J Real-Time Image Proc (2014) 9:47–59
DOI 10.1007/s11554-012-0321-2

SPECIAL ISSUE

Flexible VLIW processor based on FPGA for efficient embedded real-time image processing

Vincent Brost · Fan Yang · Charles Meunier

Received: 27 January 2012 / Accepted: 31 December 2012 / Published online: 17 January 2013

Abstract Modern field programmable gate array (FPGA) chips, with their larger memory capacity and reconfigurability potential, are opening new frontiers in rapid prototyping of embedded systems. With the advent of high-density FPGAs, it is now possible to implement a high-performance VLIW (very long instruction word) processor core in an FPGA. With VLIW architecture, the processor effectiveness depends on the ability of compilers to provide sufficient ILP (instruction-level parallelism) from program code. This paper describes research result about enabling the VLIW processor model for real-time processing applications by exploiting FPGA technology. Our goals are to keep the flexibility of processors to shorten the development cycle, and to use the powerful FPGA resources to increase real-time performance. We present a flexible VLIW VHDL processor model with a variable instruction set and a customizable architecture which allows exploiting intrinsic parallelism of a target application using advanced compiler technology and implementing it in an optimal manner on FPGA. Some common algorithms of image processing were tested and validated using the proposed development cycle. We also realized the rapid prototyping of embedded contactless palmprint extraction on an FPGA Virtex-6 based board for a biometric application and obtained a processing time of 145.6 ms per image. Our approach applies some criteria for co-design tools: flexibility, modularity, performance, and reusability.

V. Brost · F. Yang (✉) · C. Meunier
LE2I-CNRS 6306 Laboratory,
University of Burgundy, 21078 Dijon, France
e-mail: fanyang@u-bourgogne.fr

V. Brost
e-mail: vincent.brost@free.fr

C. Meunier
e-mail: charles.meunier@iut-dijon.u-bourgogne.fr

Keywords Rapid prototyping · System design · VLIW processor · FPGA · Real-time image processing · Biometric system

1 Introduction

Electronic embedded systems play an important role in diverse real-time signal and image-processing applications such as process control, telecommunication, satellites, and the medical field [1, 2]. The user-programmable FPGA is capable of performing the hardware part of a design for a significantly lower price and they maintain many of the advantages of the ASIC (application specific integrated circuit) solutions. FPGA foundries offer many built-in circuit features, such as memory, multipliers, and high-speed communication links. The most interesting characteristic of FPGA, with its reconfigurable nature, is probably the ability to quickly create a rapid and fully functional prototype that can emulate and verify solutions, or even be embedded into the final system [3, 4].

In the standard FPGA-based prototyping methodology, algorithms are first developed on a personal computer or workstation in standard software programming languages such as C or Matlab. When the algorithm is later implemented in hardware, the C (or Matlab) code is translated into a hardware description language such as VHDL or Verilog. Finally, the design is synthesized for an FPGA-based environment where it can be tested. Most hardware description languages are inherently concurrent and not considered trivial for non-hardware developers. One of the key factors that encourage the wide diffusion of electronic devices is the improvement of the man–machine interface, where the great challenge is to allow the use of complex electronic systems by software developers. Many research laboratories and industrial manufacturers are focusing their efforts to that effect [5, 6]. Two major trends emerge.

The first, based on the fact that software developers know much more about traditional software high-level programming languages such as C and C++ than about hardware description languages, a major trend consists in extending these languages with variations capable of describing hardware elements. These new hardware description languages [7–9] are, in effect, parallel synchronous programming languages where the notion of time is fundamental to its specification. In these languages, all events occur relative to a global clock that runs continuously. Information is encoded on a behavioural level in a similar manner to most high-level languages.

The second trend consists in using programmable devices, a processor core included in FPGA for example. Some parts of application can be programmed with high-level languages like C or C++. The rest of FPGA is used to manage communication peripherals or to integrate low-level processing blocks. All these programmable devices including processor cores can be considered single chip implementations of SW/HW hybrid system [5].

In this article, we propose an approach that combines these two trends. We target rapid prototyping of signal and image-processing applications on the FPGA. Firstly, algorithms are programmed in C as if they were to be executed on a classical processor. Then the advanced compiler OpenIMPACT [10] converts the original program into an independent assembler called *Lcode*. This intermediate representation provides instruction-level parallelism (ILP) which is analyzed and reorganized to VLIW instructions. Finally, a flexible VLIW VHDL processor model with a variable instruction set is generated and implemented optimally with FPGA technology.

We have applied the same methodology in previous works to implement multi-digital signal processor (DSP) TMS320C62xx of Texas Instrument into FPGA devices using the Code Composer Studio [6, 11]. The method presented in this paper can be considered as the generalization and the improvement of first experiences. It is more flexible; the generic VLIW architecture is totally customizable (number of functional units, number of registers and instruction set) in function of a target application. In addition, we use an advanced compiler of research OpenIMPACT to realize optimally hardware implementations on embedded systems.

This article is organized as follows in the following section, we discuss some existing works and state the main contribution of this paper. In Sect. 3 we introduce the application development cycle and illustrate its principles and concepts from algorithm programming to implementation on an FPGA. Section 4 describes obtained experiment results for some classical image-processing algorithms using the proposed approach. We present also, in Sect. 5, hardware implementation of a biometric application (contactless palmprint extraction) on FPGA Xilinx Viretex-6. Finally, we conclude in Sect. 6.

2 Related works

Enabling VHDL processors model onto FPGA is not new. Since 10 years, FPGA foundries provide already some common processors such as MicroBlaze of Xilinx [12], Nios II of Altera [13] and Mico32 of Latice [14]. They are usually called “soft-core” processors to make the difference with “hard-core” processors that are physically present in FPGA devices. These soft-core processors can be programmed in Assembly or others classical languages to facilitate hardware implementations on FPGA. They can be considered as general-purpose processor (GPP) with fixed architecture.

To benefit large available hardware resources of new FPGA devices and to improve implementation performance, the concept of reconfigurable processor was introduced. This allows realizing optimal and adaptive models for target applications. We can cite, for example, the Grid Alu Processor (GAP) produced by the University of Augsburg [15, 16] and KAHRISMA architecture of Karlsruhe Institute of Technology [17].

The GAP architecture mixes the advantages of a superscalar processor and those of a coarse-grained dynamically reconfigurable system. Their reconfigurable elements consist of an array of functional units (FUs), a branch unit, and several load/store units. To improve the performance and, in parallel, decrease the hardware requirements of the GAP, some DSE (design space exploration) [16] works have performed with multi-objective optimization to evaluate and define GAP architectural parameters. The great advantage of the GAP architecture is the improvement of sequential program execution that cannot be provided by modern multi-threaded and multi-core architectures.

KAHRISMA designs KArlsruhe’s Hypermorphic Reconfigurable Instruction Set Multi-grained Array architecture [17]. Based on the fact that the selection of processing architectures (e.g., GPPs, DSPs, ASICs, etc.) is determined at design time depending upon the requirements of a certain set of applications, a SoC (system-on-chip) typically does not provide the demanded efficiency when executing applications from different domains. One solution to address this problem is reconfigurable computing that utilizes resources in a time-multiplexed manner. Flexibilities at both design time and run time of the KAHRISMA architecture allow to efficiently supporting all instruction-level parallelism (ILP), data-level parallelism (DLP), and thread-level parallelism (TLP).

Lam and Srikanthan [18] present a framework that enables rapid design exploration for reconfigurable instruction set processors (RISP). In particular, it provides rapid identification of a reduced set of profitable custom instructions and their area costs on commercial architectures without the need for time-consuming hardware

synthesis process. The proposed framework can also be used for determining the optimal size of FPGAs to be embedded in cost and power sensitive SoC platforms. Their experimental results have been obtained for the generic, automotive industrial, image, network, security and telecommunications application sets.

In comparison with these works, our approach address to rapid prototyping of signal and image applications on FPGA devices. For example, processing full high-dimension images leads to millions of independent operations but identical on different parts of the image. In this domain, instruction-level parallelism extraction is particularly important and efficient before data-level parallelism exploration. Our development cycle, for the first time, provides multiple advantages:

- It possesses more programming flexibility: the input consists of classical and standard C or C++ code source (not oriented hardware languages such as Handel C or System C). Because of automatic generating of VLIW VHDL processor model, software engineers can realize rapid prototyping of signal and image processing on FPGA devices ignoring completely hardware aspects (description language, FPGA architecture). This is one of the key factors that encourage the wide diffusion of electronic devices.
- Our approach realizes performance-efficient hardware implementations: the concept of flexible VLIW processor allows us to exploit intrinsic instruction-level parallelism in maximum to reduce necessary execution cycle number. In the same time, the VHDL processor model uses just only minimum-necessary hardware resources for target application. This makes better performances in term of Area/Speed ratio.
- Our application development cycle consists of a new open and adaptive framework: in this paper, we present only basic steps to exploit low-level parallelism (ILP). Some high-level parallelism (for example, data distribution and task pipeline, etc.) can also be realized using the proposed method. This last point will be illustrated in the end of Sect. 5.

3 Application development cycle

In this section, we describe the application development cycle from algorithm programming to implementation on a FPGA platform. Figure 1 illustrates our approach composed of four stages: intermediate representation code generation, ILP analysis and hardware characteristic extraction, VHDL description of flexible VLIW processor and its implementation on FPGA.

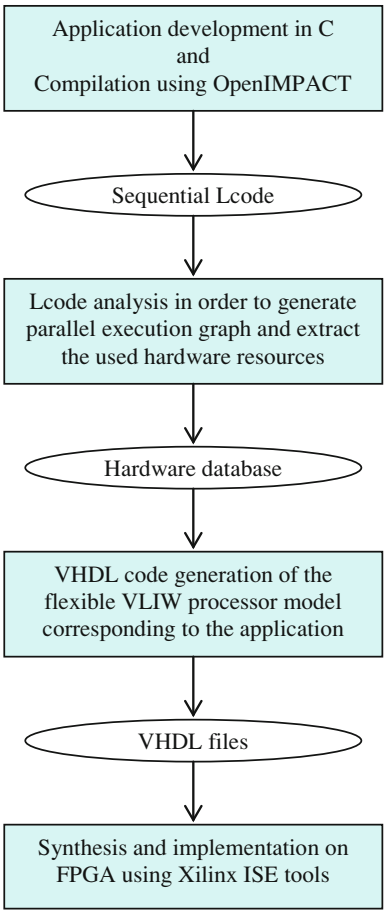

Fig. 1 Proposed application development cycle

3.1 Lcode generation using OpenIMPACT compiler

The OpenIMPACT compiler is maintained by the IMPACT group at the University of Illinois, under the direction of professor Wen-mei W. Hwu. The objective of IMPACT (Illinois Micro-architecture Project utilizing Advanced Compiler Technology) is to provide critical research, architecture expertise, and compiler prototypes for the microprocessor industry [19]. This objective is accomplished by analyzing and demonstrating the level of hardware and compiler support required by architectural enhancements to understand the cost and effectiveness of these enhancements. IMPACT's focus has historically been instruction-level parallelism (ILP).

We use OpenIMPACT to compile the original source-code into an assembly intermediate representation *Lcode*. This produced *Lcode* is optimized for ILP, but not for a specific machine. For example, OpenIMPACT generates and evaluates the code for an instruction set architecture (ISA): HPL-PD [20]. This primary/native ISA is a parametric VLIW architecture. It admits processors of different composition and scale, especially with respect to the

amount of parallelism offered. The HPL-PD parameter space includes the number of clusters in a multi-cluster processor, the make up of each cluster (types of functional units, the composition of the register files), and the instruction set including operation latencies and descriptors that specify when operands may be read and written, instruction format and resource usage behavior of each operation. The architecture instruction set is akin to an RISC load-store architecture, with standard arithmetic and memory operations.

The OpenIMPACT compiler supports many aggressive compiler optimizations including procedure inlining, loop unrolling, speculation, and predication. IMPACT organizes its optimizations into three levels [21]:

Classical—performs only traditional local optimizations,

Superscalar/Superblock—adds procedure inlining, loop unrolling, and speculative execution,

Hyperblock—adds predication (conditional execution).

3.2 ILP analysis and hardware characteristic extraction

The OpenIMPACT environment includes a trace-driven simulator and an ILP compiler. The simulator enables both statistical and cycle-accurate trace-driven simulation of a variety of parameterizable architecture models, including both VLIW and in-order superscalar data-paths. The generated *Lcode* representation is essentially a large, generic instruction set of simple operations like those found on typical RISC architectures, but not biased toward any particular architecture.

Instructions of *Lcode* begin with the word **op** and are composed of four major parts: operation number, opcode, operands, and attributes. The operation number is unique for each *Lcode* instruction in the function. Opcodes include the **define** operations seen in Fig. 2.

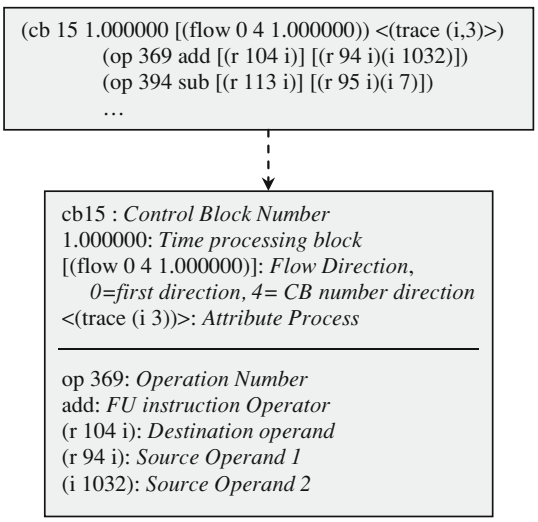

Fig. 2 Lcode block sample (cf. [21])

We have developed a software tool which performs *Lcode* analysis to expose ILP and extract need used resources for a target application. This allows us to construct hardware resource database.

The software tool executes analysis in several phases. Firstly, it performs register optimization: *Lcode* produced by the OpenIMPACT compiler uses unlimited registers while FPGA device needs limited register definition to reduce the use of hardware resources. The original *Lcode* specifies a maximum number of registers (a new output register for each basic operation); at the time of an instruction re-scheduling step, we detect and delete all redundant registers taking into account of register re-uses possibility.

Secondly, our tool examines sequentially all instructions of *Lcode* and notes those can be executed in parallel with previous instructions (without data dependences). Some optimizations are also performed. For example, according to hardware resource constraint specification (number of units, memory access number per cycle), we can change operation execution order to reduce execution time. Figure 3 gives detailed description of this ILP-extraction step. Then, we analyze the basic operation types and used registers of each parallel cycle respecting the consistency of the original *Lcode*. Finally, a parallel operation graph is generated to construct hardware resource database.

We use the Sobel filter as an example to illustrate this parallel operation graph generation step. The Sobel filter is considered as a simple procedure for performing edge extraction. It is usually implemented with a convolution using two 3×3 masks. Figure 4 displays the analysis and generation obtained results for this Sobel processing. From *Lcode* generated by OpenIMPACT compiler, we analyze data and operation dependences to transform the original sequential assembly code in parallel basic operations. To facilitate hardware implementation, we respect the constraint of intern Block RAM in FPGA: two accesses per cycle in dual port model. We can see that the main loop has be performed using ten cycles after parallelization. Here, *Ri* (i = 0–7) corresponds to a register and *ld* to the data loading from memory. The *lsl* represents data left shift. *C* indicates a constant. For example, at the step (cycle) 6, two operations have been simultaneously realized:

```
Add R0, R6 → R0
//
Sub R2, R5 → R2
```

This parallelization allows us to accelerate calculations with a ratio of *2.7* which corresponds to number of sequential cycles per number of parallel cycles.

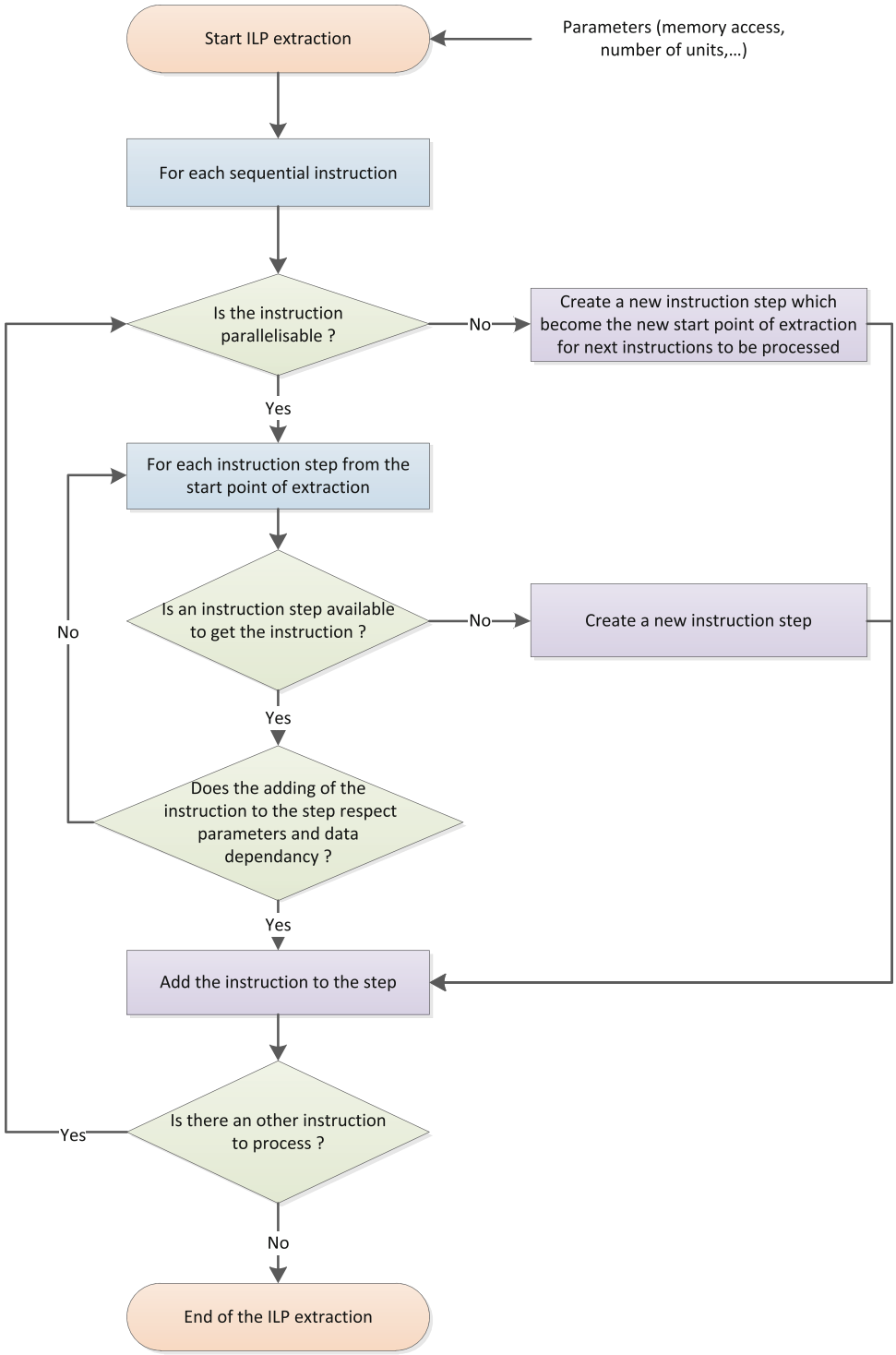

Fig. 3 Instruction-level parallelism extraction description: all sequential instructions are examined and processed

3.3 VHDL description of flexible VLIW processor and FPGA implementation

Using *Lcode* analysis and graph model generation results, a flexible VLIW processor can be composed: for a target application, we know the minimum-need hardware resources (register number, basic operation types, instruction set and length of each VLIW instruction). With hardware database information, we construct the structure of VLIW processor and describe it in VHDL language (see Fig. 5 and Sect. 4 for detail description).

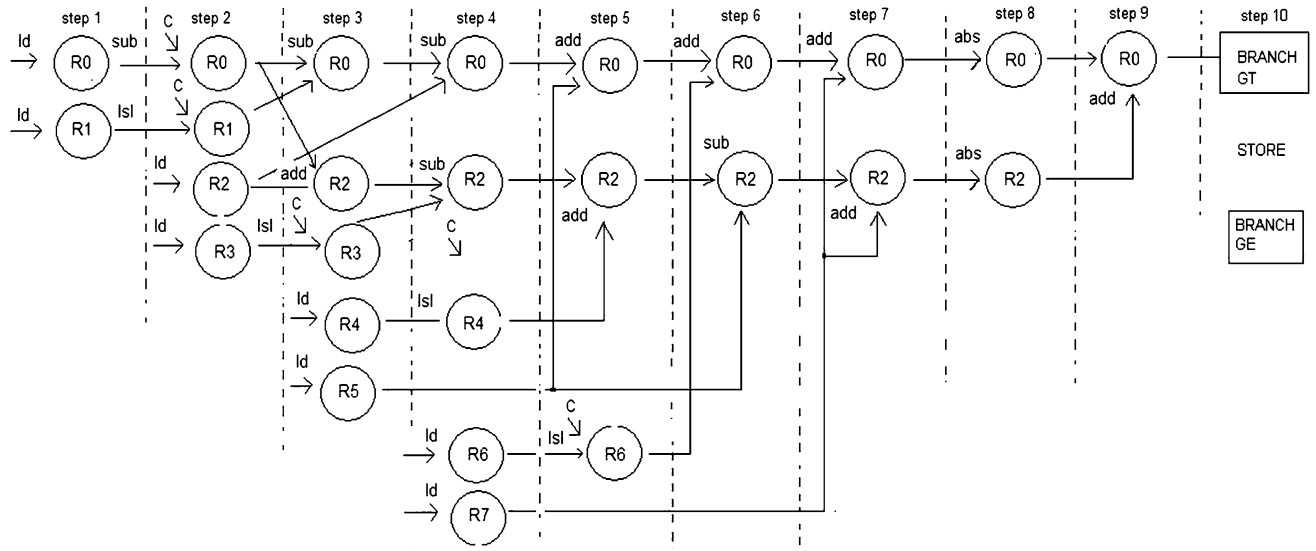

Fig. 4 Parallel operation graph generated by our software tool for the Sobel filter: each operation is realized using two operands from two registers and memorizes the result in a register (cf. [22])

It is noted that the VLIW processor is totally customizable and optimal for target application: number of functional units, operation types and used registers are just minimum necessary for given algorithm. We use a created library of hardware templates to present VLIW processor in VHDL language.

We respect different pipelined levels of VLIW processors to control execution-time scheduling: program fetching, instruction dispatching, and instruction decoding units deliver up instructions to functional units every CPU clock cycle [23]. This mechanism has been described in VHDL code and implemented onto FPGA. All generated VHDL files for a target application have been synthesized using Xilinx [24] ISE tools [25] to realize hardware implementation.

4 Experiment results

Image processing (computer vision) is compute-intensive and classified into three classes, namely, high, middle and low-level algorithms based on their computation and communication characteristics. The low-level algorithms, in particular, deal with raw and uncompressed images. Processing such images leads to millions of independent operations but identical on different parts of the image resulting in a high degree of data parallelism. Due to these reasons, low-level algorithms are right candidates for hardware implementation [26, 27]. A lot of co-design SW/HW tools are proposed, which provide a fast implementation of the compute-intensive image processing algorithms.

To test and validate our development cycle of rapid prototyping, we have chosen three widely used low-level computer vision algorithms: filter Sobel for performing edge extraction, convolution with a mask of 3×3, and FDCT (fast discrete cosine transform). These basic operations are usually considered as image pre-processing stage.

Our experiment results are obtained using an Intel Pentium 4 computer, with a clock speed of 2.8 GHz. The OpenIMPACT environment was pre-installed with Linux 2.6.22.5 kernel as underlying operating system. We use the Xilinx ISE [24, 25] simulator for FPGA based implementation and target hardware architecture is the Virtex 6-xc6vlx75T [28].

Figure 6 displays the flexible VLIW processor architecture for the Sobel filtering application. In agreement with the *Lcode* analysis and operation graph model generation results (see Fig. 4), this flexible VLIW processor is composed of 7 functional units (2 Add, 2 Sub, 2 Abs and 1 Shift left). For example, at the sixth cycle, 2 arithmetic operators are triggered. The add(2) recuperates first operand (R2) from mux_add2_e0 and second (R6) from mux_add2_e1 and performs addition operation. The result value is stored in the register R0. In the same manner, the sub(2) performs subtraction with two operands (R2 and R5) and transfers this subtraction result to the R2 register.

Based on the flexible VLIW processor architecture, its VHDL description is automatically generated. Note that after analysis steps, our tool also generates automatically a VHDL code (corresponding to ROM) to manage control signals of multiplexers for each execution cycle (see also Sect. 3.3). Synthesis and simulations of all VHDL code for the target algorithm are realized using the Xilinx ISE tools. Table 1 shows *Lcode* analysis and parallelism extraction results for these basic image-processing algorithms.

```
component reg is   Port (
   e        : in std_logic_vector(15 downto 0);
   clk          : in std_logic;
   reset    : in std_logic;
   s        : out std_logic_vector(15 downto 0)
                                 );
end component;
```

```
for i in 0 to 8 generate
begin
  re:reg port map(
      e => regist_e(i),
      s => regist_s(i),
      clk => clk_bufg,
      reset => reset);
end generate;
```

```
type type_reg is array(integer range 9 downto 0) of std_logic_vector(15 downto 0);
```

```
mux_r0:mux_gen_4 port map(
  e0 => data_mem_0_lo,
  e1 => X"0000",
  e2 => X"0000",
  e3 => regist_s(0),
  slct => data_vliw(1 downto 0),
  s => regist_e(0));
```

```
function lsl(ARG: STD_LOGIC_VECTOR; COUNT: NATURAL) return STD_LOGIC_VECTOR
     is
   constant ARG_L: INTEGER := ARG'LENGTH-1;
   alias XARG: STD_LOGIC_VECTOR(ARG_L downto 0) is ARG;
   variable RESULT: STD_LOGIC_VECTOR(ARG_L downto 0) := (others => '0');
 begin
   if COUNT <= ARG_L then
     RESULT(ARG_L downto COUNT) := XARG(ARG_L-COUNT downto 0);
   end if;
   return RESULT;
end lsl;
```

```
mux_lsl_1_e0:mux_gen_4 port map(
  e0 => regist_s(1),
  e1 => regist_s(3),
  e2 => regist_s(4),
  e3 => regist_s(6),
  slct => data_vliw(30 downto 29),
  s => lsl_1_e0);
```

Fig. 5 Segment of VHDL model generation for the Sobel application

Obtained experiments' results concerning VHDL code synthesis and hardware simulation are given, respectively, in Tables 2, 3 and 4.

We can note that an average acceleration of 3× has been obtained for these three basic image processing using our graph model of parallel execution. In general, these algorithms use a small percentage of hardware resources available on FPGA: 0.24 % of register Slices, 1.1 % of LUT slices, and 73 % of Block RAMs. Speed corresponding to number of processed images per second is, respectively, of 160, 104 and 52 images/s. These results have been obtained with an image size of 512 × 512 pixels. Intern Block RAMs of FPGA are used to store original and resulting images, while program instructions are loaded in LUT blocks of FPGA Slices.

To make some comparisons with available parallel implementation techniques, we have completed experiment results by two other hardware realizations: Full VHDL and 4-MicroBlaze in parallel. This complementary study is performed only with the third algorithm: fast discrete cosine transform (FDCT).

Full VHDL implementation use directly the hardware description language while the second realization executes the software language C source on 4 soft-core processors MicroBlaze in parallel. Firstly, the C source of FDCT algorithm was compiled using Xilinx GCC cross compilation tools. Then these 4 soft-cores with fixed architecture process simultaneously in SPMD (single program multiple data) mode, each core possess its specific block memory in order to avoid access conflict. Each soft-core performs a

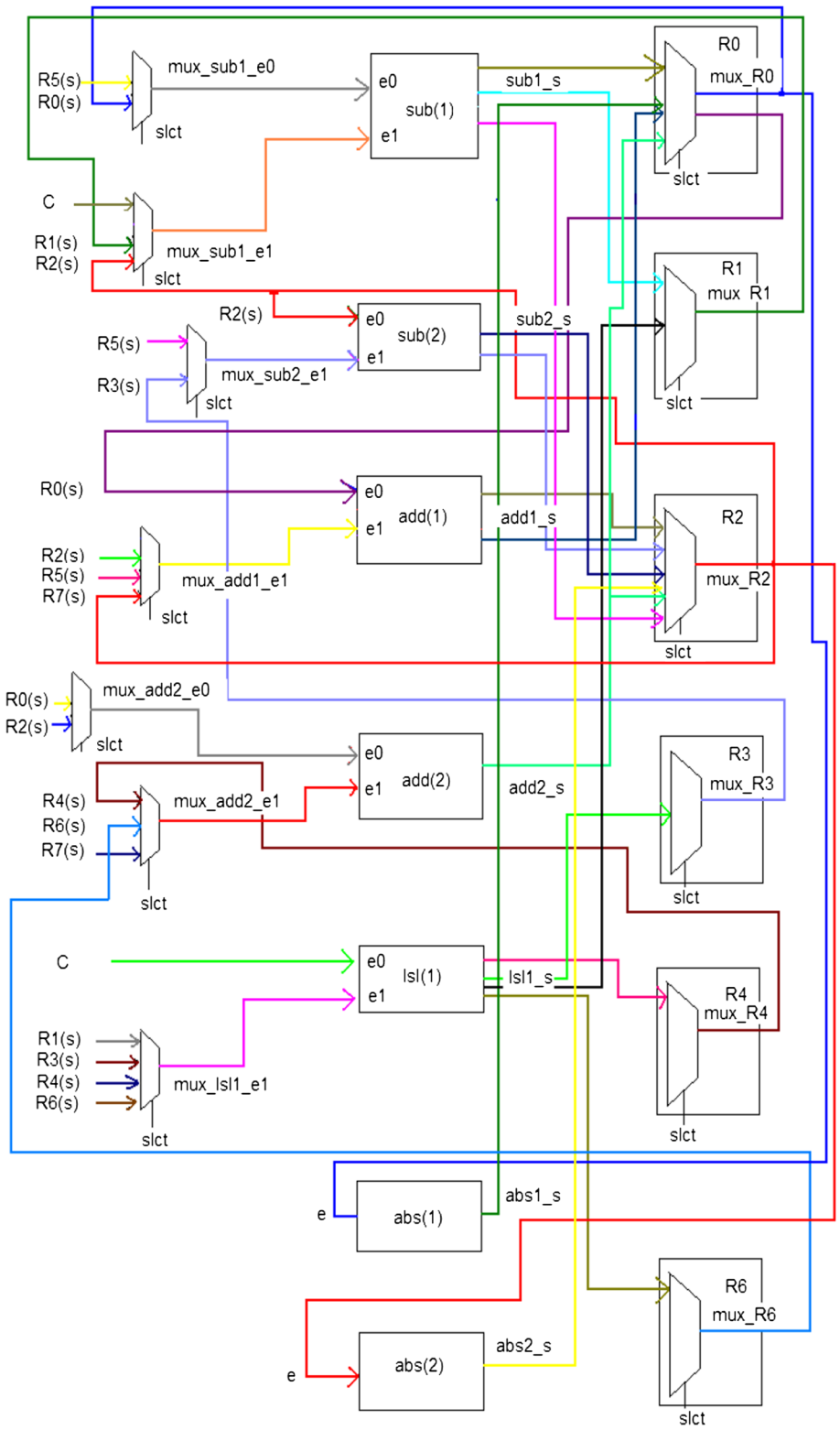

Fig. 6 Flexible VLIW processor structure for the Sobel filtering algorithm

quarter of image of 512 × 512 pixels. Obtained hardware performances are given in Table 5.

We can note that the Full VHDL implementation is better in terms of hardware resource cost and image-processing speed. The 4-MicroBlaze realization uses a lot of more hardware resources than our flexible VLIW processor and processes only the half of images per second (27 vs. 52), because this implementation exploits only the data-level parallelism.

Table 1 Lcode analysis and parallelism extraction results: numbers of cycle correspond to the pixel-level processing

Algorithm	Sobel	Conv.	FDCT
Number of sequential cycle	27	14	108
Number of parallel cycle	10	9	17
Acceleration	2.7	1.6	6.3

Table 2 Hardware implementation results for the Sobel filtering: Fr = 420 MHz

Logic utilization	Used	Available	Ratio (%)
No. of register slice	132	93,120	0.14
No. of LUT slice	409	46,560	0.88
No. of Block RAMs	114	156	73

Table 3 Hardware implementation results for the 3 × 3 convolution: Fr = 247 MHz

Logic utilization	Used	Available	Ratio (%)
No. of register slice	76	93,120	0.08
No. of LUT slice	113	46,560	0.24
No. of Block RAMs	114	156	73

Table 4 Hardware implementation results for the FDCT: Fr = 230 MHz

Logic utilization	Used	Available	Ratio (%)
No. of register slice	478	93,120	0.51
No. of LUT slice	1,035	46,560	2.2
No. of Block RAMs	114	156	73

The Full VHDL implementation is the most expensive in terms of development cycle speed: nearly 1 week for a hardware expert. One day is necessary for the 4-MicroBlaze realization, while the flexible VLIW processor is embedded instantaneously. It is to note that two first implementations need specific hardware knowledge and experience.

5 Study case: contactless palmprint extraction

In recent years, new powerful FPGA devices allow hardware realizations of complex applications in pattern recognition topic [29–31]. For example, Ahmad et al. [29] combine discrete wavelet transform and FPGA-based intellectual property (IP) core for a new approach of face recognition. We can also cite works realized by Fons et al. [31] on FPGA-based personal authentification using fingerprints. In this section, we perform rapid prototyping of a biometric application using the proposed development cycle. This consists of contactless palmprint extraction. We present different processing stages followed of hardware implementation description.

5.1 Contactless palmprint extraction presentation

Palmprint can be used as a reliable human, identifier because the pattern of ridges is unique and their details are permanent. Compared to other physical biometric characteristics, palmprint biometric has several advantages: low intrusiveness, stable line features, and low-cost capturing device. Although palmprint is traditionally a contacting biometric, our research team has developed a palmprint extraction and recognition system without contact, which allows us to keep a pleasant and hygienic system [32]. Working on palmprint in a contactless context requires some pre-processing. The region of interest (ROI) must indeed be extracted from the hand image. Figure 7 displays different stages of this extraction from contactless hand images. Palm extraction requires hand localization, followed by palm localization in the hand, and then normalization because of the rotation and scale variation induced by the free placement. Hand segmentation consists of a thresholding on the red component of the RGB space: as a green background has been chosen, the redder pixels belong to the hand. Some morphological operations are also used to enhance the hand edges. After this step, multiple reference points are defined: they correspond to the fingertips and valleys between fingers. This localization of the hand extremities is achieved in two steps.

First, a contour extraction is performed using an eight neighbourhood border-tracking algorithm known as *the*

Table 5 Hardware performance comparisons for the FDCT implementation: Full VHDL, 4-MicroBlaze and Flexible VLIW

Implementation on FPGA Virtex-6	Full VHDL	4 Micro Blaze	Flexible VLIW
No. of register slice	820	14,588	478
No. of LUT Slice	544	12,882	1,035
No. of Block RAMs	114	154	114
Frequency (MHz)	380	190	230
Proc. speed (images/s)	181	27	52

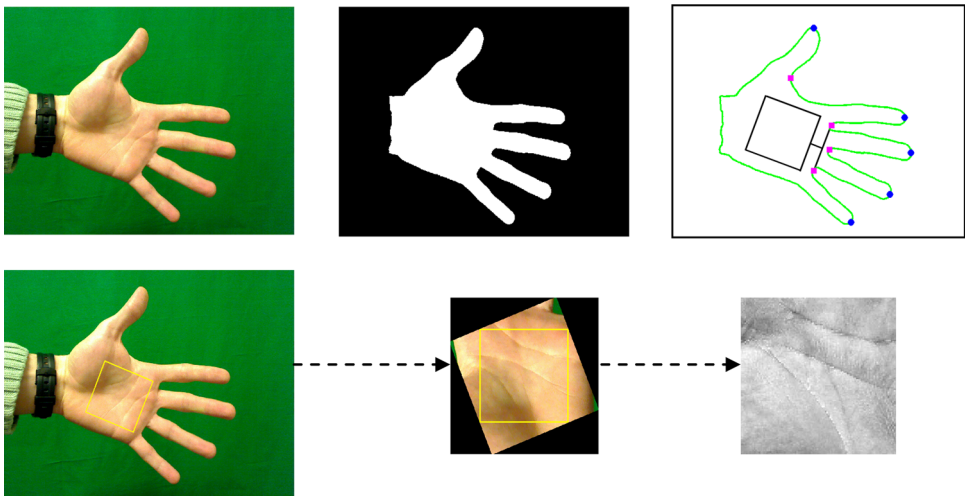

Fig. 7 Illustration of different stages for the contactless palmprint extraction: hand segmentation, Freeman edge detection, point processing and palm extraction

Freeman algorithm. Secondly, hand extremities locations are found. As subject fingers are located on the right of the image, local minima and maxima of the hand contour abscissa can be considered as fingertips and valleys. Because these initialized locations are not accurate, we applied a refining algorithm, which minimizes the Euclidean distance between the considered point and its two neighbours among the reference points.

Location of the new fiducial points is deduced from the length of the index and little finger. Figure 7 shows the square window, which corresponds to the ROI. The window size depends on the distance between the hand and the camera. Therefore, it is taken proportional to the hand width. As the palmprint images are of different sizes and orientations, we normalize them. First, they are rotated around the vertical axis. Then, they are resized to a standard image size of 64 × 64 pixels, and converted into a gray-level image. After this ROI pre-processing, biometric recognition is realized using Gabor filtering (feature extraction) and Hamming distance (similarity measurement) with multiple shifts to introduce a tolerance in translation [32].

5.2 Hardware implementation of embedded palmprint extraction

We use the ML605 evaluation board [28] as hardware platform for embedded palmprint extraction. It enables hardware and software developers to create or evaluate designs targeting the Virtex-6 XC6VLX240T-1FFG1156 FPGA. It contains 768 DSP slices (with 25 × 18 multipliers and 48-bit adder/subtracter/accumulator), which support massively parallel DSP algorithms, 416 intern Block RAMs and 37,680 configurable logic blocks (CLBs). Slices of the CLBs can be used to provide logic, arithmetic, and ROM functions; a part of them can also be used as distributing RAM or 32-bit data registers. The ML605 provides also board features common to many embedded processing systems, which include a DDR3 SODIMM memory, an 8-lane PCI express interface, a tri-mode Ethernet PHY, general purpose I/O, and a UART.

Based on the Virtex-6, we added a CMOS sensor to capture hand images, the DDR3 memory is necessary to store original and resulting images of different processing stages. Using the proposed development cycle of rapid prototyping, we have enabled five VLIW VHDL processors corresponding to five main functions of palmprint extraction. Obtained results of these hardware implementations are given in Table 6.

We present ILP extraction of each function as the ratio of necessary sequential/parallel cycle number (acceleration). In the frequency column, values correspond to obtained synthesis results of Xlinx ISE. Two first functions (*Thresholding* and *Erosion*) constitute the hand segmentation stage. They perform on images of 1,200 × 1,600 pixels and their corresponding VLIW processors execute with a lot of cycle numbers. Others functions process on less important data size: ∼5,000 edge pixels for *Freeman*, 2,480 sampled edge pixels for *Point processing*, and 64 × 64 pixels for *Palm extraction*. Their corresponding processing times are very small.

Figure 8 illustrates the realized architecture for hardware implementation of complete palmprint extraction processing chain. We use a PicoBlaze microcontroller core implemented on the FPGA to synchronize different VHDL modules. PicoBlaze is a Xilinx's intellectual property [24]; this soft-core microcontroller is programmed in assembly. It triggers firstly the Image acquisition block when a hand image arrives in the FPGA, and the Memory manager block to store this original image in the DDR3 memory. Then, it starts sequentially five VLIW VHDL processor blocks through five different signals (pb$i$, i = 1–5), and

Table 6 Hardware implementation results of 5 VLIW VHDL processors corresponding to 5 main functions of palmprint extraction

Function	Ratio: sequential/ parallel cycle number	Used slice numbers	Ratio: used slice numbers (%)	Freq. (MHz)	Necessary cycle numbers	Processing time (ms)
Thresholding	1.63	559	1.48	260	15,394,000	63.35
Erosion	2.72	1,345	3.57	273	21,176,000	77.57
Freeman	1.34	1,468	3.90	255	532,111	2.08
Point processing	1.31	4,190	11.12	190	418,947	2.20
Palm extraction	2.22	3,360	8.92	190	58,040	0.31

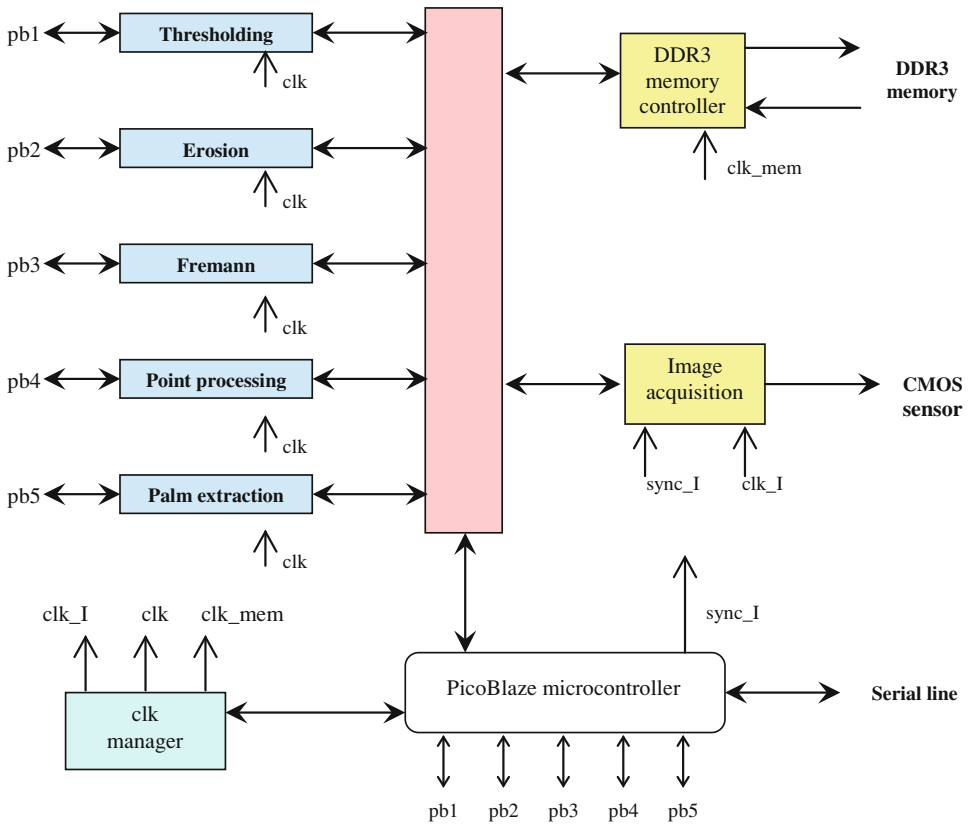

Fig. 8 Architecture of palmprint extraction hardware implementation on the Virtex-6

drives DDR3 memory using the Memory manager block. The clk_manager block allows us to generate corresponding clock for each VLIW processor execution frequency. In agreement with Fig. 8, obtained hardware implementation of entire extraction chain is given in Table 7. We can note that a small percentage of available hardware resources are used for this hardware implementation, and a processing time of 145.6 ms is obtained to extract a palmprint image.

Note that for this biometric application, pipeline execution of contactless palmprint extraction chain is not necessary. But, for some video applications, PicoBlaze would be able to manage pipeline execution of several flexible processors with memory manager study.

The implementation of the contactless palmprint extraction chain has also been simulated on a TMS320C64xx DSP platform of Texas Instruments, thanks to the Code Composer Studio tool. The C64x central processing unit (CPU) consists of eight functional units, two register files, and two data paths. Devices of the C64x family can execute, for example, four 16-bit multiplies every cycle, or eight 8-bit multiplies. They have a two-level memory architecture for program and data.

A DSP implementation description has been made in C language. We let the compiler of the CCStudio environment decide the possibilities of parallelization and optimization. Our simulations empirically show that the necessary number of CPU cycle for entire palmprint extraction is 390×10^6, which corresponds to 390 ms at a frequency of 1 GHz.

To illustrate the potential adaptability of our development cycle, we have also performed complementary experiments using erosion operation. We used a mechanism of loop

Table 7 Hardware implementation results of entire palmprint extraction chain

Used hardware resources	Slices	DSP48	RAM block
Number	12,624	12	0
Ratio (%)	33.5	1.5	0

Table 8 Hardware implementation results in function of the reduction factor of loop iteration

Reduction factor of loop iteration	Number cycle (10^3)	Slice ratio (%)	Speed (no. images/s)
1	21,176	3.6	13
2	10,616	10.4	24
4	5,336	25.7	45
8	2,696	59.6	86

unrolling present in OpenIMPACT optimization option. In fact, loop overhead can be reduced by reducing the number of iterations and replicating the body of the loop. Table 8 gives hardware implementation results with a reduction factor of loop iteration, respectively, equal to 1, 2, 4, and 8. We can observe that the processing speed (number of processed images per second) increases at the same time as the number of used slices.

Loop-unrolling operation can be considered as data-distribution processing. These preliminary results show that using our proposed development cycle, DLP can also be easily extracted. It can be used to determine the best compromise between the processing speed and the used hardware resources.

6 Conclusions and perspectives

In this paper, we present a new co-design SW/HW approach for computer vision applications. The proposed development cycle allows non-electronic specialists to realize rapid prototyping of image processing. Since they can transform automatically their C codes in VHDL description for FPGA implementation in an optimal manner. Our method is based on advanced compiler technology and uses the minimum-necessary hardware resources for a target application, thanks to flexible VLIW processor model.

Our approach has been tested and validated using firstly three common image-processing algorithms: Sobel filter, Convolution 3×3 and Fast DCT. In general, these algorithms use a small percentage of hardware resources available on FPGA and this allows considering others and complexes post-processing of image in the same FPGA device. We have also realized rapid prototyping of contactless palmprint extraction for a biometric application. This complex and less-regular ROI (region of interest) extraction is processed instantly on a FPGA Virtex-6 based board (145.6 ms vs. 390 ms on TMS320C64xx DSP of Texas Instrument). Through these experiments, we can conclude that our approach is a promising way of generating performance effective of embedded VLIW processors in FPGA and it applies some criteria for co-design tools: flexibility, modularity, performance, and reusability.

In perspectives, we want to test and valid the development cycle using another open compiler: LLVM infrastructure [33] for the conversion of C++ source to intermediate representation. This LLVM environment is more popular in the signal and image-processing topic than OpenIMPACT. We hope also to compare our approach with other advanced parallel implementation techniques.

This paper consists of only the basic stage of our complete tool suite: rapid prototyping of flexible VLIW processor for image processing. Some high-level parallelism (data-level and thread-level) can also be considered using the proposed method because of economic use of available hardware resources. On the other hand, with a given FPGA or given application, we can perform rapid design space exploration to respect simultaneously algorithmic and hardware constraints due to adaptive capacity of our approach.

References

1. Chen, Y.S., Shih, C.S., Kuo, T.W.: Processing element allocation and dynamic scheduling codesign for multi-function SoCs. Real-Time Syst. **44**(1–3), 72–104 (2010)
2. Kessal, L., Abel, N., Karabernou, S.M., Demigny, D.: Reconfigurable computing: design methodology and hardware tasks scheduling for real-time image processing. J. Real-Time Image Process. **3**(3), 131–147 (2008)
3. Dang, P.: VLSI architecture for real-time image and video processing systems. J. Real-Time Image Process. **1**(10), 57–62 (2006)
4. Botella, G., Martin, J.A., et al.: FPGA-based multimoda embedded sensor system integrating low- and mid-level vision. J. Sens. **11**, 8164–8179 (2011)
5. Aguirre, M.A., Tombs, J.N., et al.: Microprocessor and FPGA interfaces for in-system co-debugging in field programmable hybrid systems. Microprocess. Microsyst. **29**(2–3), 75–85 (2005)
6. Brost, V., Yang, F., Pandaivone, M., Farrugia, N.: Multiple modular VLIW processors based on FPGA. J. Electr. Imaging SPIE **16**(2), 023001(1–10) (2007)
7. Panda, P.R.: SystemC—a modeling platform supporting multiple design abstractions. Proceeding o the 14th International Symposium on System Synthesis, pp 75–80 (2001)
8. Loo, S.M., Wells, B.E. et al.: Handel-C for rapid prototyping of VLSI coprocessors for real-time systems. Proceedings of 34th Southeastern Symposium on System Theory, pp 6–10 (2002)
9. The Ocapi XL Initiative: http://www.imec.be/design/ocapi.org
10. Gelato.org: Impact advanced compiler technology. http://gelato.uiuc.edu (2009)

11. Brost, V., Yang, F., Paindavoine, M., Farrugia, N.: A modular VLIW processor. IEEE International Symposium on Circuits and Systems (ISCASS'07), New Orleans (2007)
12. Sadoun, R., Belouchrani, A., et al.: An FPGA based soft multiprocessors for DNS/DNSSEC authoritative server. Microprocess. Microsyst. **35**(5), 473–483 (2011). doi:10.1016/j.micpro.2011.3.004
13. Zou, L., Fu, Z.A., et al.: A pipelined architecture for real time correction of non-uniformity in infrared focal plane array imaging system using multiprocessors. Infrared Phys. Technol. **53**(4), 254–266 (2010). doi:10.1016/j.infrared.2010.3.001
14. http://www.latticesemi.com/products/intellectualproperty/ipcores/mico32/index.cfm
15. Uhrig, S., Shehan, B. et al.: The two-dimensional superscalar GAP processor architecture. Int. J. Adv. Syst. Meas **3**(1–2), (2010)
16. Jahr, R., Ungerer, T., Calborean, H. et al.: Automatic multi-objective optimization of parameters for hardware and core optimizations. Proceedings of the 2011 International Conference on High Performance Computing & Simulation (HPCS) (2011)
17. Koenig, R., Bauer, L. et al.: KAHRISMA: a nouvel hypermorphic reconfigurable-instruction set multi-grained-array architecture. 2009 Design, Automation & Test in Europe Conference & Exhibition (2010)
18. Lam, S.K., Srikanthan, T.: Rapid design of area-efficient custom instructions for reconfigurable embedded processing. J. Syst. Archit. **55**, 1–14 (2009)
19. Cyllenhaal J., Hwu, W., Rau, B.: HMDES version 2.0 specification. Technical Report IMPACT-96-3, University of Illinois, Urbana-Champaign (1996)
20. Kathail, V., Schlansker, M., Rau, B.: HPL-PD architecture specification, Technical Report HPL-93-80 (2000)
21. http://www.trimaran.org/docs/trimaran4_manual.pdf
22. Saptono, D., Brost, V., Yang, F., Wibowo, E.P.: Concept and development of modular VLIW processor based on FPGA. The Second International conference of ICSEM, Bangkok (2010)
23. Fihser, J.A., Faraboschi, P., Young, C.: Embedded computing: a VLIW approach to architecture, compilers and tools. Elsevier Morgan Kauffman, New York (2005)
24. Xilinx: Xilinx homepage. URL: http://www.xilinx.com/ (2009)
25. ISE: ISE design suite 11. URL: http://www.xilinx.com/tools/designtools.htm (2009)
26. Uzun, I.S., Amira, A., Bouridane, A.: FPGA implementations of Fast Fourier Transform for real-time signal and image processing. IEE Proc Vision Image Signal Process **152**(3), 283–296 (2005)
27. Bravo, I., Balinas, J., et al.: Efficient smart CMOS camera based on FPGAs oriented to embedded image processing. J. Sens. **11**, 2282–2303 (2011)
28. Virtex-6: Xilinx ds150 virtex-6 family overview. URL http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf (2009)
29. Ahmad, A., Amira, A., et al.: FPGA-based IP cores implementation for face recognition using dynamic partial reconfiguration. J. Real-Time Image Process. (2011). doi:10.1007/s11554-011-0221-x
30. Cao, T.P., Elton, D., Deng, G.: Fast buffering for FPGA implementation of vision-based object recognition systems. J. Real-Time Image Process. (2011). doi:10.1007/s11554-011-0201-1
31. Fons, M., Fons, F., et al.: FPGA-based personal authentification using fingerprints. J. Signal Process. Syst. (2011). doi:10.1007/s11265-011-0629-3
32. Poinsot, A., Yang, F., Brost, V.: Palmprint and face score level fusion: hardware implementation of a contactless small sample biometric system. Opt. Eng. **50**(2), 000000-1-000000-13 (2011). SPIE
33. The LLVM Compiler Infrastructure: http://llvm.org

Author Biographies

V. Brost received the master's degree and Ph.D. degrees from University of Burgundy. He is currently an associate professor at the University of Burgundy and member of Laboratory of Electronic, Computing, and Imaging (LE2I), France. His research interests are in the areas of rapid prototyping system, VLIW architecture, and Electronic Devices.

F. Yang received the B.S. degree in electrical engineering from the University of Lanzhou, China, in 1982 and the M.S. (D.E.A.) (computer science) and Ph.D. degrees (image processing) from the University of Burgundy, France, in 1994 and 1998, respectively. She is currently a full-time professor and member of LE2I CNRS-UMR, Laboratory of Electronic, Computing, and Imaging Sciences at the University of Burgundy, France. Her research interests are in the areas of patterns recognition, neural network, motion estimation based on spatio-temporal Gabor filters, parallelism and real-time implementation, and, more specifically, automatic face image-processing algorithms and architectures. Prof. Yang is member of the French research group ISIS (Information, Signal, Images and Vision), she livens up the theme C: Algorithm Architecture Adequation.

C. Meunier received the master's degree from the University of Burgundy, in July 2010. Since September 2010, He is a Ph.D. student in image processing and instrumentation at the Laboratory of Electronic, Computing, and Imaging. His research interests are in hardware implementations, real-time processing, electronic devices, and VLIW architecture concept.