FPGA-based many-core System-on-Chip design

Mouna Baklouti, Philippe Marquet, Jean-Luc Dekeyser, Mohamed Abid

FPGA-based Many-core System-on-Chip Design

M. Baklouti[a,*], Ph. Marquet[b], JL. Dekeyser[b], M. Abid[a]

[a] *Computer Embedded Systems (CES), Univ. Sfax, Tunisia*
National School of Engineers (ENIS), BP 1173, Sfax 3038, Tunisia
[b] *Univ. Lille, CRIStAL, UMR CNRS 9189, 59650 Villeneuve d'Ascq, France.*
Inria Lille Nord-Europe - Dreampal, France.

Abstract

Massively parallel architectures are proposed as a promising solution to speed up data-intensive applications and provide the required computational power. In particular, Single Instruction Multiple Data (SIMD) many-core architectures have been adopted for multimedia and signal processing applications with massive amounts of data parallelism where both performance and flexible programmability are important metrics. However, this class of processors has faced many challenges due to its increasing fabrication cost and design complexity. Moreover, the increasing gap between design productivity and chip complexity requires new design methods. Nowadays, the recent evolution of silicon integration technology, on the one hand, and the wide usage of reusable Intellectual Property (IP) cores and FPGAs (Field Programmable Gate Arrays), on the other hand, are attractive solutions to meet these challenges and reduce the time-to-market. The objective of this work is to study the performances of massively parallel SIMD on-chip architectures

*Corresponding author, Fax: 0021674404154
Email addresses: mouna.baklouti@enis.rnu.tn (M. Baklouti), philippe.marquet@lifl.fr (Ph. Marquet), jean-luc.dekeyser@univ-lille1.fr (JL. Dekeyser), mohamed.abid@enis.rnu.tn (M. Abid)

with current design methodologies based on recent integration technologies. Flexibility offered by these new design tools allows design space exploration to search for the most effective implementations. This work introduces an IP-based design methodology for easy building configurable and flexible massively parallel SIMD processing on FPGA platforms. The proposed approach allows implementing a generic parallel architecture based on IP assembly that can be tailored in order to better satisfy the requirements of highly-demanding applications. The experimental results show effectiveness of the design methodology as well as the performances of the implemented SoC.
Keywords: Field Programmable Gate Arrays, Intellectual Property, Single Instruction Multiple Data, System-on-Chip, intensive signal processing.

---

1. Introduction

Modern embedded systems tend to be more and more sophisticated with the integration of multiple functionalities in the same system, often implemented on a single chip, called System-on-Chip (SoC). Adding to that, the wide spread of data-intensive applications, such as multimedia applications, medical imaging, numerical filtering, radar or sonar signal processing, etc., requires powerful architectures with higher execution performances. With the huge number of transistors available in today chips, and the stagnation of clock frequencies due to power dissipation issues, chips with multiple processor cores are becoming more commonplace, in particular Single Instruction Multiple Data (SIMD) on-chip architectures with intensive parallel computations are possible. SIMD architectures are widely recognized as being well-suited for media-centric applications like image and video processing

applications [2] because they can efficiently exploit massive data parallelism available with minimal energy [1]. As most multimedia algorithms operate on relatively independent groups of data, such as sound frame samples, rows of pixels, or video frames, it is often easier to use a data-oriented perspective to parallelize these algorithms. SIMD on-chip systems offer many advantages. The SIMD processing presents a computational efficiency originating from the reduced overhead of control operations. The cost of instruction and address decoding is amortized over the many processing elements [34]. Adding to that, the SIMD parallel architecture enables higher performance for intensive data-parallel applications at lower power consumption [3, 33]. The design of these high-performance embedded systems for signal processing applications is facing the challenges of not only increased computational demands but also increased demands for adaptability to different requirements for these applications.

Various SIMD on-chip implementations have been proposed. Most of them are dedicated solutions for a specific application (many processors with short memory/small amount of processor with large memory/a given interconnection network). This normally results in good performance for the targeted application; however the performance of other applications may not be so good due to the diversity of parallel application requirements. Their design approaches also necessitate a long development time if some optimizations or modifications need to be made.

From the above observations, there is clearly a need of a generic SIMD prototype that can satisfy different application requirements. The proposed design methodology is based on IP assembly to easily and rapidly imple-

ment different SIMD configurations. The provided parallel architecture is programmable, parametric, extensible and configurable; presenting high performances for data-parallel algorithms. FPGA (Field Programmable Gate Arrays) are targeted devices to implement the proposed design. Compared to ASIC (Application Specific Integrated Circuit), FPGA requires much less implementation costs and offers more flexibility. FPGA can also be re-wired and remotely reconfigured at any time. The inflexible and costly ASICs give FPGA-based solutions an upper hand in terms of implementation flexibility and cost effectiveness.

This paper presents an SIMD massively parallel processing System-on-Chip. This system is an FPGA IP-based programmable system. Its architecture is flexible and can be customized to satisfy the requirements of data-intensive parallel applications. This opens a rich design space to the user and allows greater area/performance tradeoffs. We propose a hierarchical design implemented at Register Transfer Level (RTL) using the VHDL (VHSIC Hardware Description Language) language.

The remainder of this paper is organized as follows. The next Section presents some significant work related to the design issues of SIMD parallel on-chip architectures. It also deals with some proposed IP-based design frameworks. Section 3 introduces the SIMD massively parallel platform model. Section 4 describes the IP assembly approach. Section 5 highlights the data-parallel programming. In Section 6, performance evaluation is given through prototype implementation and video processing computations. Section 7 discusses the efficiency of the proposed SIMD on-chip system and gives comparisons with other implementations. Finally, Section 8 concludes this

paper with a brief outlook on future work.

2. Related work

SIMD architectures have been proposed, implemented, and thoroughly studied for almost 50 years [4]. Early SIMD machines were designed to provide supercomputing levels of performance [5, 14, 6, 7, 8]. However, the end of the 90s saw the decline of SIMD machines due to their high fabrication cost, the impossible instruction broadcast at a high clock frequency and their scalability limits in terms of clock distribution and synchronous communications. Recently, the increase in the integration density and the use of novel design approaches as IP reuse [9] make possible the implementation of complex systems with reduced cost. Advances in technology have led to a wide range of SIMD systems: from SIMD arrays integrated into memory chips [10] to SIMD arrays exploiting sub-word parallelism in multimedia instruction set enhancements [11, 12] (Intel SSE, AMD 3dNOW, Motorola AltiVec). In this work, we focus on FPGA-based massively parallel architectures. Some proposed examples are presented in the next paragraphs.

A scalable streaming-array (SSA) architecture is presented in [19]. SSA is an unidirectional linear array of pipelined-stage modules (PSMs), implemented on multiple FPGAs. Each PSM, implemented as an independent SIMD processing unit, consists of programmable simple PEs connected with a bi-directional 1D torus network. The PE's implementation is tailored to perform stencil computations. This work proposes a dedicated architecture for high-performance stencil computations on FPGAs. Moreover, a multi-chip solution would mean higher developmental effort and a larger power

expenditure by the system compared to single-chip. In [20], an FPGA-based pixel array processor is presented. The hardware architecture comprises of primitive pixel processors that use bit-serial arithmetic to compute. Each processor processes a single pixel and is connected in a 2-dimensional mesh topology to form the overall array processor. The carried implementation is dedicated to perform Laplacian filtering on a 40 by 40 pixel gray scale video. Tanabe et al. [24] propose an FPGA-based SIMD processor. An opencore processor that belongs to SuperH processor series was modified to include more number of computation units and then performs SIMD processing. A JPEG case-study was presented however no idea about programming was given. The achieved clock frequency of the SIMD processor is too low (55 MHz) while many soft CPUs run faster these days. An SIMD architecture with 31 processing elements (PEs) is implemented on Xilinx FPGA [25]. It is dedicated to run a motion estimation application. Another FPGA-based SIMD architecture with 30 PEs is proposed [26]. It is specific for edge detection performing the convolution of a mask over an image. An SIMD programmable processor dedicated to image processing sensors is implemented on FPGA, based on a self designed soft IP processor cell PicoBlaze [27]. The 1-bit PEs can only execute simple computations on binary images. These work remain appropriate solutions. While these application specific systems deliver good performance and efficiency, they lack the programmability and versatility required to support the changing standards of multimedia. Their design is also time consuming, and costly process, increasing the overall production costs. Although some implementations are based on IP design, the designers do not always find the suitable IPs that match the application.

Some IPs have excess performance and some do not have enough performance.

Many researchers propose soft vector processor architectures to accelerate data parallel applications. Yiannacouras et. al. [16] described an implementation of the VESPA FPGA-based vector processor. The proposed processor can be customized in terms of vector chaining and vector lanes. It consists of a scalar MIPS-based processor coupled with a parameterized vector coprocessor based on the VIRAM vector instruction set. In [17] a new soft vector architecture, named VEGAS, is presented. VEGAS consists of a standard NiosII/f processor, a vector core, a DDR2 controller plus external memory, and a DMA engine. Vector and DMA instructions are implemented as custom instructions in the regular Nios II instruction stream. For programming, the programmer needs to manually manage vector address registers, which is error-prone. In [18], a soft vector processor for FPGA applications, named VENICE, is presented. VENICE can be used to accelerate tasks that fit the SIMD programming model. It is implemented as an accelerator to the Nios II fast processor. In this case, the Nios executes all control flow instructions and issues instructions to the VENICE vector core as one or two tandem Nios custom instructions. The drawback of the VENICE implementation is its dependency on the Nios processor and hence can only be exploited in Altera FPGAs. The design isn't so flexible to fit various processor architectures. Generally, the disadvantages of vector processing is its inefficiency when dealing with irregular parallelism and the memory can easily become a bottleneck especially if data is not mapped appropriately to memory banks.

Our work considers SIMD on-chip implementation more generally and

thoroughly explores a large design space. The proposed massively parallel SoC extends these work since it provides a configurable and parametric architecture in order to be suited to a wide range of data-parallel applications. The IP-based methodology followed to generate one configuration provides a short development time and enough flexibility. An IP library is provided with the RTL design to help the designer selecting the suitable IP that delivers the highest performance for his targeted application. The proposed implementation is parametric and hierarchical making any modification and regeneration of the hardware parallel configuration easy. This design methodology provides lots of flexibility and enables to explore SIMD architectures. In the next section, the basic SIMD SoC platform model is described.

3. SIMD Parallel SoC Model

In this section, the basic SIMD parallel SoC model is exposed. The work deals with the very typical and simple SIMD model inspired from traditional SIMD systems mainly the famous MasPar [28]. The parallel system is designed for FPGA prototyping. Based on FPGA programmability and by using replication of IPs effectively, massively parallel architectures can achieve high performance at low cost. The proposed system supports the following features:

- a main processor, the Array Controller Unit (ACU), connected to instruction *ACUIns* and sequential data memory *ACUData*. It synchronously controls the whole system.

- a parametric set of elementary processing elements (PEs), each one

connected to a local data memory *PEM* (each PE has its own private memory). All PEs perform parallel computations.

- ACU/PE interface: this interface is devoted to send control/execution orders to all PEs. It is modeled as a bus connecting the ACU with all PEs.

- a configurable neighboring interconnection network to connect PEs with their neighbors.

- a Network-on-Chip (NoC), working as a global router to perform point-to-point communications through different types of connections.

- PE activity mechanism: each PE executes the parallel instruction if its activity bit is set to '1'. This bit is controlled by the ACU via control instructions. This is especially useful in the case of conditional instructions.

- OR Tree: it computes logic global OR of all PE activity bits to test the status of PEs. It allows the ACU to know if at least one PE is active.

Figure 1 highlights the proposed parallel platform with a focus on its IP-based design. The used IPs are mainly memories (RAM, ROM), processors and routers. The SIMD model is flexible and can be customized to target diverse applications.

The whole system is synchronously controlled by the ACU, which is responsible for fetching and decoding instructions, computing addresses and scalar data values, issuing control signals to the PE array, and monitoring the status of the PE array. A data-parallel program is composed of sequential

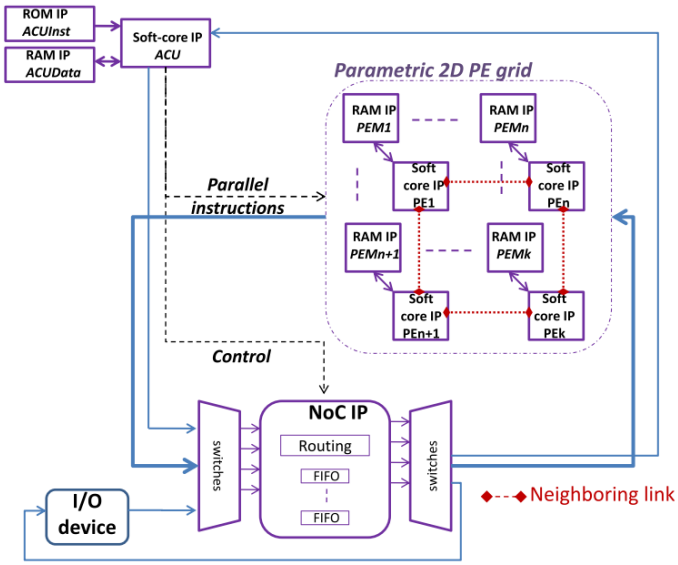

Figure 1: IP-based SIMD SoC platform

as well as parallel instructions. The ACU executes any sequential instruction and transfers parallel instructions to the PE array. The ACU occupies in fact two main roles: a processor to execute sequential instructions and a controller to control the whole system. It has also to decode parallel instructions when PEs are reduced processors. It cannot be implemented by a simple Finite State Machine (FSM). The ACU can be replaced with a full complex FSM. However, our aim in this work is to reuse available IPs to facilitate the design process and not to implement the component from scratch. So, a simple way is to use a simple processor to implement the ACU as well as the PE. Using the same IP for both ACU and PE facilitates the interconnection between the two components and guaranteed the fact that both processors use the same instruction set.

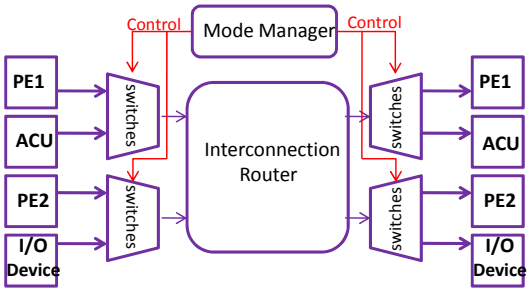

Figure 2: Simple example of NoC configuration

Masking schemes and activity states are used to control the status of each PE during the execution of an instruction. Each PE, identified in the system by a unique number, may be either enabled or disabled; only enabled PEs perform computation.

The problem of several massively parallel architectures is their inability to respond the need of a high bandwidth of input/output. To face this problem, the proposed parallel SoC contains a point-to-point communication network, which is a multi-purpose NoC component in the architecture. Figure 2 illustrates a simple example of the NoC architecture in a 2-PE SIMD SoC configuration. This network is able to connect many I/O devices to the ACU and PEs. It has three functions: connecting in parallel any PE with another one (global router), connecting the PEs to the hardware devices offering parallel I/O transfers and connecting the ACU to any PE.

The contribution of this paper is to design an SIMD architecture conformed to the generic model and adapted to the application. A parametric component-based approach is proposed to design flexible SIMD system at RTL level. The architecture is based on IP assembly mainly processor,

memory and network IPs. The modular architectural model can include only the needed components to execute a given application. It alleviates the complexity of the system, in particular in the case of on-chip implementation. This programmable FPGA-based parallel system offers productivity and time-to-market advantages, and allows algorithms to be easily modified without changing the FPGA bitstream, which could otherwise lead to convergence issues such as timing closure.

The following section details the parallel SoC implementation.

4. IP-based massively parallel SIMD design

The proposed massively parallel RTL implementation is hierarchical and parametric, allowing the designer to build different SIMD SoC hardware architectures, in order to select the best configuration with the highest performances for a given data-parallel application. This implementation is based on IP assembly. For this purpose, an IP library is provided with the design, containing different IPs that can be used to generate different SIMD configurations.

4.1. IP-Library

The following paragraphs detail the provided IPs. While some IP construction, such as memories, does not require much attention, some others, such as processors, are much more complex. The following paragraphs deal with particularities of each IP block and its integration into the SIMD SoC.

Table 1: Consumed logic of the miniMIPS processor and the reduced miniMIPS processor

Processor	ALUTs	Registers
miniMIPS	3339	1936
reduced miniMIPS	1265	1318

4.1.1. Processor IP

The proposed design is based on Open hardware concept. So, open-source processor IPs are mainly chosen to implement SIMD configurations. To build ACU and PE processors, two methodologies are proposed. The first one, called *the processor reduction methodology*, is based on refining the main processor in order to obtain a small reduced one. In the second methodology, called *the processor replication methodology*, the PE is chosen to be the same processor as the ACU to reduce the design time and facilitate the architecture building.

Processor reduction methodology

The PE is a reduced processor derived from the same ACU processor and only responsible of executing instructions. It receives decoded instruction from the ACU, thus its instruction memory is eliminated. The reduction simplifies the PE by eliminating its instruction logic and instruction memory, and thus saves millions of gates and hundreds of megabytes of memory in the overall system. As example, Table 1 compares between the processor miniMIPS and its reduced version in terms of consumed logics.

It is clearly shown that the reduced miniMIPS occupies smaller area (approximately the half surface) than the miniMIPS. Thus, integrating reduced processors allows to gain surface in order to implement a massively parallel system.

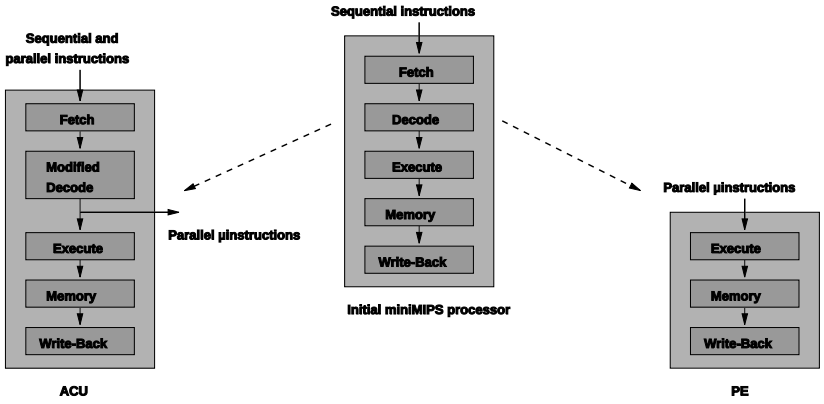

Figure 3: Processor reduction methodology applied on miniMIPS processor

The connection between ACU and PEs is a bus of micro-instructions delivered from the ACU decode stage to the input of the PE execute stage [32]. The reduction methodology was tested on the miniMIPS processor (Fig 3) [35] and on the OpenRisc processor [36, 29]. It enables to construct smaller and less complex PE based on IP reuse, and so integrate a large number of PE on a single chip.

Processor replication methodology

In this methodology, ACU and PE are similar (the same processor IP) in order to facilitate and accelerate their design. Compared to the reduction methodology, each PE receives parallel instructions from the ACU (Fig 4). The replication methodology offers a large gain in the development time. However, we are unable in this case to integrate a large number of PEs in a single chip. In this case, we have to choose a smaller processor that can be fitted in large quantities into the FPGA.

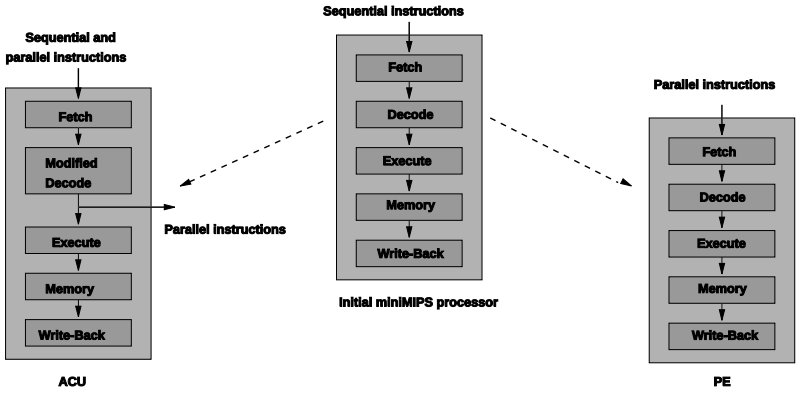

Figure 4: Processor replication methodology applied on miniMIPS processor

4.1.2. Memory IP

Two memory components are distinguished in the proposed SIMD system: the sequential memory that contains data and instructions, attached to the ACU and the data memory attached to each PE. The memory size is a parametric value in the architecture. The user can set the needed size according to the application requirements. High bandwidth is achieved by accessing the memory structures in parallel.

4.1.3. Network IPs

The massively parallel system integrates two communication networks: a regular one to connect each PE to its neighbors, and an irregular one to assure point-to-point communications [30]. The designer can use none, one or both routers to build the needed parallel configuration depending on the application requirements. The communications are managed through communication instructions that will be described later.

Global network

The global network is designed to perform any communication pattern between hardware components (PE-PE, ACU-PE, PE-I/O device). It provides global communication between all PEs and forms the basis for the parallel I/O system. It mainly consists of a communication mode manager and an interconnection network IP. The mode manager is responsible of establishing the needed communication mode.

The internal interconnection network transfers data from sources to destinations and can have different types (shared bus, full crossbar, Delta Multistage Interconnection Network). The NoC IP is parametric (in terms of size and internal network) to offer powerful integration in various systems [31].

Neighborhood regular network

This network performs neighbor communications between PEs and can have different topologies (linear array, ring, mesh, Torus, and Xnet (a two-dimensional mesh with extra diagonal links)). It may connect the data in a specific direction depending on the destination as well as the distance, which defines the number of paths between every couple of PE sender and PE receiver. It is composed of a controller and several routers (equal to the number of PEs in the system). The architecture of the router IP is depicted in Fig 5. The data communicated through the router are not stored in FIFO registers since the neighborhood communications are synchronous and all performed in a single direction at a given time. An SIMD interconnection function is a bijective function, thus the data transfer occurs without conflicts. In every communication, each router only activates the required link to the needed direction. The other links will be disabled, reducing the power consumption.

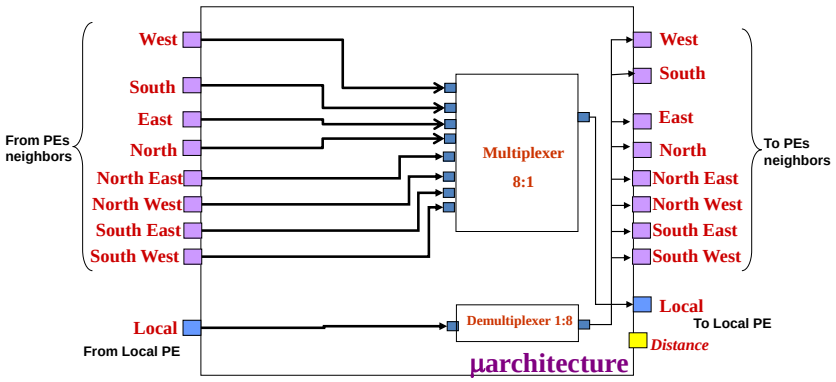

Figure 5: Regular router IP architecture

Up to this point, we can see that one of the great properties of the proposed SIMD parallel SoC platform is that the same architectural model can satisfy the requirements of various applications. The proposed VHDL implementation is hierarchical, modular and parametrized in terms of the number of PEs and their arrangement, the amount of local PE memory as well as the ACU memory, the neighborhood network topology and the type of the interconnection network in the global NoC. The following Section gives a brief overview of how to execute a data-parallel program on the designed parallel system.

5. Data-Parallel programming

The SIMD SoC is programmed by a single instruction stream partitioned into sequential and parallel instructions. It is programmed in a manner similar to inline assembly in C. However, C macros are used to simplify programming and make instructions look like C functions without any run

time overhead.

A sequential instruction is carried out by the ACU as in the usual sequential architecture. A parallel instruction is executed in a synchronous manner by all active PEs, each PE taking its operands from its local memory and storing the result in this same memory (or may be in its own local registers). The massively parallel system instruction set is derived from the processor IP's instruction set used in the design. We distinguish different instructions classified into five main groups: arithmetic (add/sub/mul/div), memory access, communications, jump and branch, and system control. The first three types are executed both by ACU and PE, whereas the two others are only executed by the ACU. The arithmetic and memory instructions are duplicated for the PE. So that, in the instruction set we distinguish for example between a sequential addition and a parallel addition. This is accomplished through implementing the parallel instruction in the processor instruction table while modifying its decoding. Communication instructions are encoded from the processor instructions mainly load LW and store SW instructions. Other system control instructions are also added and rely on LW and SW instructions too. These instructions consist on storing/loading data to/from special registers or memory addresses to accomplish their functions. Examples of these instructions control the NoC's switching mode, the PE's activity state, etc. Table 2 details these instructions' coding.

6. Experiments: video processing-based algorithms

In this section, different massively parallel on-chip configurations are designed and tested to run video processing-based algorithms. Performance

Table 2: New instructions

Macro ASM	Description	Coding	
		miniMIPS	OpenRisc
SET_MODE_NOC (reg)	NoC mode instruction: select comm. mode (reg value).	SW reg,0x9003(r0)	l.addi r1,r0,0x9003 l.sw 0x0(r1),reg
Modes 0,1 and 4: P_NOC_SEND (reg,dest,adr) **Modes 2 and 3:** P_NOC_SEND (reg,adr)	NoC SEND instruction: transfer data reg to the needed destination specified by dest.	p_addi r1,r0,dest p_addi r1,r1,adr p_SW reg,0(r1)	l.addi r1,r0,dest l.addi r1,r1,adr l.sw 0x0(r1),reg
		p_addi r1,r0,adr p_SW reg,0(r1)	l.addi r1,r0,adr l.sw 0x0(r1),reg
Modes 0,1 and 4: P_NOC_REC (reg,src,adr) **Modes 2 and 3:** P_NOC_REC (reg,adr)	NoC RECEIVE instruction: read data from appropriate sender.	p_addi r1,r0,src p_addi r1,r1,adr p_LW reg,0(r1)	l.addi r1,r0,src l.addi r1,r1,adr l.lwz reg,0x0(r1)
		p_addi r1,r0,adr p_LW reg,0(r1)	l.addi r1,r0,adr l.lwz reg,0x0(r1)
P_REG_SEND (reg,dir,dis,adr)	Neighboring SEND instruction: transfer data (of reg) from source to destination.	p_addi r1,r0,dir p_addi r1,r1,dis p_addi r1,r1,adr p_SW reg,0(r1)	l.addi r1,r0,dir l.addi r1,r1,dis l.addi r1,r1,adr l.sw 0x0(r1),reg
P_REG_REC (reg,dir,dis,adr)	Neighboring RECEIVE instruction: read data from appropriate sender.	p_addi r1,r0,dir p_addi r1,r1,dis p_addi r1,r1,adr p_LW reg,0(r1)	l.addi r1,r0,dir l.addi r1,r1,dis l.addi r1,r1,adr l.lwz reg,0x0(r1)
P_GET_STATUS (reg,ident)	read activity bit of the PE identified by "ident".	p_lui r1,0x9 p_ori r1,r1,0 p_addi r1,r1,ident p_LW reg,0(r1)	l.movhi r1,0x9 l.addi r1,r1,ident l.lwz reg,0x0(r1)
P_SET_STATUS (val,ident)	modify activity bit of the PE (identified by "ident") by "val" value.	p_lui r1,0x9 p_ori r1,r1,0 p_addi r1,r1,ident p_addi r2,r0,val p_SW r2,0(r1)	l.movhi r1,0x9 l.addi r1,r1,ident l.addi r2,r0,val l.sw 0x0(r1),r2
P_GET_IDENT (reg)	get identity	p_lui r1,0x2 p_ori r1,r1,0 p_LW reg,0(r1)	l.movhi r1,0x2 l.lwz reg,0x0(r1)
GET_OR_TREE (reg)	read the value of the OR_Tree	addi r1,r0,0x9005 LW reg,0(r1)	l.addi r1,r0,0x9005 l.lwz reg,0x0(r1)

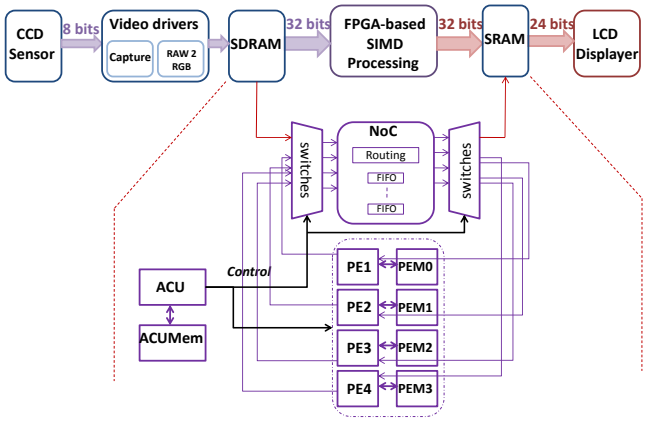

Figure 6: FPGA-based Video Processing Chain

results, observations and detailed analysis are reported. Through experimental results, the designer can choose the adequate SIMD configuration.

The parallel SoC performs parallel pixel processing on a captured video sequence obtained from a video camera driver. In fact, video is captured by an analog camera then digitized by a video decoder. The data from the video decoder is stored in an SDRAM memory and then processed using the FPGA-based massively parallel system. Computed results are then stored in an SRAM memory to be transmitted to a LCD displayer. The design is implemented on the Altera DE2 70 development board [37] equipped with a Cyclone II FPGA (EP2C70F896C6) with 68416 Logical Elements (LE) and 250 M4K RAM blocks. It is equipped with two 32 Mbytes SDRAM and 2 Mbyte SSRAM. Briefly, the design is composed of a camera TRDB D5M equipped with 5 Mega-Pixel CMOS sensor and working with 15 frames per second, an SDRAM to store pixel data coming from the camera, the parallel architecture to process the pixels, an SRAM to store computed data and the

800×RGB×480 LCD TRDB LTM displayer working at 33.2 MHz (Figure 6).

6.1. Capturing and displaying video

In this design, the video processing system carries out the capture, buffering and video display. The SIMD parallel architecture is configured and explored with different number of PEs and with the global NoC to manage the needed I/O communications since I/O transfers present a performance challenge for this kind of high performance computing systems. To validate the configurability of the processor IP, two processors are chosen: miniMIPS [35] and OpenRisc [36]. The two processor design methodologies (reduction and replication) are also tested. The NoC is implemented based on a crossbar interconnection network in order to assure a rapid data-parallel transfer to all PEs. It is chosen because it is a non blocking fully connected network in comparison with a multi-stage network, which is blocking and to the bus that has a limited bandwidth. Based on the parametric VHDL implementation, it was easy to generate and synthesize these different parallel VHDL configurations. The FPGA based synthesis results are presented in terms of consumed LEs as well as total memory bits in the case of the two methodologies. The consumed power and the maximum obtained frequency are also illustrated (Table 3 and Table 4).

Tables 3 and 4 show, as expected, that the processor reduction methodology allows implementing a large number of PEs on a single chip. With the processor replication methodology we can not put 16 miniMIPS PEs on the CycloneII FPGA, whereas with the reduction methodology up to 32 miniMIPS reduced PEs can be integrated on the same FPGA device. Comparing between the two used processors, we notice that the miniMIPS

Table 3: Synthesis results - Processor reduction methodology

Nb PEs	Processor IP	Logic Utilization			Total memory			Power cons. (mW)	Fmax (MHz)
		Comb. functions	registers	% LE	ACU (bytes)	PE (bytes)	% Mem		
4	miniMIPS	14176	4762	23	4096	4096	24	852	89.21
8	miniMIPS	21425	6006	34	4096	4096	33	940	87
16	miniMIPS	35673	8366	55	4096	4096	52	1296	85.82
32	miniMIPS	61130	13006	93	4096	2048	66	1969	83.04
4	OpenRisc	19674	7678	30	4096	4096	15	911	188.61
8	OpenRisc	34667	13534	53	4096	4096	22	1205	186.06
16	OpenRisc	64270	25160	98	4096	4096	36	1999	185.22

Table 4: Synthesis results - Processor replication methodology

Nb PEs	Processor IP	Logic Utilization			Total memory			Power cons. (mW)	Fmax (MHz)
		Comb. functions	registers	% LE	ACU (bytes)	PE (bytes)	% Mem		
4	miniMIPS	21971	9243	37	4096	1024	11	1207	93.12
8	miniMIPS	42059	16863	71	4096	1024	18	1799	90.91
4	OpenRisc	18491	10229	41	4096	1024	13	1298	192.52
8	OpenRisc	40185	23966	91	4096	1024	22	1921	189.97

occupies smaller area than the OpenRisc and slightly has a reduced power consumption. Another important issue is the consumed memory. From Table 4, we notice that when doubling the number of integrated PEs, the total memory blocks will be 1.6% higher. The major SoC constraint is the limited amount of on-chip memory. Results also show that the power consumption slightly increases when doubling the number of integrated PEs because the consumed power depends on the number of components in the architecture as well as the interconnection node capacitance, circuit voltage and switching frequency [39].

In this design, each PE reads one pixel at a time from the SDRAM through the global network, then sends it in the same manner to the SRAM. This NoC performs the PE-I/O communication mode: PE-SDRAM(I/O) and PE-SRAM(I/O). The displayer is directly connected to the SRAM to

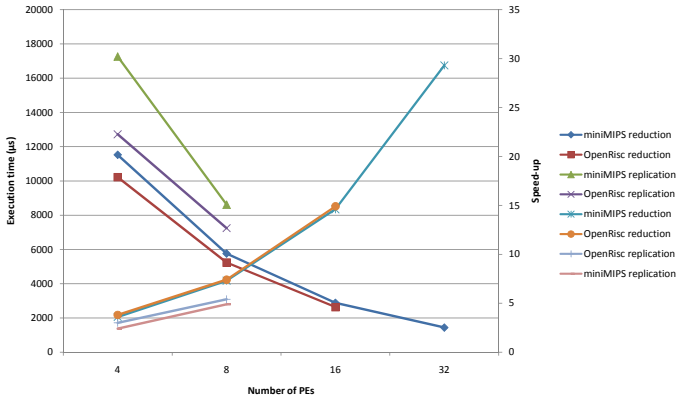

Figure 7: Experimental results to process one image of size 800×480 pixels

read and display pixels. The video camera and LCD displayer drivers are implemented in full hardware, using VHDL language.

The execution time results as well as the speed up are presented in Figure 7 depending on the number of PEs, the used processor IP and the applied design methodology. While increasing the number of PEs, we demonstrate the parallel SoC's scalability since it refers to the ability of the I/O bandwidth to increase as the number of processors participating in I/O activities grows. It is shown that the SIMD system achieves good performances when increasing the number of PEs working in parallel. As expected, the reduction methodology allows reducing execution times comparing to the replication methodology. A 4-PE configuration based on reduced miniMIPS is approximately 1.5x faster than the same configuration based on replicated miniMIPS. This is due to the complexity of the processor replication, induced by the extra decoding charge added in all PEs. We clearly face a compromise between design/execution time when choosing the processor design method-

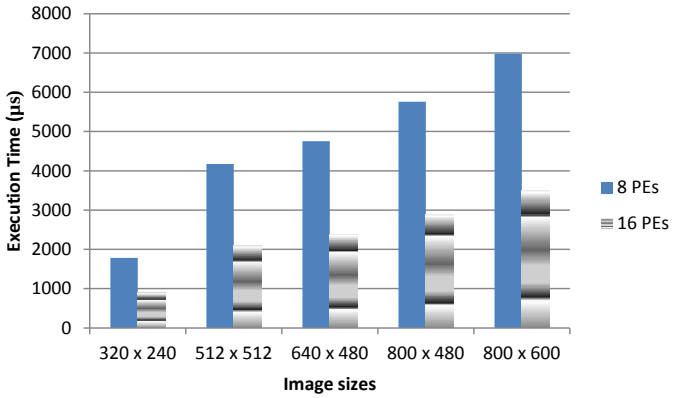

Figure 8: Experimental results to process different sized images on reduced miniMIPS PEs

ology. The obtained execution times show that the OpenRisc achieves good results. A speed up of 7.49 can be reach with 8 PEs. The miniMIPS based configurations can also have the same performances when integrating more reduced PEs with lower power consumption. It is clear from Figure 7 that the speed up follows the increasing number of PEs in the SIMD SoC. The better speed up is achieved using the reduced miniMIPS.

According to previous results, the reduced miniMIPS is chosen to build hardware SIMD configurations for the following experiments.

Figure 8 shows the processing time results obtained on reduced PEs (8 and 16) with different image sizes. Figure 8 highlights that the processing time scales with the image size. For example, the time needed to process one image of size 640×480 is 2.6 higher than the time spent to process an image of size 320×240. This demonstrates the performance and the scalability of the system since it depends on the number of processors and the size of the data being worked on.

6.2. RGB to YIQ color conversion

RGB to YIQ color conversion is used in many encoders where the RGB inputs from the camera are converted to a luminance (Y) and two chrominance information (I,Q). This algorithm explores multiply/accumulate capability, which is especially dedicated to SIMD and VLIW architectures. This conversion's type can be accomplished using the following equations:

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (1)$$

For this application, SIMD configurations with many PEs working in parallel are needed to assure real-time processing. We also need to integrate the global NoC to perform I/O data transfers. Based on the parametric RTL implementation, SIMD configurations are easily built. They contain the following features: many PEs (8, 16 and 32), miniMIPS processor, reduction methodology, ACU with a memory of 4096 bytes and each PE with a memory of 128 bytes, and a crossbar based NoC.

The execution time results as well as the speed up values are presented in Figure 9 depending on the number of PEs. The number 1 in the x-axis means that the SIMD configuration only contains the ACU to measure the sequential execution time. As shown in the previous experiment, the time reduces when increasing the number of PEs. Compared to the first application, only computing operations are added while maintaining the same communication transfers through the NoC. Thus, we clearly notice that the speed up is almost the same. To reach real-time, a minimum of 8 PEs is needed in this application. Each pixel processing should not exceed 30.12 ns.

An image filter convolution is tested in the following subsection.

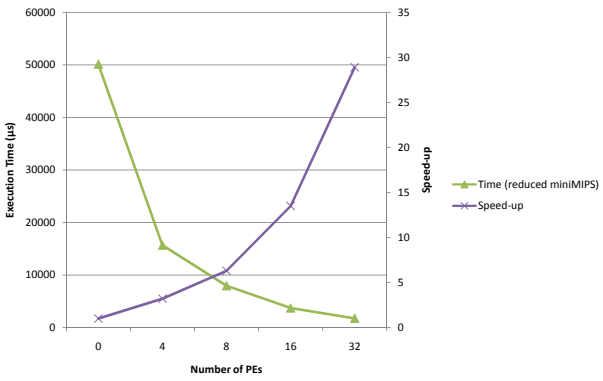

Figure 9: Color conversion experimental results

6.3. Image filter convolution

An image's convolution with a mask is tested. As a case study, a sharpen filter (3x3 kernel) is selected: $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$. The convolution of the mask can be done in parallel following the SIMD functioning so that multiple pixels can be processed at a time. The image convolution algorithm is quite similar to the color conversion algorithm since they perform image processing computations. Therefore, according to previous results we consider that we need more than 16 PEs to assure real-time processing.

The two tested parallel SIMD configurations are composed of an ACU, 32 PEs (each PE is connected to its local data memory) and a crossbar based NoC. The second configuration also integrates a 2D torus neighboring interconnection network compared to the first one. Table 5 shows the needed FPGA resources for the two designs obtained via Quartus tool. It is clear that the second parallel configuration couldn't be fitted on the Cyclone II FPGA.

To compute N convolved pixels at a time, each PE reads five pixels at

Table 5: Synthesis results for the convolution SIMD designs

Neighb. network	Logic Utilization			Total memory		
	Combin. functions	reg	%	ACU (bytes)	PE (bytes)	%
none	61130	13006	93	2048	600	22
configured 2D Torus	81796	22478	119	2048	600	25

a time (corresponding to the 5 non-zero filter coefficients), does the same computations and stores the resultant pixel. Each PE needs to have at least a local memory with 150 words (600 bytes). The LCD displayer begins reading the resultant data from the SRAM after the initialisation phase and the processing of the 32 first pixels in order to assure real-time processing. The processing of one pixel doesn't exceed 30.12 ns since it is the elapsed time to read one pixel by the LCD. In this algorithm, to process one pixel each PE does 5 memory loads, 1 multiplication, 4 negations, 4 additions and 1 memory store. Through prototyping, a pixel-processing time equal to 28.75 ns $<$ 30.12 ns is reached.

This implemented algorithm doesn't consider the neighboring pixels between PEs. So, a second algorithm version is implemented to take into account these pixels. In this case, the design can not be fit in the FPGA (table 5). The simulation results have shown a pixel-processing time equal to 47.5 ns $>$ 30.12. Thus, an SIMD SoC integrating more than 32 PEs and an FPGA with more resources than the Cyclone EP2C70 are needed to respond to real-time constraints.

All these experiments show the scalability of the proposed parallel system, demonstrates that its peak performance scales linearly with the number of PEs and proves the efficiency of the proposed design methodology to explore different parallel configurations in order to choose the best one.

Table 6: Comparison between SIMD SoC and 2D systolic architectures

System	Image size	Number of rows processed in parallel	Processing Time (Ms)
SIMD SoC (32 PEs)	512x512 (gray-level)	15	3
2D systolic (49 processors)	512x512 (gray-level)	15	5

Table 7: Comparison between SIMD SoC and C2H HW acceleration

System	Speed up	Frequency (MHz)
SIMD SoC	17x	184
C2H	13.3x	95

7. Performance

For performance comparisons, we adopted the image filter convolution benchmark. The tested parallel SoC is based on replicated NiosII processors, running at 200 MHz. Our system is firstly compared to a special purpose FPGA-based 2D systolic architecture dedicated to implement window-based image processing algorithms [13]. Table 6 summarizes the obtained results. Results show the efficiency of the proposed system since it achieves better processing time results compared to a 2D systolic architecture even with smaller number of PEs (32 compared to 49).

The SIMD system is then compared to hardware acceleration based on the C-to-HW (C2H) acceleration compiler of the Nios processor [23]. We ran the application on the NiosII/f processor at 200 MHz to establish the baseline performance. The two executions are compared in Table 7. Compared to a dedicated HW acceleration, the proposed parallel and programmable SIMD system achieves a better speed up. This demonstrates the efficiency of the SIMD SoC with the advantage of flexibility compared to a HW based implementation.

Table 8: Comparison between SIMD SoC and SIMD instruction extension

SIMD instructions extension	CPP	SIMD system	CPP
C version	34.7	ACU version	62.66
C version with SIMD instructions	17.4	SIMD version	12.85
Speed up	2	Speed up	4.8

Table 9: Comparison between SIMD SoC and [24]

Proposed SIMD SoC	Fmax	Power (mW)	[24]	Fmax	Power (mW)
4 PEs	89.21	852	4 CUs	56.2	1149
8 PEs	87	940	8 CUs	55.3	1233
16 PEs	85.82	1296	16 CUs	52.7	1385

We also compare our system to the NIOS implementation based on customized SIMD instruction extension [21]. Table 8 shows experimental results. The results are presented with the Cycle per Pixel metric (CPP), which is the total number of clock cycles divided by the number of pixels, running the convolution on 256x256 gray image. The results clearly demonstrate that our SIMD system achieves a speed up equal to 4.8 compared to the sequential execution performed by the ACU. This speed up is more than 2 times greater than when using customized SIMD instructions with the NiosII processor.

Table 9 compares between our system composed of miniMIPS reduced PEs and the FPGA-based SIMD processor described in [24] composed of one control unit and a number of computation units (CU) in terms of power consumption and frequency. As illustrated, our system achieves good results since it presents a higher frequency and a lower power consumption compared to the SIMD processor. The proposal processor reduction methodology is easier to design the SIMD architecture with a parametric number of PEs compared to the used dedicated implementation [24] that makes adding

more computational units to the SIMD processor a heavy task.

The preceding performance comparison confirms the promising advantages of the proposed FPGA-based SIMD architecture. Its performance has been evaluated for image convolution benchmark with good results that validate the proposed high performance architectural model. In this paper, we have shown that an FPGA-based SIMD design built with soft-core processors is powerful to compute image processing. One of the main advantages of the proposed architecture is its configurability. The architecture is scalable and flexible enough to support several applications.

Considering multimedia applications that comprise not only SIMD operations but also operations suitable for exploiting task parallelism too, we can benefit from the proposed system. In fact, in vision algorithms for example, some tasks are performed on the objects found to analyse their quality or properties in order to make decisions on the image contents. It appears that SIMD type of architectures is not very efficient for this treatment. A DSP is often more appropriate. So, we can replace the ACU by a DSP. Our proposed system can also assure task parallelism by parallelizing the program between the ACU and one PE. The proposed many-core SIMD system can also work as an accelerator for another system (which can be a unique processor or an MPSoC) or as a coprocessor of the main CPU reducing its workload and accelerating the data-parallel expensive computation. In this case, it executes the data-parallel parts of the program, and the ACU can process any task of the program (while the PEs are working). This manner may introduce some control and communication overhead, which can be ameliorated via tailored components. As example, to reduce data access latency, we can integrate a

DMA controller between the memories of the PEs and the main memory.

Some work have been proposed to exploit SIMD architectures for computer vision algorithms like the SIFT (Scale-invariant feature transform) algorithm. In [40], authors propose a novel parallel approach for SIFT algorithm implementation using a block filtering technique in a Gaussian convolution process on the SIMD Pixel Processor. They demonstrate that their system can perform real-time processing with good performance. In [41], authors propose a hybrid SIMD/MIMD architecture for image processing. The architecture is composed of a signal-processing CPU and an image coprocessor. This latter supports two working modes SIMD and MIMD. In [42] authors demonstrate that the interconnection network used in the SIMD architecture influences the performance of computer vision applications. Considering a set of basic vision tasks namely convolution, histogramming, hough transform, extreme point identification, etc. authors show the performance and the advantages of the polymorphic torus communication when executing these tasks. In [43], a parallelization of SIFT using multi-core architecture with per-core SIMD support is shown. Authors present good results compared to the State-of-the-Art parallel SIFT algorithms.

So, we conclude from the aforementioned implementations that we can use a pure SIMD system while optimizing/modifying the algorithm to take benefit from the HW architecture, or tailor the architecture to the requirements of the algorithm.

8. Conclusion

This work presents an SoC platform to facilitate generating SIMD massively parallel on-chip architectures. It proposes an IP-based implementation to build these architectures leading to the design of large and complex architectures with lower costs and higher performance. Different SIMD configurations can be derived from the generic SIMD SoC model. The proposed massively parallel system is characterized by its flexibility allowing matching the design with the application as well as improving its performances and satisfying its requirements. This flexibility allows a designer to choose the area/performance of a parallel system without laborious hardware design, and can better meet application requirements. The proposed RTL implementation makes any hardware modification easy in order to adapt the system to a wide range of data dependent algorithms. All these features strongly contribute to the increase of the designer's productivity. Through prototyping results, the user is able to choose the appropriate SIMD SoC configuration satisfying his needs and meeting performance requirements.

This work opens an interesting topic for future research and development on parallel applications. Future work will be to provide a high-level design framework. An automatic exploration level can help the designer to generate the most appropriate and efficient massively parallel configuration for a given application. Another future work is to study the dynamic runtime use of reduction and replication proposed methodologies.

[1] A. Gentile, and D. Scott Wills, *Portable Video Supercomputing*, IEEE Trans. Comput. Volume 53, Issue 8, Aug. 2004.

[2] A. Gentile, and D. Scott Wills, *Impact of Pixel per Processor Ratio on Embedded SIMD Architectures*, Proceedings of the 11th International Conference on Image Analysis and Processing (ICIAP), 2001.

[3] A. A. Abbo, and R. P. Kleihorst, *A programmable smart camera architecture*, Proceedings of the Conference of Advanced Concepts for Intelligent Vision Systems, Sept 2002.

[4] A. Lokhmotov, *Programming and compiling for embedded SIMD architectures*, University of Cambridge, 2008. [Online]. Available: http://www.doc.ic.ac.uk/ãnton/misc/thesis.pdf

[5] W.J. Bouknight, S.A. Denenberg, D.E. Mclntyre, J.M. Randall, A.H. Sameh and D.L. Slotnick, *The illiac IV system*, Pro. IEEE, Volume 60, Issue 4, pp. 369−388, April 1972.

[6] T.J. Fountain, K.N. Matthews and M.J.B. Duff, *The CLIP7A image processor*, IEEE Trans. Pattern Anal. Mach. Intell, Volume 10, Issue 3, pp. 310−319, May 1988.

[7] W. Kim and R. Tuck, *MasPar MP-2 PE chip: A totally cool hot chip*, Proceedings of Hot Chips V, March 1993.

[8] L.W. Tucker and G.G. Robertson, *Architecture and Applications of the Connection Machine*, IEEE Comput, Volume 21, Issue 8, pp. 26−38, August 1988.

[9] J. A. Rowson and A. Sangiovanni-Vincentelli, *Interface-Based Design*, Proceedings of the Design Automation Conference, pp. 178−183, 1997.

[10] N. Yamashita, T. Kimura, Y. Fujita and al., *A 3.84 GIPS integrated memory array processor with 64 processing elements and a 2-MB SRAM*, IEEE J. Solid-State Circuits, Volume 29, Issue 11, pp. 1336−1342, 1994.

[11] H. Nguyen and L. John, *Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology*, Proceedings of the International Supercomputer Conference, pp. 11−20, June 1999.

[12] A. Peleg, S. Wilkie and U. Weiser, *Intel MMX for multimedia PCs*, Communications of the ACM, Volume 40, Issue 1, pp. 25−38, Jan 1997.

[13] C. Torres-Huitzil and M. Arias-Estrada, *Real-time image processing with a compact FPGA-based systolic architecture*, Real-Time Imaging, Elsevier, pp. 177−187, 2004.

[14] John E. Dorband, *The Massively Parallel Processor: Architecture and Application*, Journal of Forth Application and Research, Volume 5, Issue 1, pp. 27−38, 1987.

[15] D. Dale, L. Grate, E. Rice and R. Hughey, *The UCSC kestrel general purpose parallel processor*, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 1243−1249, Las Vegas Nevada, 1999.

[16] P. Yiannacouras, J. Gregory Steffan, and J. Rose, *Fine-Grain Performance Scaling of Soft Vector Processors*, Proceedings of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems, Grenoble, France, 2009.

[17] Christopher H. Chou, A. Severance and Alex D. Brant, *VEGAS: Soft Vector Processor with Scratchpad Memory*, Proceedings of the ACM/IEEE International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 15−24, Monterey, California, USA, 2011.

[18] A. Severance and G. Lemieux, *VENICE: A Compact Vector Processor for FPGA Applications*, Proceedings of the Workshop on the Intersections of Computer Architecture and Reconfigurable Logic, 2012.

[19] K. Sano, Y. Hatsuda and S. Yamamoto, *Scalable Streaming-Array of Simple Soft-Processors for Stencil Computations with Constant Memory-Bandwidth*, Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines, pp. 234−241, 2011.

[20] Z. Ping Ang, A. Kumar and Y. Ha, *High Speed Video Processing Using Fine-Grained Processing on FPGA Platform*, Proceedings of the IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines, pp. 85−88, 2013.

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

[22] nVIDIA, *GeForce Graphics Cards*, 2014. [Online]. Available: http://www.nvidia.com/object/geforce_family.html

[23] Altera, *Nios II C2H Compiler User Guide*, 2009. [Online]. Available: http://www.altera.com/literature/ug/ug_nios2_c2h_compiler.pdf

[24] S. Tanabe, T. Nagashima and Y. Yamaguchi, *A study of an FPGA based flexible SIMD processor*, ACM SIGARCH Computer Architecture News, Volume 39, Issue 4, pp. 86−89, Sep. 2011.

[25] M. Sayed, W. Badawy and G. Jullien, *Towards an H.264/AVC HW/SW Integrated Solution: An Efficient VBSME Architecture*, IEEE Transactions on Circuits and SystemsII, Volume 55, Issue 9, Sep. 2008.

[26] R. Lopez Rosas, A. de Luca and F. Barbosa Santillan, *SIMD Architecture for Image Segmentation using Sobel Operators Implemented in FPGA Technology*, Proceedings of the 2nd International Conference on Electrical and Electronics Engineering (ICEEE) and XI Conference on Electrical Engineering, 2005.

[27] F. Schurz and D. Fey, *A Programmable Parallel Processor Architecture in FPGA for Image Processing Sensors*, Proceedings of the Integrated Design and Process Technology, 2007.

[28] T. Blank, *The MasPar MP-1 architecture*, Proceedings of the IEEE Compcon Spring, pp. 20−24, 1990.

[29] B. Dammak, M. Baklouti and M. Abid, *Soft-core reduction methodology for SIMD architecture: OPENRISC case study*, Proceedings of the 5th IEEE International Design and Test workshop, Abu Dhabi United Arab Emirates, 2010.

[30] M. Baklouti, Ph. Marquet, JL. Dekeyser and M. Abid, *Reconfigurable Communication Networks in a Parametric SIMD Parallel System on*

Chip, Proceedings of the 6th International Applied Reconfigurable Computing Symposium ARC, 2010.

[31] M. Baklouti, Y. Aydi, Ph. Marquet, JL. Dekeyser and M. Abid, *Scalable mpNoC for massively parallel systems Design and implementation on FPGA*, Journal of Systems Architecture, Volume 56, Issue 7, pp. 278−292, 2010.

[32] M. Baklouti, Ph. Marquet, JL. Dekeyser and M. Abid, *A design and an implementation of a parallel based SIMD architecture for SoC on FPGA*, Proceedings of the Conference on Design and Architectures for Signal and Image Processing DASIP, 2008.

[33] M. Dulong and T. M. Bernard, *Recherche d'une exploitation nergtique optimale des ressources de calcul dans un systme de vision sur puce*, GRETSI symposium, 2009.

[34] W. Raab, J. Berthold, U. Hachmann, H. Eisenreich and J-U. Schluessler, *Low Power Design of the X-GOLD SDR 20 Baseband Processor*, Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 792−793, 2010.

[35] Opencores, *minimips overview*, 2007. [Online]. Available: http://www.opencores.org/?do=project&who=minimips

[36] Opencores, *OR1200 OpenRISC processor*, 2010. [Online]. Available: http://opencores.org/openrisc,or1200

[37] Terasic, *Altera DE2-70 Board*, 2010. [Online]. Available: http://www.terasic.com

[38] C. Paul, B. Mike, B. Brad and W. Paul, *Multi-mode sensor processing on a dynamically reconfigurable Massively Parallel Processor Array*, Proceedings of SPIE, the International Society for Optical Engineering, 2008.

[39] D. Curd, *Power Consumption in 65 nm FPGAs*, White Paper: Virtex-5 FPGAs, WP246, Feb 2007.

[40] Le T. Su , J-S. Lee. *Novel Parallel Approach for SIFT Algorithm Implementation*. Journal of information and communication convergence engineering. Dec, Volume 11, Issue 4, pp. 298−306. 2013

[41] A. Nieto, D.L. Vilarino, V.M. Brea. *SIMD/MIMD dynamically-reconfigurable architecture for high-performance embedded vision systems*. 23^{rd} IEEE International Conference on Application-Specific Systems, Architectures and Processors. pp. 94−101. 2012.

[42] M. Maresca, H. Li, M. M. C. Sheng. *Parallel Computer Vision on Polymorphic Torus Architecture*. Machine Vision and Applications, Volume 1989, Issue 2, pp. 215−230. 1989.

[43] F. Wu, Q. Wu, Y. Tan, X. Sun. *Speeding up SIFT algorithm by multi-core processor supporting SIMD instruction sets*. 13^{th} International Conference on Computer-Aided Design and Computer Graphics. pp. 451−452. 2013.