Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 923415, 11 pages
doi:10.1155/2009/923415

Research Article

Experiencing a Problem-Based Learning Approach for Teaching Reconfigurable Architecture Design

Erwan Fabiani[1,2]

[1] *Université Européenne de Bretagne, France*
[2] *Université de Brest and CNRS, UMR 3192 Lab-STICC, ISSTB, 6 Avenue Victor Le Gorgeu, 29200 Brest, France*

Correspondence should be addressed to Erwan Fabiani, fabiani@univ-brest.fr

Received 31 December 2008; Revised 14 June 2009; Accepted 28 August 2009

Recommended by Peter Zipf

This paper presents the "reconfigurable computing" teaching part of a computer science master course (first year) on parallel architectures. The practical work sessions of this course rely on active pedagogy using problem-based learning, focused on designing a reconfigurable architecture for the implementation of an application class of image processing algorithms. We show how the successive steps of this project permit the student to experiment with several fundamental concepts of reconfigurable computing at different levels. Specific experiments include exploitation of architectural parallelism, dataflow and communicating component-based design, and configurability-specificity tradeoffs.

1. Introduction

When teaching "reconfigurable computing" (RC), a coarse distinction should be made between two different aspects:

(1) reconfigurable architecture programming: knowing the fundamental steps for programming, being able to use languages and environments to produce a configuration, usually targeting an FPGA; these skills can be divided into technical and conceptual ones;

(2) reconfigurable architecture design: understanding and designing the "reconfigurable" nature of an architecture.

The difficulty for the first aspect is to manage the distribution and the interactions between the technical and conceptual parts. If the focus is mainly directed at technical skill development, then students would not have the ability to adapt their experience to other languages and environments and to reuse programming strategies.

Conceptual aspects include using programming models to ease the design of an ad hoc component structure for a specific application and to be able to apply parallelization algorithms to improve performance. If the conceptual aspects are favored, then the students eventually would not see their advantages, since they do not experiment the difficulties of programming and achieving performance objectives without it. Moreover, a lack of technical skills in their curriculum may be a disadvantage for employment. Also, technical skills may be necessary to optimize a program issued from a high-level synthesis tool.

The second aspect induces the need for generic tools to design, program, and simulate a reconfigurable architecture. If focus is put on physical design (basic configurable blocks and routing channels), there are academic frameworks like VPR [1] or Madeo [2] that permit definition of a reconfigurable architecture of various granularity, with generic synthesis, place and route tools that act on any design description. However such a design level may be too much advanced and difficult to be comprehended by software CS students. Moreover, it does not answer the question: how to teach design of a parallel architecture (and not how to teach expressing an architecture specification as a synthesizable description)?

The course presented here aims to be at the interface between RC programming and RC design, without being as exhaustive as other courses (e.g., [3, 4]). It gives students some basic skills and knowledge on reconfigurable computing, in a limited time, by designing a reconfigurable

coarse grain datapath architecture for an application class of image processing algorithms and implementing it on an FPGA. The design methodology is similar to intermediate level component design [5].

Our pedagogical approach relies on "problem-based learning" [6]. The goal is that skill and knowledge acquisition should be induced from the realization of a concrete project by the students.

The course context and pedagogical objectives are detailed in Section 2. Section 3 is focused on the "teaching by project" approach applied to our course. Sections 4, 5, 6, 7, and 8 follow the design methodology, respectively, defining the application model, deducing an architectural model, implementing it on FPGA, and reusing the architecture design pattern via object modeling.

2. Course Context and Objectives

2.1. Course Context. This project is included in a CS master course (1st year) on parallel architectures. Scope of this course includes knowledge of basic parallel architecture models, ability to identify in an application the different kinds of potential hardware parallelism, programming for various parallel computing models, and experiencing the design of a specialized, reconfigurable or programmable architecture, using automated methods for regular architecture synthesis.

Usually, practical work is based on lab sessions with a limited focus, followed by an evaluated project with a "closed subject": the teacher gives a very detailed specification which should plan all aspects, and oral explanation is provided to help the students to understand the subject. There are two main disadvantages of this kind of project: students do not participate in the design and the project could lead to "making the students do exactly what the teacher has defined and potentially penalising the students for not precisely adhering to this."

However, considering an "open subject" project instead induces other difficulties. An "open subject" means that a significant part of the design of the solution is a result of the student's work; thus the subject is opened to various solutions. It is difficult to define an adequate level of difficulty of the project, and the risk of "off topic" answers is increased. It is also difficult to manage evaluation criterion.

Consequently, in order to facilitate knowledge and skill acquisition by students in a limited time, most of the course is based on a participative project (problem-based learning) where students learn by participating to design a solution to a problem. Problem-based learning is known to emphasize not only the final solution but also the method used to design this solution [7].

The project begins with a subject which is opened to various design alternatives, instead of having a highly detailed and structured subject that would make students just implement and not design a solution. Even if the teacher knows which design approach may be the best one to study, it is important that the students feel that they reach a solution by themselves and by cooperation. The teacher helps and directs students to express complete specifications. This kind of project also facilitates the activation of previous knowledge (previous courses and self acquired knowledge), helping to place the subject in a more general context.

Since 2006, practical work has been based on design implementation using FPGAs. We have chosen the Handel-C language [8] for several reasons. First of all, in the student's curriculum, a previous course on distributed algorithms, introducing synchronous and asynchronous modeling, uses a CSP- (Communicating Sequential Processes-) based language (*kroc* [9]) for specifying and simulating algorithms. Students are by this way familiar with basic programming primitives of a CSP model: system organization with communicating processes, expression of parallelism (*PAR statement*), reading (*? statement*), and writing (*! statement*) in synchronous communicating channels. Secondly, the high-level language structure and the C syntax make the apprenticeship of Handel-C faster for CS students than standard hardware design languages.

Concerning the reconfigurable computing platform, we have chosen the RC10 board from Celoxica [10], which includes a Xilinx Spartan 3L-1500 (containing 26 624 LUTs and flip flops). VGA output and camera peripherals included in the RC10 board permit easy development of real-time video image processing, which is one of the major application domains of reconfigurable computing [11]. Although the course is not image processing oriented, the requirements for the hardware platform are similar to [12]: those kinds of applications are a motivating context for students.

Concerning the toolset used, it is divided in two parts.

(i) The DK design suite [13] provides the Handel-C language and EDA toolsuite for design capture, compilation and synthesis of a design into an EDIF netlist. It permits the compilation, and synthesis of an Handel-C program into an EDIF netlist. Prebuilt libraries are available to abstract board drivers (camera, VGA display, flash memory, etc.).

(ii) The Xilinx ISE design suite [14] is used to generate a bitstream from the EDIF netlist, through map, place, and route processes.

2.2. Project Objectives. The pedagogical objectives of this course are:

(i) to improve technical skill in programming FPGA and experiment the development cycle of a configuration: high-level description, synthesis, map, place, route, bitstream generation; simulation is not addressed in this course and is studied during the 2nd year of the master;

(ii) to experiment the different concepts of architectural parallelism;

(iii) to experiment the partitioning of an architecture into various components and their design, to feel the tradeoff and choices between flexibility, specificity, and efficiency;

(iv) to understand the advantages of object modeling for productivity;

(v) to look at solutions fixing a problem, to present them, to compare them, and to select the best solutions relating to fixed constraints.

Consequently, the choice of the project subject has been a reconfigurable architecture that exploits data flow parallelism and control parallelism, as shown in the following sections.

3. Course Development

3.1. Global Scheduling. The project has been introduced, defined, and carried out in several sessions.

(i) The preliminary courses are two lectures (2 hours) on parallelism and parallel architectures and one lecture on FPGA, reconfigurable architecture, and Handel-C.

(ii) The first supervised practical work (2 hours) aims to highlight the main problems or questions. It is based on the initial subject and is composed of the following steps:

(1) individual reading and analysis of unknown notions and most difficult points;
(2) group working (5 or 6 students): merge of individual reflections, mutual help based on personal experience and knowledge, and selection of most important points to be solved in the project; each point is categorized into lack of knowledge, technical skill, or design reflection;
(3) the whole group questions are orally discussed and classified in similar themes, in order to arrive at a consensus on the main tasks: points to be solved or knowledge to be acquired (six tasks);
(4) the course ends up at assigning one task to each student of a group, who must prepare a presentation about it for the next session.

(iii) The second directed work course (2 hours) is a synthesis of the student's work:

(1) presentation of the student's assigned work: synthesis on a theme or reflection on a problem;
(2) critical analysis of each presentation, questions and answers;
(3) the session ends up at a consensus for solving each point;
(4) following the session, the teacher edits a synthesis document of all discussed points, that becomes the specification reference for developing the project.

(iv) Project labs and tutorials permit the student to take the environment in hand and to acquire the basic technical skills needed to begin the project:

(1) Lab 1 (2 hours): first contact with language and environment, exercises for video acquisition and display, and pipeline for simple pixel processing;
(2) Lab 2 (2 hours): exercises on flow parallelism and data parallelism;
(3) Lab 3 (2 hours): design of a FIFO unit for moving window (neighborhood).

(v) After students have completed the project, they are required to complete an anonymous questionnaire mainly focused on their evaluation of the "problem based learning" approach used. This questionnaire permits to have a student advice on various aspects of the project. For instance, questions can be related to group management, evaluation of the contribution of a student in his group, and evaluation of the benefits of the pedagogical method for knowledge acquisition.

3.2. Initial Subject. The submitted project addresses the design of a coarse grain datapath reconfigurable architecture well suited for a family of image processing algorithms and that can be implemented on an FPGA. Unlike most image processing FPGA implementations (as an example, implementations in Handel-C [15, 16], or for teaching purpose [12]), the architecture is "reconfigurable" in the way that once the bitstream is loaded, a configuration (list of symbols) is read from the flash memory and defines the behavior (taken in a limited set of function) of the units involved in the computations. It is similar to the "configurable window processor" architecture of Torres-Huitzil and Arias-Estrada [17]. However it remains at a lower level than SIMD instruction customization [18]. This configuration is set by the user prior to FPGA reconfiguration and is defined as a list of symbols.

This project was conceived for teaching purpose, but it is not far from real needs. For instance, designing such architecture permits to have a single bitstream that can be used to compute various image processing applications. It is comparable to IP core; or processor core hence it is not specific neither programmable but reconfigurable. Consequently an end user with no knowledge in FPGA programming can easily specify the behavior of his circuit, given the restricted application model defined. This configuration model permits to avoid synthesis, place, and route process for each specific behavior implemented.

On the other hand, such an architecture would obviously be both more resource consuming and both have lower clock frequency than a specialized design. However those disadvantages should be moderated by the real time video processing context: having the higher clock frequency is not the goal; the circuit should just be fast enough to have a satisfying frame rate. Moreover, camera and display peripheral drivers use functions that limit the maximal reachable frequency. Thus the real tradeoff can be considered to be between flexibility and area.

The application-class target of the architecture is spatial filtering for image processing where the new value of a pixel is computed depending on the values of its neighbors (here we consider a 3×3 neighborhood). Typical applications of this kind are morphological operators, convolution filters, and image segmentation, detailed in Section 4.

Concerning input/output timing constraints, at each system cycle an input pixel is read and an output pixel is written. A system cycle requires one or more clock cycles. The latency for computing a pixel is not constrained: this allows maximal level of pipelining. Moreover the architecture must include the memory resources needed for pixel buffering.

3.3. Student's Work. The first session ended up with the list of following points to study:

(1) architectural model: partitioning the architecture into units (computation units, I/O, configuration controller, memory unit);

(2) application model: doing a research on application model (context of use, typical convolution matrix);

(3) application/architecture appropriateness: enumerating the needed functions on the architecture to implement the different kinds of applications and giving typical examples;

(4) programming model: defining the format used to specify the behavior of the computation unit, and how the computation module will be configured;

(5) memory component of the computation module: I/O, size, structure;

(6) typing: size of types in the computing module and in the control unit.

Consequently, each student has to study one specific point for the second session. During this session, students present their ideas. For each point, ideas are compared, discussed, and a choice is made, with the help of the teacher if needed. These ideas and choices are the basis for the synthesis document edited by the teacher, which the students must follow for the implementation of the project. The following Sections 4 and 5 present a synthesis of the results of those studies, respectively, for the application model (point 2) and the architectural model (points 1,3,4,5).

4. Application Model

The application model includes morphological, filtering, and segmentation operations.

Morphological operations are, for instance, the following:

(i) dilatation and erosion, that, respectively, assign to the center pixel the maximum or the minimum of the significant neighboring pixel values identified by a mask;

(ii) pattern recognition, that recognizes a specific pattern, given a mask that specifies if a pixel must be black, white, or could have any value, in a binary image.

Filtering operations are based on convolution (sum of products with varying coefficients), eventually followed by a division.

(i) Smoothing filters are used for blurring and noise reduction. The most known filter is the mean filter (coefficient 1 for all pixels and division by 9) or the Gaussian filter.

(ii) Sharpening filters accentuate the details of an image contour (in fact a subtract of the blur image from the initial image). As the result can be negative, it is eventually necessary to add a fixed value or to take the absolute value.

Image segmentation operations enhance characteristic areas of the image, for instance, edge detection, Prewitt filter, Sobel filter, and Laplacian filter. The computed values may be negative; then an absolute value operation is needed.

Moreover, several operations can be sequenced, to accentuate the effect by repeating the same processing, or to compose a new processing, for instance an opening (one erosion followed by dilatation, to diminish noises) or a closing (one dilatation followed by an erosion, to smooth the form contours).

5. Architectural Model

The basic element of the architecture is a module. A module has one input and one output that are streams of pixel values: pixel coordinates are not transmitted as we assume that pixel order follows a raster scan order (column by column and line by line). The latency between a pixel input and the corresponding pixel output depends on the image width and the number of parallel components that are crossed by a pixel. In cruising speed, a module receives and sends one pixel at each clock cycle.

A module is composed of various units in a pipeline; each of them implements a different class of functionality, each being "configurable"; that is to say its computation behavior is specified by the user and setup before the computations begin. All units operate in parallel and simultaneously, on different instances of pixels in the stream. Thus, data flow parallelism is fully exploited. However the clock cycle is limited by the function with the longest delay. For a more efficient architecture, function delay estimation and node fusion would be necessary.

Modules are designed to be easily interconnected to each other by abutment in order to implement a more complex processing. Scalability is guaranteed by the fact that linking two modules would not increase the critical path, similarly to regular systolic arrays. The set of configurable units defines the programming model: it should be sufficient for describing the implementation of the chosen application model and can be extended.

Figure 1 presents the integration of 3 modules in the experimental context: pixels are generated by the camera controller and sent to the first module. Several modules are chained, using the same interface (one input pixel and one output pixel). The last module sends the output pixels to a frame buffer controller, which store in an embedded block ram the content of a complete frame and display it via the VGA output.

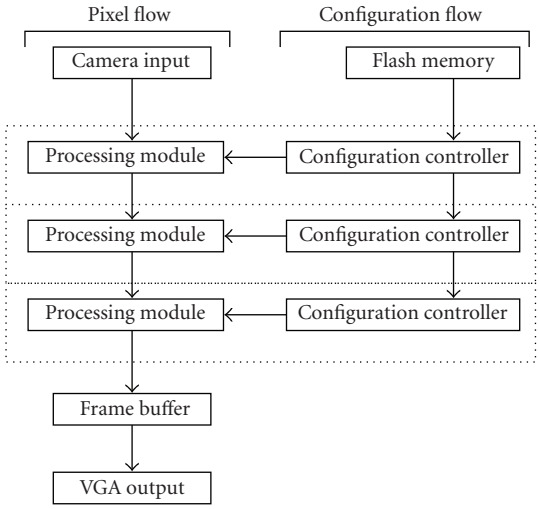

Figure 1: Integration of an array of 3 processing modules in the experimental context.

A configuration controller is used to configure the architecture. In case of several modules, controllers are chained in order to avoid the delay increase and simultaneous access problems that a parallel access of controllers to flash memory would induce. After reading its own configuration, the controller of the first module continues reading configuration file and sending configuration stream to next controller. The architecture is configured only once after the bitstream is loaded, but run-time reconfiguration would also be possible.

An image processing module is composed of various units (see Figure 2): units have a fixed behavior or have configurable parameters or operators. The various units that form the module have been defined by analyzing the application model and extracting common functionality patterns.

(i) Unary identical preprocessing units: they input one pixel and output one pixel (see Figure 3). These units perform an identical processing on all pixels, for instance, thresholding (with a parameter), inverse video, and format conversion.

(ii) Moving window unit: it inputs one pixel and outputs 9 pixels (see Figure 4). This unit memorizes the pixels in order to output a neighborhood at each clock cycle. It is a classical "moving window" approach, where pixels are stored as long as needed (a pixel is received one time and is used for 9 computations). So the moving window corresponds to a $3 * 3$ matrix that regularly goes over the image, left to right and top to bottom. Concretely this unit is made of a shift register of size $2 * W + 3$, W being the number of pixels in an image line. This unit has an initial latency of $W + 3$ clock cycles, that is to say the number of cycles between the input of the first pixel and the output of the first pixel neighbor. Image borders have a configured fixed value.

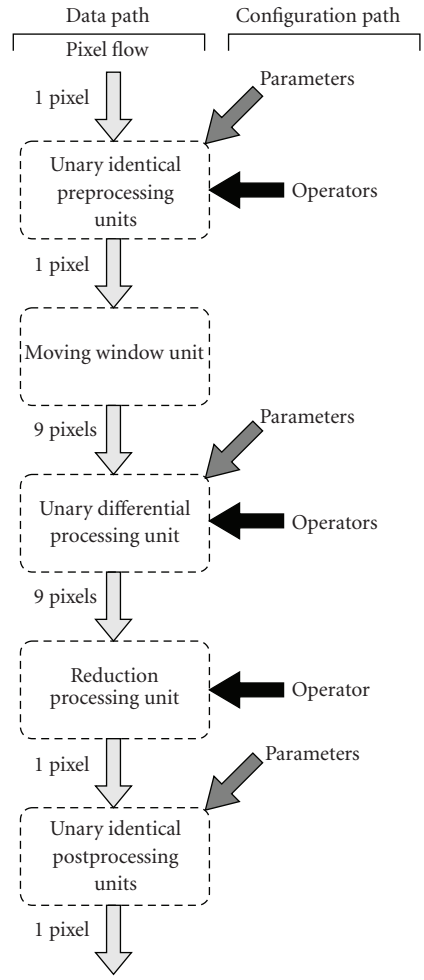

Figure 2: Decomposition of the image processing module into configurable units. On the right-hand side, configuration possibilities are expressed for each unit.

(iii) Unary differential processing unit: it inputs 9 pixels and outputs 9 pixels (see Figure 5). This unit applies a potentially different processing on each pixel of the neighborhood, for instance, multiply by a constant, test of value (black or white), and fixed assignment of value (black or white), *nop*.

(iv) Reduction processing unit: it inputs 9 pixels and outputs one pixel (see Figure 6). This unit is a tree of nodes processing simultaneously the same functionality on different values, to implement a global reduction function on the neighborhood, for instance, addition, minimum, maximum, bitwise *and*, bitwise *or*. Some nodes in the tree are *nop* nodes that are used to synchronize values that belong to the same flow generation.

(v) Unary identical postprocessing units: they input one pixel and output one pixel (see Figure 7). These units

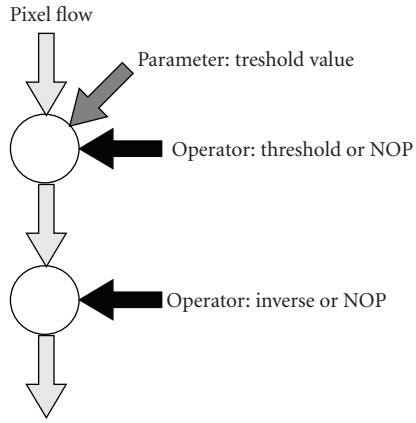

Figure 3: Unary identical preprocessing units.

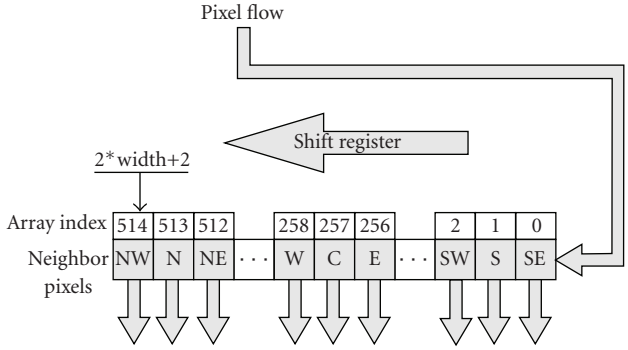

Figure 4: Moving window unit.

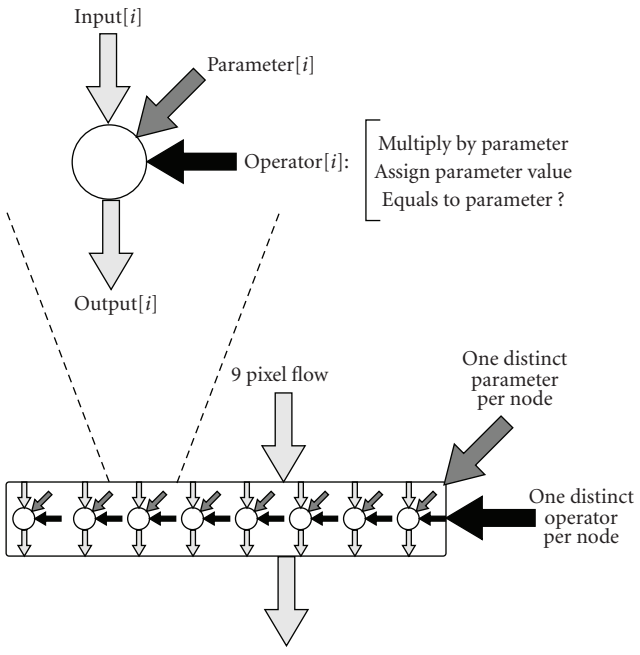

Figure 5: Unary differential processing unit.

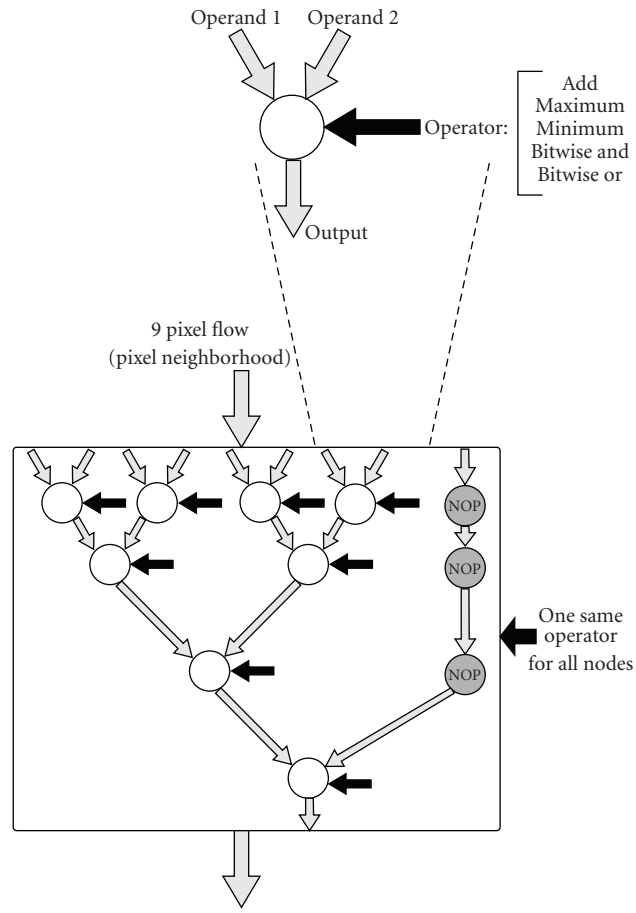

Figure 6: Reduction processing unit.

perform an identical processing on all pixels outputted from the previous unit, for instance, division, absolute value, adding a constant, and saturate. The last unit is not configurable and always computes the normalization of the resulting pixel value.

In addition to the parallel functioning of all units, units have also an internal pipelining: inputs/outputs are processed simultaneously with computation. So at a given time, a unit acts at least on three different pixel generations.

It is clear that the reconfigurability area cost is high compared with specific architecture, for instance, if the convolution matrix has uniform or symmetric values (in this case, similar computations are done several times).

6. Implementation

The architecture design in Handel-C is modeled as an assembly of macro procedures (see Algorithm 1 for the main higher level procedure). Each macro procedure correspond to a functional component of the architectural model and has input and output channels. There are two forms of macro procedures.

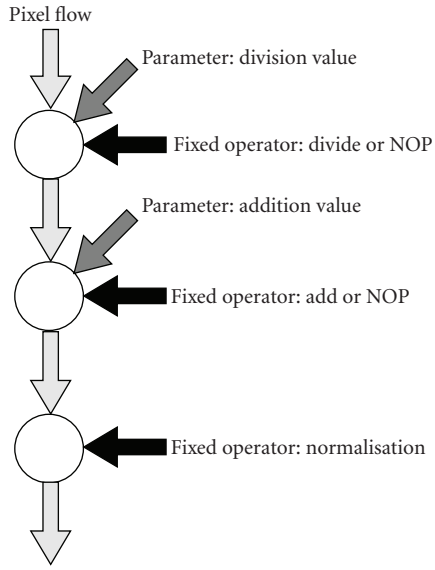

Figure 7: Unary identical postprocessing units.

```
void main (void){

  /*declaring communication channels*/
  chan unsigned 16 pixelCamFlow;
  chan unsigned 8 pixelFlow [4];
  chan unsigned char config [3];

  /*parallel activity*/
  par{
    camera (pixelCamFlow);
    conversion (pixelCamFlow, pixelFlow [0]);

    /* first module */
    module (pixelFlow [0], pixelFlow [1]
                              ,config [0], TRUE);

    /* following modules */
    par(i=1; i < moduleNumber − 1; i++){
      module (pixelFlow [i],pixelFlow[i + 1]
                              ,config [i],FALSE);

    display (pixelFlow[moduleNumber − 1]);
}}
```

Algorithm 1: Main program: all subcomponents (macro procedures) are active in parallel and are connected by synchronous channels.

(i) Terminal macro procedures execute an infinite loop where they read values from input channel, compute a result, and write this result to output channel.

(ii) Intermediate macro procedures interconnect terminal macro procedures to obtain the behavior of a component of the architectural model. The code is mainly made of channel declaration and specification of simultaneous activity of subcomponents.

Algorithm 2 shows a typical code example for a compute node (terminal), illustrated by a node from the reduction

```
macro proc computeUnit
        (functionChan, inChan, outChan){
/*variables*/
signed 13 vIn [2]; /*inputs*/
signed 14 result; /*output*/
unsigned char functionSymbol;

/*receiving the function symbol*/
functionChan ? functionSymbol;

/*initialization of the pipeline*/
/*not shown here (just comments)*/
/*receiving first input value set*/
/*then, in parallel :*/
/*receiving second input value set*/
/*computing first result*/

/*infinite loop*/
While (1){
  par{
  /*doing in parallel in one clock cycle*/

    /*receiving input values*/
    par(i=0; i < 2 i++){
      inChan[i] ? vIn[i];}

    /*computing result*/
    switch(functionSymbol){
     /*ADD*/ case '+' :
       result = vIn[0]+vIn[1];
       break;
     /*MIN*/ case '<' :
       result =
         vIn[0] < vIn[1] ? vIn[0] : vIn[1];
       break;
     /*MAX*/ case '>' :
       result =
         vIn[0] > vIn[1] ? vIn[0]:vIn[1];
       break;
       default : delay; break;}

    /*sending result*/
    outChan ! result;
}}}
```

Algorithm 2: Typical code example for a compute node (terminal macro procedure): in a clock cycle, in parallel, inputs are received, result is computed, and output is sent.

processing unit (initializations and type casting have been removed for comprehension). One can see that, in a infinite loop, three instructions are executed simultaneously during each loop cycle. While first instruction is receiving the pixels of generation $i + 1$, second instruction is computing the result for generation i, and the third instruction is sending the result of generation $i - 1$. The three instructions take one clock cycle to be achieved. Simultaneous reading and writing are not a problem in this case, as all value registers are updated at the same clock tick.

Algorithm 3 shows the code for the reduction computing unit, instantiating and interconnecting compute nodes and nop nodes. It should be noticed that we need different

```
macro proc reductionUnit
            (funcChan, inChan, outChan){
  /*declaring communication channels*/
  chan signed 13 c5,c6,c7;
  chan signed 14 c2[4];
  chan signed 15 c3[2];
  chan signed 16 c4;

  /*parallel activity of compute nodes*/
  par{
    par(j = 0; j < 4; j++){
      computeUnit1(funcChan,
        inChan[2 * j], inChan[2 * j + 1], c2[j]);}
    nopNode (inChan[8], c5);

    par(k = 0; k < 2; k++){
      computeUnit2 (funcChan,
                    c2[2 * k], c2[2 * k + 1], c3[k]);}
    nopNode (c5, c6);

    computeUnit3(funcChan,c3[0],c3[1],c4);
    nopNode (c6, c7);

    computeUnit4(funcChan,c4,c7,outChan);
}}
```

Algorithm 3: Code example for the reduction computing unit (intermediate macro procedure).

```
macro proc movingWindowUnit
            (pixelInChan, pixelsOutChan){

unsigned 8 pixels[2 * Width + 3], pixelIn;

while(1)
  par{/*in parallel*/

    /*receiving a pixel*/
    pixelInChan ? pixelIn;

    /*storing the last received pixel*/
    pixels[0] = pixelIn;

    /*shifting the memory values*/
    par(i = 1; i < 2 * Width + 3; i++){
      pixels[i] = pixels[i - 1]; }

    /*sending the neighborhood*/
    par(j = 0; j < 3 ; j++){
      par{
        pixelsOutChan[j] ! pixels[2 * Width + 2 - j];
        pixelsOutChan[3 + j] ! pixels[Width + 2 - j];
        pixelsOutChan[6 + j] ! pixels[2 - j];}}

}}
```

Algorithm 4: Code example for the moving window unit (simplified version without initialization and border management).

compute nodes corresponding to the different input and output types, even if the behavior is similar.

Algorithm 4 shows the code for the moving window unit. The pixel array is implemented with LUT configured as shift-registers, which limits the number of register and prevents the need of instantiating block ram for it.

As far as pedagogical issues are concerned, the project has also developed student's consciousness about FPGA application development regarding "compile times" (i.e., compile, synthesis, map, place, and route times). It is a fact that, in their curriculum, students have rarely seen "compile times" that exceed dozens of a second. When facing, for the largest circuits, "compile times" that exceed one hour, there is a great disappointment. This is one bad aspect of using a "high-level language" description to program an FPGA: it is easier for CS students, but if students just have in mind the program and the result without having in mind the complexity of the chain tools, comparison with software programming is quite in disfavor.

Practically this fact has the advantages to make students aware of an efficient use of the time spending on development: you cannot manage to efficiently spend your time if you wait for compilation finishing. Solutions include validating the component on a restricted set that is known to be scalable without loss of functionality (e.g., reducing image width) or working on and validating several components independently.

Concerning technical issues, another problem that students faced was managing the data types and the type casting. Even if Handel-C permits some automated type inference in particular cases, we have chosen to define all types explicitly. In fact, managing types in the reduction processing unit induces irregularity since types are different in the various compute nodes; so each node level must be described explicitly. This highlights for the students the importance of being able to precisely know the input minimal type, minimal value interval, or minimal number of value in order to improve the design performances.

Simulation, which is an important aspect of project development process, is not addressed in this course. Using a low-level behavioral simulator, like ModelSim, is out of the scope of the course: the main reason is that it would be a hard task for CS students to make the link between their high-level Handel-C program and the simulated signals as net names are generated by Handel-C compiler. Another solution would be to use the high-level Handel-C simulator of DK design suite, which permits simulation within a graphical environment (input and output images) based on the cycle accurate simulation of integer variable values. However such a software simulation is quite time-consuming and requires additional training for students. Consequently the development process for this project avoids simulation: the implementations are directly tested on the FPGA. The application context (image processing with real-time screen display) allows visual detection of most of errors or problems (e.g., black screen or pixel shift).

One interesting aspect of the implementation is the immediate visual impression about the performances of the system, as a low frame refresh rate implies a too long time for a system cycle. This aspect motivates the students in increasing the parallelism level of their design, in order to obtain a visually acceptable frame rate.

Table 1: Comparing resource use and critical path delay for various multimodule implementations.

Number of modules	LUTs	Flip Flops	Critical path delay
1	3595	2024	31.18 ns
2	7220	3749	45.57 ns
3	10717	5584	55.22 ns
4	14709	7500	49.09 ns
5	18750	9357	49.49 ns

Table 2: Comparing resource use and critical path delay for reconfigurable module and specialized module.

Design	LUTs	Flip Flops	Critical path delay
Configurable module	3595	2024	31.18 ns
Mean filter module	778	553	21.70 ns
Low pass filter module	608	520	19.90 ns
High pass filter module	612	524	20.84 ns

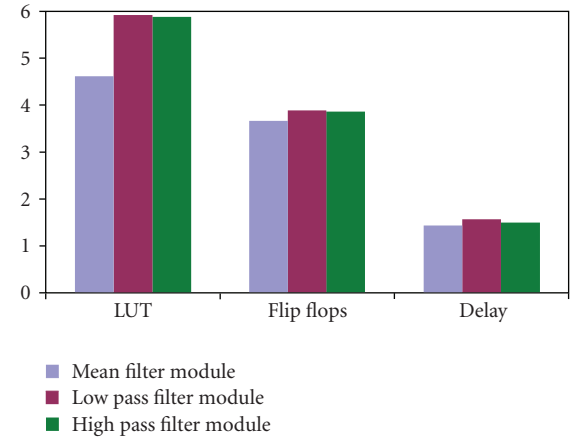

Figure 8: Size multiplication factor of a reconfigurable module compared to a specialized module for three applications.

7. Results

Students have implemented multimodule design for an image of 240 × 256 pixels. Initial pixel values are coded as an 8-bit integer (gray levels). The maximal width for the data path is of 18 bits (signed). Operands for multiplication, division, and last addition have, respectively, 5-, 6- (signed), and 8-bit width.

Table 1 shows the resource use (in LUT and flip flops) and critical path delay for a growing number of modules (maximal reachable number was 5), obtained after synthesis with DK4, map, place, and route with default parameters of ISE 8.2. Presented area is just for the composite of modules: the whole system needs approximately 1450 LUTs and 1150 flip flop more. One should notice that arithmetic units of the spartan 3L were also used (9 Mult 18 × 18 units per module). Results show that implementing several modules increases the critical path delay but it stays low enough to keep a satisfying real time computing and display.

A coarse estimation of the reconfigurability costs has been done by implementing modules with fixed behavior: global organization is the same but configuration controller is removed, and compute node is limited to one fixed function. Table 2 compares the area and delay of reconfigurable module with specialized module for mean filter, low pass filter, and high pass filter. Figure 8 presents the size multiplication factor for the different applications when using a reconfigurable module. Results show a division by 5 of the needed resources for the specialized modules. Obviously reconfigurability is costly in this case; however this should be related to the conceptual gain for user.

8. Object Modeling and Automated Generation

Even if the specification of the architecture is simplified by the language (Handel-C), the communicating process model (CSP), and the restrictive pattern for taking advantage of data flow parallelism, experience has shown that resulting programs still needed various repetitive and unproductive tasks, mainly managing type of variables, type casting, and channel connection. Those aspects prevent a program to be simply adapted to a different operational context, assuming a same application context. Solving these problems require having a generic model instance that is equipped with automated code generation methods, similar to an algorithm description in terms of hardware skeletons [19].

This is the last step of the project: to show concretely what are the gains of abstracting functionalities in terms of productivity. When the project is about to be finished, the principles of an object modeling with automated code generation are presented to students. If more time was allowed to the course, it would have been important to let them develop also this part, but this work has only been proposed as an optional project included in the curriculum.

We do not intend to describe here the exhaustive advantages of abstract modeling of an architecture class, but for this particular case we can list the main ones.

(i) Automated variable type determination and casting: from the initial type of pixel and the operator information, all types in the module are automatically processed.

(ii) Simple assembly and modifications of units and module connections: the logical links between units are specified and communicating channel are automatically allocated. For instance, *nop* nodes are automatically included in a reduction processing unit depending on the total number of inputs and number of inputs of basic nodes.

(iii) Architectural prospection: one can easily modify the functionalities and the granularity of a compute unit, generate the corresponding program, and evaluate its performances to choose the best solution, for instance, merging several successive nodes to a single node with a sequence of functions for saving cost of communication, without loss of performance if the time spent in the sequence is less than the time of the longest function in the architecture. An optimal

combination of pixel buffers and compute units can be determined depending on available resources and targeted computation time [20].

(iv) Code productivity and correctness: compute units (unary, binary, ...) share common structures which are clearly specified in a superclass. Generated code is factorized for all instances of this class.

(v) Agility: functionalities can be easily extended or restricted to focus on a specific application model.

For our problem, the object modeling of the architecture component is made up of the following classes:

(i) computing function, defined by function symbol, behavioral code (compound of inputs, parameters and operators), output type (if fixed), or dynamic type (e.g., for an addition increasing of the output width by one related to the maximal input or parameter widths);

(ii) computing node, defined by input array (pixels and parameters) and function array;

(iii) moving window node, defined by the size of the square neighborhood window and the position of the center pixel in this window, the image size, and the type of input pixel;

(iv) tree computation node defined by the basic computation node to use, and the number and the type of inputs; it defines the reduction processing unit;

(v) computation module: composite of nodes that uses instance of the previous classes to define an architecture similar to the one of the project;

(vi) classes for manipulating values, distinguishing data, parameters, and values that are just read (for a test) or used in a computation.

9. Conclusion and Perspectives

In this paper we have presented a problem-based pedagogical approach for teaching reconfigurable computing to unspecialized CS students in terms of reconfigurable architecture design, programming and technical skills, in a relatively limited time.

It is difficult to precisely evaluate the impact of such an approach on the students. The questionnaires completed by the students at the end of the project help to have various indicators. For instance, in a class of 22 students, 81% of them were more motivated by the course with this approach, 86% of them evaluated their contribution to the group to be significant, 90% of them thought that problem-based learning eased their knowledge acquisition, and 95 % of them wished to have more problem-based courses in their curriculum. Obviously CS students would not be able to design such a system without the gradual steps of the problem-based approach.

Concerning reconfigurable computing fundamentals, some aspects are missing and would merit integration. For instance, data parallelism could be tackled by dividing the image into strips computed simultaneously by parallel modules, with synchronization for border management. On another project, this data parallelism has been illustrated by using concurrent modules operating on same data (e.g., horizontal and vertical edge detection). Moreover we should go further into optimizing functions: as we assumed that a function is computed in one clock cycle, the clock frequency is limited, even if an automated retiming is performed by low-level synthesis tools. Explicit pipelining of functions (e.g., division) would increase clock frequency and data rate. If we go further into this pipelining, another interesting aspect is the tradeoff between flow parallelism and cost of channel communication; that is to say to determine what is the minimal granularity for a function that guarantees that it is not more costly to include it in a communicating process than to sequentially aggregate it with other functions in a coarser process.

We have seen in Section 7 that the end of the project opens up about high-level modeling of architecture and associated tools for optimization, automated typing, and code generation, which is close to research problems. Integrating the project into a more global environment like Madeo [21] would allow to use its automated typing inference by value enumerating, for instance, to optimize the operators in case of a restricted set of available values for parameters, which is the case for most of image processing. Other generalization of the problem would be to define it as a "configurable" CDFG [22] (Control/Data Flow Graph) to take advantage of targeting several systems more efficiently than a specific code generation.

However, including it into a course would require research tools to be accessible to students, which is a great challenge: reconfigurable computing is a fast evolving discipline; but we believe that this access to current research is a key issue to offer companies well-educated, innovation aware, and highly concerned engineers.

Acknowledgment

The author thanks Loïc Lagadec for his fruitful comments.

References

[1] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[2] C. Dezan, L. Lagadec, and B. Pottier, "Object oriented approach for modeling digital circuits," in *Proceedings of the IEEE International Conference on Microelectronic Systems Education (MSE '99)*, IEEE Computer Society, July 1999.

[3] C. Bobda, "Building up a course in reconfigurable computing," in *Proceedings of the IEEE International Conference on Microelectronic Systems Education (MSE '05)*, pp. 7–8, 2005.

[4] E. P. Ferlin and V. Pilla Jr., "The learning of reconfigurable computing in the computer engineering program," in *Proceedings 36th ASEE/IEEE Frontiers in Education Conference (FIE '06)*, San Diego, Calif, USA, October 2006.

[5] E. Fabiani, C. Gouyen, and B. Pottier, "Intermediate level components for reconfigurable platforms," in *Proceedings of the International Workshops on Computer Systems: Architectures,*

Modeling, and Simulation (SAMOS '04), vol. 3133, pp. 59–68, Springer, Samos, Greece, July 2004.

[6] A. Kolmos, "Problem-based and project-based learning," in *University Science and Mathematics Education in Transition*, pp. 261–280, 2009.

[7] A. Stojcevski and D. Fitrio, "Project based learning curriculum in microelectronics Engineering," in *Proceedings of the 4th IEEE International Conference on Parallel and Distributed Systems (ICPADS '08)*, pp. 773–777, IEEE Computer Society, Australia, December 2008.

[8] Celoxica, "Handel-C Language Reference Manual," 2005.

[9] P. H. Welch and D. C. Wood, "The Kent Retargetable occam Compiler," in *Proceedings of the 19th World Occam and Transputer User Group Technical Meeting on Parallel Processing Developments (WoTUG '96)*, B. O'Neill, Ed., pp. 143–166, March 1996.

[10] Celoxica, "Platform Developer Kit: RC10 Manual".

[11] R. B. Porter, "Image processing," in *Reconfigurable Computing*, Springer, New York, NY, USA, 2005.

[12] P. Guermeur, P. Dokladal, E. Dokladalova, and A. Manzanera, "FPGA lab sessions in a general purpose image processing course," in *Proceedings of the 2nd International Workshop on Reconfigurable Computing Education*, May 2007.

[13] Celoxica, "DK Design Suite user guide," 2005.

[14] Xilinx, "ISE 8 Software Manuals," 2006.

[15] V. Muthukumar and D. V. Rao, "Image processing algorithms on reconfigurable architecture using HandelC," *Journal of Engineering and Applied Science*, vol. 1, no. 2, pp. 103–111, 2006.

[16] D. V. Rao, S. Patil, N. A. Babu, and V. Muthukumar, "Implementation and evaluation of image processing algorithms on reconfigurable architecture using C-based hardware descriptive languages," *International Journal of Theoretical and Applied Computer Sciences*, vol. 1, no. 1, pp. 9–34, 2006.

[17] C. Torres-Huitzil and M. Arias-Estrada, "FPGA-based configurable systolic architecture for window-based image processing," *EURASIP Journal on Applied Signal Processing*, vol. 2005, no. 7, pp. 1024–1034, 2005.

[18] D. Etiemble and L. Lacassagne, "Introducing image processing and SIMD computations with FPGA soft-cores and customized instructions," in *Proceedings of the International Workshop on Reconfigurable Computing Education*, 2006.

[19] K. Benkrid, D. Crookes, and A. Benkrid, "Towards a general framework for FPGA based image processing using hardware skeletons," *Parallel Computing*, vol. 28, no. 7-8, pp. 1141–1154, 2002.

[20] X. Liang and J. S.-N. Jean, "Mapping of generalized template matching onto reconfigurable computers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 3, pp. 485–498, 2003.

[21] L. Lagadec, B. Pottier, and O. Villellas, "A LUT based high level synthesis framework for reconfigurable architectures," in *Domain-Specific Processors: Systems, Architectures, Modeling and Simulations*, pp. 19–39, Marcel Dekker, 2003.

[22] M. Rashid, T. Goubier, and B. Pottier, "A high level generic application analysis methodology for early design space exploration," in *Proceedings of the Workshop on Design and Architectures for Signal and Image Processing (DASIP '07)*, November 2007.