker and M. Vorbach, "Coarse-grain reconfigurable XPP devices for adaptive high-end mobile video-processing," in *Proceedings of IEEE International SOC Conference*, pp. 165–166, September 2004.

[16] G. Lu, M. hau Lee, H. Singh, N. Bagherzadeh, F. J. Kurdahi, and E. M. Filho, "Morphosys: a reconfigurable processor targeted to high performance image application," in *Proceedings of the International Symposium on Parallel and Distributed Processing*, pp. 661–669, 1999.

[17] F. Campi, A. Deledda, M. Pizzotti et al., "A dynamically adaptive DSP for heterogeneous reconfigurable platforms," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 9–14, April 2007.

[18] C. Mucci, C. Chiesa, A. Lodi, M. Toma, and F. Campi, "A C-based algorithm development flow for a reconfigurable processor architecture," in *Proceedings of the International Symposium on System-on-Chip (SoC '03)*, pp. 69–73, November 2003.

[19] L. Lagadec, B. Pottier, and O. Villellas-Guillen, "A lut-based high level synthesis framework for reconfigurable architectures," in *Domain- Specific Processors : Systems, Architectures, Modeling, and Simulation*, S. Batttacharyya, E. Deprettere, and J. Teich, Eds., pp. 19–39, Marcel Dekker, 2003.

[20] J. M. Lin and Y. W. Chang, "TCG: a transitive closure graph-based representation for non-slicing floorplans," in *Proceedings of the 38th Design Automation Conference*, pp. 764–769, June 2001.

[21] "Modelsim," http://www.model.com/.

[22] D. Picard and L. Lagadec, "Multilevel simulation of heterogeneous reconfigurable platforms," *International Journal of Reconfigurable Computing*, vol. 2009, Article ID 162416, 12 pages, 2009.

[23] R. Schank, Tech. Rep., Institute for the Learning Sciences (ILS), Northwestern University.

[24] A. Lodi, M. Toma, and F. Campi, "A pipelined configurable gate array for embedded processors," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '03)*, pp. 21–29, February 2003.

[25] L. McMurchie and C. Ebeling, "PathFinder: a negotiation-based performance-driven router for FPGAs," in *Proceedings of the 3rd ACM International Symposium on Field-Programmable Gate Arrays*, pp. 111–117, February 1995.