# Run-time Scalable Systolic Coprocessors for Flexible Multimedia SoPCs

Andrés Otero, Eduardo de la Torre, and Teresa Riesgo
Centro de Electrónica Industrial
Universidad Politécnica de Madrid
Spain
{joseandres.otero, eduardo.delatorre, teresa.riesgo}@upm.es

Yana E.Krasteva
Parallel Architectures Group
Universidad Politécnica de Valencia
Spain
yana.krasteva@gap.upv.es

*Abstract*—**Multimedia Systems on Chip have high computational requirements, as well as significant flexibility demands. Flexibility can be related with the reusability of the cores in charge of the execution of computation-intensive tasks, but also with the run-time adaptation of these cores to the execution of time-variable tasks, or to changing system conditions. Among the run-time flexibility requirements of hardware IPs, functional scalability has been identified as an interesting feature. The proposal in this paper is to take advantage of the regularity and the high-processing capability of systolic arrays, to develop run-time functional scalable cores, making use of spatial scalability, by means of replicating and relocating basic processing elements of the array. The relocation process is performed using the dynamic-reconfiguration possibilities offered by commercial FPGAs. In this paper, an architectural template is proposed to develop systolic scalable coprocessors following this approach, together with its corresponding software drivers that may be executed within an embedded processor. In addition, a design flow is proposed to adapt the architectural template to different problems, together with some examples of scalable cores created following this design. This solution provides better results, regarding the reconfiguration time and the memory necessity overhead, compared with other dynamically scalable solutions.**

***Keywords-component; Digital signal processing, adaptable cores, scalability, systolic array, partial runtime reconfiguration***

## I. INTRODUCTION

The intensive innovation efforts in multimedia and communications fields lead to the appearance of new applications and standards at a very high rate [1] [2]. In addition, development time of new products has to be shortened, due to the high market pressure.

A way to reduce design cycle time and cost of new generations of multimedia systems is to divide the processing burden into independent modules. These blocks, in the form of pre-designed software or hardware intellectual properties (IPs), would be reused in different applications, avoiding the design of new systems from scratch. The IP hardware cores act like coprocessors, speeding-up the execution of computation-intensive tasks. By joining one or multiple general purpose processors, some memory and a communication infrastructure, coprocessors can be integrated on a single chip, leading to the System on a Chip (SoC) methodology [3]. To increase the core reusability, some design-time approaches, like the customization of the HDL description with generic attributes, has been widely used in the state of the art [4] [5].

However, some applications require run-time adaptation. This means that some of the computational tasks of the system may depend on varying conditions imposed by the application or by the user. In addition, system conditions, like changes in the amount of free hardware resources in constrained multitasking devices, or the available energy in portable terminals, also demand run-time flexibility capabilities to the processing cores. These run-time requirements cannot be fulfilled with traditional design-time customizations, and other solutions have to be explored. In the state of the art, clock gating of some fine-grain core elements [6] and the configuration by means of registers [4], are possible alternatives. In this paper, a more advanced solution based on the dynamic and partial reconfiguration feature offered by some FPGAs will be explored. This technique consists in the run-time modification of the of the core functionality, while the rest of the system continue working. Furthermore, due to the advances in programmable silicon solutions, full partially run-time reconfigurable Systems can be implemented on a single System on Programmable Chip (SoPC).

Among run-time flexibility requirements of hardware IPs, functional scalability has been identified as an interesting feature. This means, to online modify the size of the operation performed by the core. Some examples in the state of the art of functional scalable cores are the Discrete Wavelet Transform (DWT) presented in [7] , the variable-size Discrete Cosine Transform (DCT) in [8] or the motion estimation and filters in [9]. This paper addresses a solution where the functional scalability of a hardware block is achieved by means of spatially scaling the physical implementation of a core. That is, modifying the area occupied by the core inside the reconfigurable system. A direct approach to create variable-size scaling cores would be to implement the same task in several cores, with different performance and area requirements, and load the most suitable one in the system depending on the available hardware resources. Differently, in this work, highly parallel, modular and regular architectures have been studied as a scalable core architecture alternative to reduce the overhead of the adapting process. These architectures can be scaled, in advance of the execution of a task, by means of the addition and removal of parallel blocks,

resulting in lower adaptation times. This approach also allows the execution of tasks with different tradeoff points in the area/time design space.

Among the suitable architectures to implement scalable solutions based on dynamic and partial reconfiguration, systolic arrays [7] and distributed arithmetic [10] are the most commonly used in the state of the art. Distributed arithmetic provides scalable solutions to perform arithmetic operations, while systolic architectures can solve full computing-intensive tasks in a broad range of fields.

In this paper, an architectural template to develop run-time scalable cores based on systolic arrays on commercial FPGAs is provided. Differently from the existing solutions, this proposal, instead of being focused on offering solutions to a specific problem [9] [10] [12], is a generic template that can be tuned to solve different computational problems. The adaptation is carried out using a custom design-flow described in a previous work [13]. This work includes both hardware aspects, as well as software drivers to create and manage the cores. To prove the claimed generality of the proposed template, and the validity of the design flow, some examples of scalable cores for multimedia systems are provided. As will be seen, the use of this approach provides better results, regarding the reconfiguration time and the memory necessity overhead, compared with other dynamically reconfigurable solutions.

The rest of the paper is organized as follows. In section 2, the system architecture is overviewed. Next, the scalable core template is described, including the hardware details in section 3, and software aspects in 4. Section 5 provides a summary of the design flow. Some implementation use cases and results are shown in section 6, and finally, the conclusions can be found in section 7.

## II. SYSTEM ARCHITECTURE OVERVIEW

Dynamically programmable devices allow the development of SoCs with self-reconfiguration capability. In the case of Xilinx FPGAs, the logical configuration of the system can be autonomously changed by means of the Internal Configuration Access Port (ICAP), together with a hardware control system. The implementation follows an approach similar to [14].

SoPC architectures typically divide the FPGA into two regions: the static side, containing the embedded processor, the ICAP control and reconfiguration management, as well as the input/outputs with the external multimedia data links; and the reconfigurable side, where the different peripherals will be configured. In the reconfigurable area, some non-scalable peripherals can be included, as well as the dynamically scalable ones. The system architecture can be seen in Figure 1.

Bistreams corresponding to the reconfigurable cores are stored in an external memory, in this case, a CompactFlash. When a new core is required, the reconfiguration manager has to read the configuration data from the external memory and load the corresponding bitstream into the internal device configuration SRAM. Also the basic processing blocks of scalable peripherals will be stored in this external memory, and the configuration manager will obtain them from this memory, in case the required one has not already been configured in any position of the reconfigurable fabric.

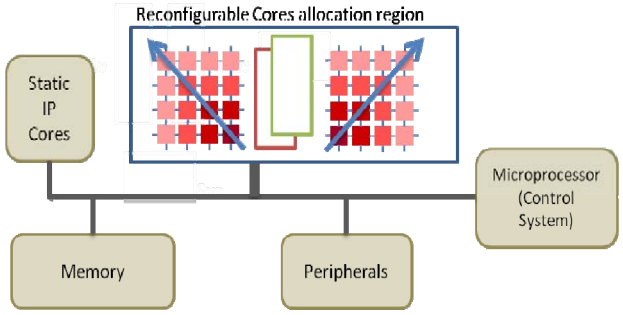


**Figure 1.** System general view

The scalable core architecture presented in this paper has been integrated into de Embedded Design Kit, commercially available from Xilinx. In this framework, cores use an IPIF interface to gain access to the CoreConnect bus technology from IBM [15]. This allows the core to be managed through the 64-bits PLB bus, from either a PPC or a Microblaze. The inclusion of the scalable cores into EDK offers the possibility of integrating them into complex systems, including also other commercial or used designed peripherals, as well as software implemented tasks. The following section shows hardware related aspects in more detail.

Regarding the software component, a generated C program for a certain application could include instructions to generate a systolic coprocessor, in advance it is required to implement certain applications. For instance, before implementing a filtering, some instructions to create the filter and to adapt its response can be executed. Software related aspects will be seen in section IV.

## III. COPROCESSOR HARDWARE

Systolic arrays are regular networks of interconnected processing elements that process data streams in a rhythmic fashion [16]. Mainly because of its inherent parallelism, pipelining and data processing capabilities, systolic arrays have been adopted for implementing multiple architectures in very different computational-intensive fields, from linear algebra to multimedia applications. In [17], a set of examples of algorithms implemented in SA is reported. In addition, its high regularity, modularity and homogeneity, as well as the regularity of its interconnections, mainly limited to the closer neighbors, make them very suitable to implement scalable solutions based on dynamic reconfiguration, like those provided in [7] and [18].

The main components of the systolic arrays are the processing elements. These modules, arranged like a two dimensional matrix, are in charge of the computational tasks of the core. In addition, some blocks have to be created to arrange the data samples to be processed by the matrix, as well as to generate the necessary control signals. Also an input block to interconnect the array with the static design is necessary, as well as some communications blocks.

In Figure 2, the basic architectural template, with the different element types, is shown. Details about the elements are provided below.

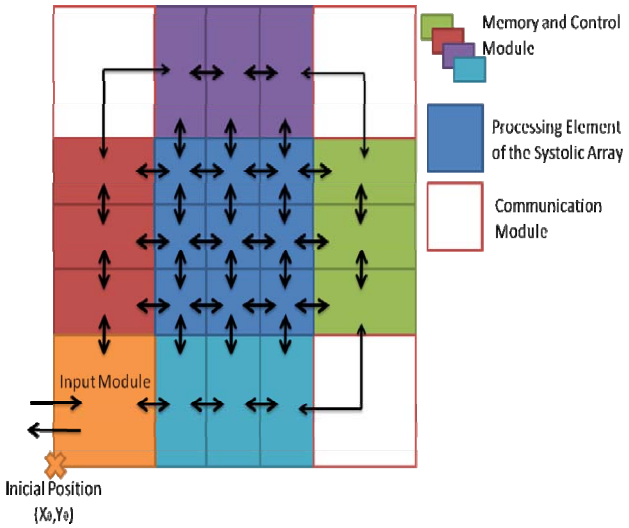


**Figure 2.** Overview of the Scalable Architecture, with the different kinds of elements.

### *A. Processing Matrix*

The basic processing capacity of the Systolic Coprocessor is performed by the Processing Elements Array. This is a two-dimensional structure formed by fine-grain elements, each one in charge of performing a single operation. This structure is inherently parallel and pipelined, and the scalability can be achieved by means of the addition of parallel basic processing elements. In the case of completely homogeneous systolic arrays, all these processing elements are equal; however, some designs are heterogeneous, including different processing elements from a reduced set. The initial approach is focused on rectangular arrays, but can be easily reduced to triangular or linear designs, depending on the final algorithm. Since the interconnections between elements are reduced to their closer neighbor, the cost and the complexity of defining compatible communications among them is reduced. Inter-element connections will be described in the design flow section.

### *B. Scalable Memories and Control*

These modules are responsible for performing data distribution to each processing element of the matrix, and to store the processed output results. In addition, they generate the necessary control signals. To reduce the reconfiguration overhead of the peripheral, both regarding the reconfiguration time, as well as the storage memory for the bitstreams, the control and the memory components of the peripheral cannot be centralized, since it would be dependent of the size of the core. The solution is to distribute also memory and control, that is, to slice it into modules associated to each reconfigurable level, which can be also dynamically added or removed, when the peripheral is scaled. This approach increases the reusability of the memory and control blocks, and also allows the reduction of the communications to the closer modules to a certain one.

### *C. Input Module*

Each dynamically scalable peripheral will have an entrance point to allow the interconnection with the static area of the design; this entrance point will receive the input control and data values from the embedded processor through accesses to the peripheral address space, and also will transmit the processed outputs to the static area. These registers will be connected with the PLB bus using the IPIF, following the approach in [19]. The input module will remain in the same position independently of the run-time scale level of the core, and this will be the location reference to calculate the run-time position of the processing modules

### *D. Communication Modules*

The purpose of the communication modules is to complete signal paths, or to communicate some modules which are not directly neighbors, like those situated in the corners of the array. Also reconfigurable blocks including bus-macro connections to run-time bypass heterogeneous columns of the FPGA can be included in this category, as it is done in [13].
As it has been already said, the proposal of this paper to scale the core, is to run-time vary the number of its parallel modules, including processing elements, memories and communication blocks. As a result, instead of generating a single bitstream corresponding to the configuration of the full core, a single smaller bitstream corresponding to each different module of the coprocessor is generated. This approach has some advantages. Since most of these processing elements are equal, the necessity of storing different configuration bits is highly reduced. In addition, during the reconfiguration process, most of the cores are already configured in the FPGA, so they can be directly replicated from the configuration memory itself, avoiding accesses to the external configuration memory. This is a PRR-PRR approach, as in [20]. A set of modules can be created, allowing the reusability of these fine grain elements. In addition, constant adaptations of each module can be performed at run-time, for instance, to adapt internal filter or transformation coefficients.

## IV. COPROCESSOR SOFTWARE

As has been already said, peripherals integrated in EDK include both a hardware description, as well as software drivers for operating them. These drivers are source code created in a high level language, typically C or C++, that implement functions that allow the easy integration of the peripherals with the system.

The proposed software driver for the scalable IPs is composed by two files: the Peripheral Definition File, and the Peripheral Operation File. The Peripheral Definition File describes the peripheral itself, including the size and position of each basic element, as well as the steps to scale it. The Peripheral Operation File is a set of functions to create, scale and adapt the cores, using the data structures included in the Peripheral Definition File. This file hides the dynamic reconfiguration details of the coprocessor, to the system user.

### A. *Peripheral Definition File*

This file contains the definition of the structure of the scalable peripheral, as Figure 3 shows. This includes the description of each basic block of the architecture, regarding its size and the name of the file containing the corresponding bitstream; In addition, it defines a matrix indicating which type of element is located in each position of the core, for each possible size. The definition file also contains the coordinates of the input block, which will be independent of the scalability level of the coprocessor.

```
// Peripheral Definition File
#define NumElem 10
#define SizeMin 3
#define SizeMax 9

typedef struct {

        char Elem [NumElem][2];
        char ElemName[NumElem][50];
        char initial;

} Scalable_Peripheral_Elements;


typedef struct {

        char Perif[SizeMax-SizeMin][SizeMax][SizeMax];

} Scalable_Peripheral_Definition;

typedef struct {

        char X0;
        char Y0;

} Scalable_Peripheral_Position;
```

**Figure 3.** Data structures included in the Peripheral Definition File

### B. *Peripheral Operation File*

This file describes some software functions to perform the basic core operations, to create it, and to manage the scale process. This eases the integration with the system, hiding the reconfiguration details to the user. In addition, it includes functions to control the configuration registers of the peripheral, as well as to control input and output data transferences. Prototype functions included in this file are:

- *Create_ Peripheral:* Includes instructions to configure the input block of the core, and to reconfigure the necessary elements to the smaller considered size.
- *Scale_Peripheral*: This function drives the reconfiguration process. Figure 4 and Figure 5 show its flow graph and a graphical example over a peripheral matrix, respectively. The first task is to search the position of the input module in the matrix corresponding to the new size. From this point, the matrix is divided into four quadrants. Since the input module will always be located in a frontier position of the peripheral (In Figure 5 it is in a central position to show the matrix division), at least, two of these quadrants will be empty, and any reconfiguration operation will be performed on them. For the others, the matrix is explored, comparing the new element of each position with the already configured ones. When a difference is found, the new element is reconfigured. The position of these elements is calculated on run-time, also using the information included in the peripheral definition file.
- *PerformOperation*: This file is dependent on the final function of the core, and generates the necessary instruction and commands, to transfer the data to the peripheral, and perform an operation.

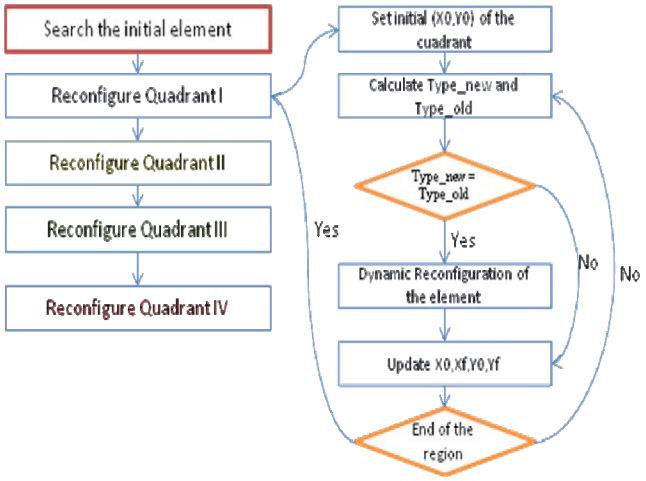


**Figure 4.** Flow graph of the peripheral scale function

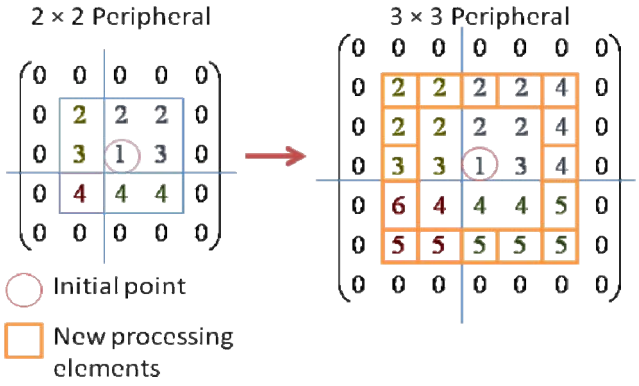


**Figure 5.** Example of the division of a peripheral matrix into quadrants. The initial point and the new elements to be reconfigured are highlighted.

These functions use a low level API to reconfigure basic elements, from a file name and a description of its coordinates, to indicate the origin and the destination in the reconfigurable fabric.

## V. SCALABLE COPROCESSOR DESIGN FLOW

The proposal of scalable coprocessor provided in this paper is more an architectural template, that means, a framework to develop scalable cores to be added to embedded systems, than an architecture itself. Consequently, it is necessary to provide

a design flow that allows the creation of new cores for specific purposes, by means of the adaptation of the template. Details of this design flow, as well as a numerical comparison with other proposals of the state-of-the-art, are provided in [13]. An important requirement to this design flow is the compatibility with the commercial Xilinx design flow. This would make the creation of scalable cores much easier for system designers.

The first step is to analyze the adaptation of the problem to be solved, to a systolic architecture. Afterwards, the scalable systolic architecture has to be described using a Hardware Description Language. At this step, the behavior of the full systolic array can be simulated and debugged. Regarding the HDL description, it is important to create a separate module for each different element in the architecture. In addition, regarding the elements interconnections, it is necessary to limit them to the closer neighbors of each module.

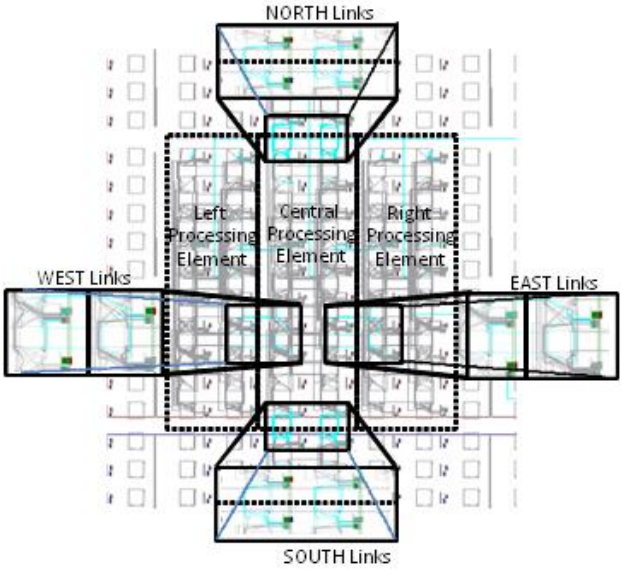


**Figure 6.** Symmetry of the north/south and east/west connections of the processing element.

Once the behavior of all elements is validated, it is necessary to synthesize each one separately. The previously created HDL is used to synthesize the elements, but in some cases, due to the impossibility of commercial tools to introduce constraints to the routing of the modules, it is necessary to perform the placement and routing process by hand, using FPGA Editor. Once the modules are created, two approaches can be followed, depending on the area optimization needs. The first one is to use the Modular Design flow proposed by Xilinx [21]. According to this flow, modules have to be connected through bus-macros that allow signals to cross reconfigurable module boundaries. These macros are instantiated in the top design, and in order to act as hard modules that do not change, they are constantly reloaded in the systems with partial reconfigurations. This approach has the advantage of requiring less design time, however it is not suitable to fine-grain processing elements, like those proposed in this systolic arrays, since bus-macro increases the area overhead of the architecture.

However, we propose another alternative, which does not require bus-macros. This is achieved by exploiting the symmetry in the communications between processing elements in the systolic array. To achieve this, the north and west connections of an element have been designed using the same FPGA routing resource positions than the south and east connections. This symmetry has been also exploited to the design of the array processing elements that use the same routing resources to transmit the same signals. As a result, when a new element is added to the array in the reconfigurable area, its south routing wires will match with the north wires of the element below, and their north wires with the south port of the element above. If the east-west connections keep the same conditions, this technique permits all elements to be wire-compatible without the use of bus macros and the communications between the processing elements is guaranteed during dynamic reconfiguration. An example of this design can be seen in Figure 6.

However, heterogeneous columns may be bypassed by using bus-macros that are compatible with the interconnections between elements. This technique, although does not make use of these resources, permits to extend the array to much larger sizes. The connection between the scalable elements and the static area is also achieved with bus macros.

## VI. Use Cases: A library of reconfigurable elements for Multimedia Signal Processing

To study the feasibility of the implementation of the systolic processor in real-world applications, some examples have been generated. In addition, since we claim that the proposed architectural template is general, this has been proved with the application of the proposed methodology to different designs. As we will see, these basic elements are integrated in a library, so they can be reused, not only to scale a single core, but also in different cores.

### *A. 2D Convolution Core for Image Filters*

The first use-case is a two-dimensional convolution operator, to implement window-based operations like image filters. Due to the regularity of this operation, it is suitable to be implemented in a systolic architecture, like the one reported in [18]. Figure 7 shows the mapping of the convolution kernel like a scalable architecture. The different blocks of the architecture are highlighted, including the processing elements, the input memories and control, as well as the output modules and the entrance point. Two versions of the core with different size have been included, to show how the scalability is achieved by means of the replication of some basic elements.

The implementation results of the different modules are shown in Table 1, together with the dimensions and the number of reconfiguration frames of each one. Compared with the non-scalable scenario, a high improvement in terms of the bitstream memory requirements is achieved, since, instead of having to store a full bitstream for each possible size of the core, only a bitstream corresponding to each basic element is required. In the case of the image filter, with the scalable proposal of this paper, 750 reconfiguration frames have to be stored. Considering a non-scalable filter with 3×3, 5×5, 7×7 and 9×9

possible sizes, a bitstream for the region occupied by the larger version of the core (considering a maximum area of 2470 CLBs and 8 FIFOs) 4920 frames has to be stored. The reduction of the bitstream size is about the 85%. Also, in terms of the scaling time, with the non-scalable scenario, the full core has to be reconfigured, while with the proposal of this paper, only the new elements has to be configured. In Table 2, the number of frames to be reconfigured in different scale operations and a comparison with a non-scalable solution is provided.

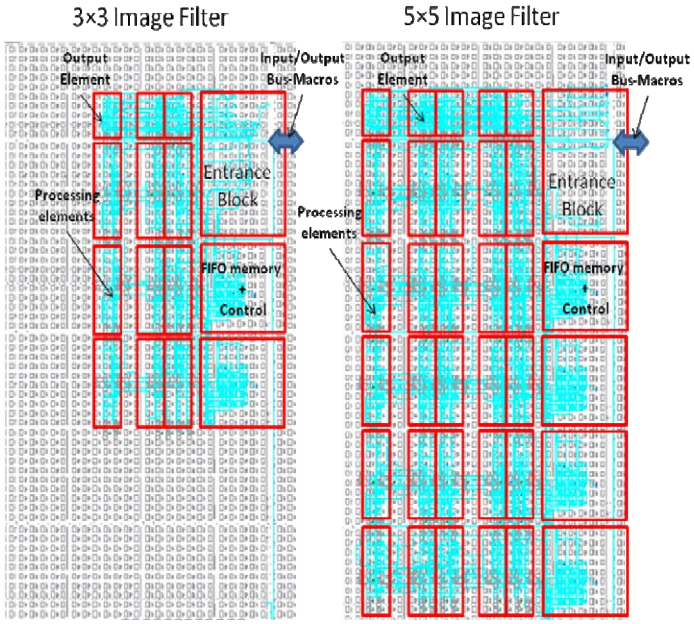


**Figure 7.** 3×3 and 5×5 versions of the Image Filter core. The basic blocks are highlighted

**Table 1.** Basic blocks of the convolution core with its resource occupation

| Basic Block | Size | Configuration Frames |
|---|---|---|
| Entrance Block | 8 × 15 CLBs | 288 |
| FIFO memory + Control | 8 × 10 CLBs + 1 Block RAM | 318 |
| Output Element | 2 × 10 CLBs | 72 |
| Processing Element | 2 × 5 CLBs | 72 |

**Table 2.** Number of frames to be reconfigured with the proposed scalable core, and comparison with the non-scalable scenario

| Scale Operation | Proposal | Comparison with the static core |
|---|---|---|
| Create Core (3 × 3) | 1788 | 36 % |
| 3 × 3 to 5 × 5 | 1932 | 39% |
| 3 × 3 to 5 × 5 | 2508 | 51% |
| 5 × 5 to 7 × 7 | 3084 | 62% |

By scaling this core, the size of the mask applied in the filtering process can be run-time varied, allowing different run-time trade-offs between the noise rejection and the area occupancy of the filter.

### *B. Scalable Matrix Multiplication*

Matrix multiplications are computational intensive functions that are extensively used in a broad range of applications, like image processing or algebraic calculus. Figure 8 shows the implemented architecture, and Figure 9 provides the basic definition matrix and the other software data structures stored to manage the core. Two different versions of the Processing element and the Input Memory and control have been created, with the same functionalities, but adapted to different positions on the FPGA.

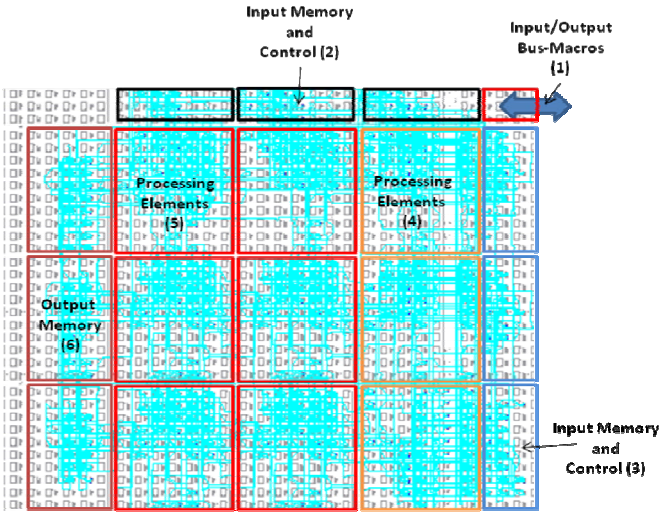


**Figure 8.** 3×3 Matrix multiplication core.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 2 & 1 & 0 \\ 0 & 6 & 5 & 5 & 4 & 3 & 0 \\ 0 & 6 & 5 & 5 & 4 & 3 & 0 \\ 0 & 6 & 5 & 5 & 4 & 3 & 0 \\ 0 & 6 & 5 & 5 & 5 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

**Figure 9.** Peripheral definition matrix corresponding to the matrix multiplication core

The quantitative details of the Scalable Matrix Multiplication are similar to the case of the image filter reported in the previous section. Matrix multiplication can be seen like and example of run-time computational trade-offs, since, for instance, an N × N matrix multiplication can be implemented on an N × N co-processor in a single processing round, or in 8 processing rounds, using a (N/2) × (N/2) accelerator.

### C. Scalable DCT

Discrete Cosine Transformation is used in image and video compression, like MPEG4 or JPEG. The implementation provided in this work is a systolic architecture following the approach reported in [21]. The output memories of the Scalable Matrix multiplication, as well as the entrance element can be reused.

This is an example of the functionally scaling of the core, depending on the required dimensions in each application, but also, in compression applications, can be used to set quality tradeoff between the size of the core, and its area occupancy.

## VII. Conclusions and Future Work

This paper describes the architecture of a generic core based on a systolic array with spatial scalability. The scaling method is based on replication and relocation of basic elements using dynamic partial reconfiguration. This technique allows generating a processing array with run-time adaptable size, to match with device available area or with varying functional requirements. Resources released in the device can be freely used to load other cores. The scaling process is driven by a set of software functions included in the core driver. To allow the scaling process, a design flow that generates a communication structure that does not require the use of bus-macros is proposed, resulting in important area savings and improved FPGA occupation. In addition, with this approach, improvements are also achieved for both the reconfiguration time overhead and the amount of configuration data. An image filter, a matrix multiplication and a DCT core have been developed as use cases.

Current work is being carried out to implement new scalable cores in order to cover the processing needs of a scalable video decoder SVC. On the other side, the control of the reconfiguration is being addressed, in order to automatically solve trade-offs between the size of the cores, and certain metrics, like quality or security measurements. Automatic area scheduling and management will also be addressed in the future.

## VIII. Acknowledgements

This work was supported by the Spanish Ministry of Science and Research under the project DR.SIMON (Dynamic Reconfigurability for Scalability in Multimedia Oriented Networks) with number TEC2008-06486-C02-01.

## References

[1] Haavisto, P.; Castagno, R.; Honko, H.; , "Multimedia standardization for 3G systems," *5th International Conference on Signal Processing Proceedings, 2000. WCCC-ICSP 2000.*, vol.1, no., pp.32-39 vol.1, 2000

[2] Tseng, P.; Chang, Y.; Huang, Y.; Fang, H.; Huang, C.; Chen, L.; , "Advances in Hardware Architectures for Image and Video Coding - A Survey," *Proceedings of the IEEE* , vol.93, no.1, pp.184-197, Jan. 2005

[3] Gupta, R.K.; Zorian, Y.; , "Introducing core-based system design," *Design & Test of Computers, IEEE* , vol.14, no.4, pp.15-25, 1997

[4] Zhao Junchao; Chen Weiliang; Wei Shaojun; , "Parameterized IP core design," *4th International Conference on ASIC, 2001. Proceedings.*, vol., no., pp.744-747, 2001

[5] Nordin, G.; Milder, P.A.; Hoe, J.C.; Puschel, M.; , "Automatic generation of customized discrete Fourier transform IPs," *Proceedings. 42nd Design Automation Conference, 2005.*, vol., no., pp. 471- 474, 13-17 June 2005

[6] Jian Huang; Jooheung Lee; Yimin Ge; , "An array-based scalable architecture for DCT computations in video coding," *2008 International Conference on Neural Networks and Signal Processing,* , vol., no., pp.451-455, 7-11 June 2008

[7] Syed, S.B.; Bayoumi, M.A., "A scalable architecture for discrete wavelet transform," *In the Proceedings of the Conference on Computer Architectures for Machine Perception, 1995.* CAMP '95

[8] Jian Huang; Jooheung Lee; Yimin Ge, "An array-based scalable architecture for DCT computations in video coding," *In the Proceedings of the International Conference on Neural Networks and Signal Processing*, 2008, vol., no., pp.451-455, 7-11 June 2008

[9] Meher, P.K.; Chandrasekaran, S.; Amira, A., "FPGA Realization of FIR Filters by Efficient and Flexible Systolization Using Distributed Arithmetic," *In IEEE Transactions on Signal Processing,* vol.56, no.7, pp.3009-3017, July 2008

[10] Mehendale, M.; Sherlekar, S.D.; Venkatesh, G., "Area-delay tradeoff in distributed arithmetic based implementation of FIR filters," *In the Proceedings of the Tenth International Conference on VLSI Design*, 1997, pages.124-129, Jan 1997

[11] Gause, J.; Cheung, P.Y.K.; Luk, W., "Reconfigurable shape-adaptive template matching architectures," *In the Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002. vol., no., pp. 98-107, 2002

[12] J. Huang, M. Parris, J. Lee, and R. F. DeMara, “Scalable FPGA Architecture for DCT Computation using Dynamic Partial Reconfiguration”, *In the Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA),* Las Vegas, U.S.A., July 2008.

[13] Otero, A. ;Krasteva, Y.; De la Torre, E.; Riesgo, T.; ,”Generic Systolic Array for Run-Time Scalable Cores”, *In the Proceedings of the 6th International Symposium on Applied Reconfigurable Computing (ARC),* Bangkok, Thailand, March 2010

[14] Corbetta, S.; Morandi, M.; Novati, M.; Santambrogio, M.D.; Sciuto, D.; Spoletini, P.; , "Internal and External Bitstream Relocation for Partial Dynamic Reconfiguration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.17, no.11, pp.1650-1654, Nov. 2009

[15] Murgida, M.; Panella, A.; Rana, V.; Santambrogio, M.D.; Sciuto, D.; , "Fast IP-Core Generation in a Partial Dynamic Reconfiguration Workflow," *Very Large Scale Integration, 2006 IFIP International Conference on* , vol., no., pp.74-79, 16-18 Oct. 2006

[16] Kung, H.T.; , "Why Systolic Architectures?," *Computer* , vol.15, no.1, pp. 37- 46, Jan 1982

[17] Tselepis, I.N.; Bekakos, M.P.; , "VHDL Code Automatic Generator for Systolic Arrays," *Information and Communication Technologies, 2006. ICTTA '06. 2nd* , vol.2, no., pp.2330-2334, 0-0 0

[18] Saldana, G.; Arias-Estrada, M., "FPGA-based customizable systolic architecture for image processing applications," *In the Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, 2005. ReConFig 2005., vol., no., pp.8 pp.-3

[19] Canto, E.; Fons, F.; Lopez, M.; , "Reconfigurable OPB Coprocessors for a Microblaze Self-Reconfigurable SOC Mapped on Spartan-3 FPGAs," *- 32nd Annual Conference on IEEE Industrial Electronics, IECON 2006*, vol., no., pp.4940-4944, 6-10 Nov. 2006

[20] Sudarsanam, A.; Kallam, R.; Dasu, A.; , "PRR-PRR Dynamic Relocation," *Computer Architecture Letters* , vol.8, no.2, pp.44-47, Feb. 2009

[21] Xilinx Corp., "XAPP 290: Two flows for Partial Reconfiguration: Module Based or Difference Based," www.xilinx.com, Sept 2004.

[22] Cariccia, F.; Cariccia, P.; Martina, M.; Molino, A.; Vacca, F.; , "Multimedia SoC: a systolic core for embedded DCT evaluation," *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers, 2002.*