Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 97929, 13 pages
doi:10.1155/2007/97929

Research Article

A Predictive NoC Architecture for Vision Systems Dedicated to Image Analysis

Virginie Fresse, Alain Aubert, and Nathalie Bochard

Laboratoire de Traitement du Signal et Instrumentation, CNRS-UMR 5516, Université Jean Monnet Saint-Étienne, Bâtiment F, 18 Rue Benoit Lauras, 42000 Saint Étienne Cedex 2, France

Received 1 May 2006; Revised 16 October 2006; Accepted 26 December 2006

Recommended by Dietmar Dietrich

The aim of this paper is to describe an adaptive and predictive FPGA embedded architecture for vision systems dedicated to image analysis. A large panel of image analysis algorithms with some common characteristics must be mapped onto this architecture. Major characteristics of such algorithms are extracted to define the architecture. This architecture must easily adapt its structure to algorithm modifications. According to required modifications, few parts must be either changed or adapted. An NoC approach is used to break the hardware resources down as stand-alone blocks and to improve predictability and reuse aspects. Moreover, this architecture is designed using a globally asynchronous locally synchronous approach so that each local part can be optimized separately to run at its best frequency. Timing and resource prediction models are presented. With these models, the designer defines and evaluates the appropriate structure before the implementation process. The implementation of a particle image velocimetry algorithm illustrates this adaptation. Experimental results and predicted results are close enough to validate our prediction models for PIV algorithms.

1. INTRODUCTION

More and more vision systems dedicated to a large panel of applications (tracking, fault detection, etc.) are being designed. Such systems allow computers to understand images and to take appropriate actions, often under hard real-time constraints and sometimes under harsh environments. Moreover, current algorithms are computing resource-intensive. Traditional PC or DSP-based systems are most of time unsuitable for such hard real-time vision systems. They cannot achieve the required high performance, and dedicated embedded architectures must be designed. To date, FPGAs are increasingly used because they can achieve high-speed performances in a small footprint. Modern FPGAs integrate many different heterogeneous resources on one single chip and the number of resources is incredibly high so that one FPGA can handle all processing operations. Data coming from the sensor or any acquisition device is directly processed by the FPGA; no other external resources are necessary. These systems on chip (SoCs) become more and more popular as they give an efficient quality of results (QoR: area and time) of the implemented system. FPGA-based SoCs are suitable for vision systems but their designs are complex and time-consuming as hardware specialists are required. It is crucial that designed architectures are adaptive to dynamic or future algorithms to increase the design productivity. Adding new parts or replacing some blocks to the previous design may be required. FPGAs are reconfigurable, which ensures architecture adaptations by functional block modifications [1]. From an FPGA synthesis point of view, the reuse aspect is as important as the QoR. New SoC architectures and design methods break global problems down into local ones and rely on networks on chip (NoCs) to compose local solution [2, 3]. With NoC, it is possible to design the blocks independently as stand-alone blocks and create the NoC by connecting the blocks as elements in the network.

A regular topology NoC has much higher throughput and a better scalability compared to on-chip buses. For large SoCs with multiple IPs (intellectual property), bus architectures often fail to deliver required throughput and need large chip areas. Regular topology NoC was proposed as on-chip communication architectures primarily using switching and routing techniques [2, 4]. To date, topologies use more sophisticated techniques as related to literature [5]. Regular topology NoC is inspired by general-purpose multicomputer networks. A two-dimensional (2D) folded torus NoC is proposed by Dally and Towles in [4]. Two-dimensional mesh NoC, such as CLICHÉ, Nostrum, Eclipse, and aSoC, is respectively presented by Millberg et al. in [6], by Forsell in [7], and by Liang et al. in [8]. RAW is a multiprocessor system based on a 2D mesh NoC [9]. SoCIN uses a 2D mesh or torus [10]. Spin has a fat-tree topology [11]. Octagon has a fixed topology [12]. Proteo uses a ring topology [13]. 2D mesh topology is preferred in most studies, because of its simplicity and corresponding tile-based floorplan. However, most NoCs are application/domain-specific and existing NoCs are not specific to image processing domain. Our objective is to apply the NoC concept to design an adaptive architecture for image processing algorithms.

An important NoC design task is to choose the most suitable NoC topology for a particular application and mapping of the application onto that topology. There are many image processing algorithms, and their identifying characteristics may be quite different. One topology is not suitable for all image processing algorithms. Nevertheless, one topology can be suitable for some algorithms with similar characteristics. Image processing algorithms must be classified according to some identified characteristics (input and output data flow, global/local operations, etc.). For each category, an adaptive NoC topology is presented. The overall project provides a library of NoC topologies, of architecture models and of interchangeable IP blocks. This provides a fast and efficient implementation of any image processing algorithm on FPGA. This paper addresses the first category of image algorithms, which concerns image analysis applications. This application consists of extracting some relevant parameters from a large flow of data.

Understanding major characteristics of image analysis algorithms leads to design an adaptive and predictive embedded architecture using an NoC approach. The designer predicts the suitable structure according to the algorithm using associated timing and resource prediction models.

The paper is organized into 6 further sections. In Section 2, the main characteristics of image analysis applications are used to define an adaptive NoC structure. Each module is also explained as well as the communication protocol. Section 3 presents architecture analysis. Its characterization and modelling are presented in Section 4. The PIV application is applied to the architecture in Section 5 in order to illustrate the principle of models definitions (timing and resource models). Results are analyzed and interpreted in Section 6. Section 7 contains the conclusion.

2. THE ADAPTIVE NoC ARCHITECTURE FOR IMAGE ANALYSIS ALGORITHM

The aim of this section is to describe an adaptive architecture for vision systems dedicated to image analysis algorithms. This architecture is designed so that the linear effort property presented in [3] is guaranteed. The effort of modifying some parts only depends on these parts but not on the rest of the architecture. The adaptivity of the architecture must therefore be taken into account during the design process. The first task consists of identifying the major characteristics of image analysis algorithms. These characteristics are then used to define an appropriate architectural model.

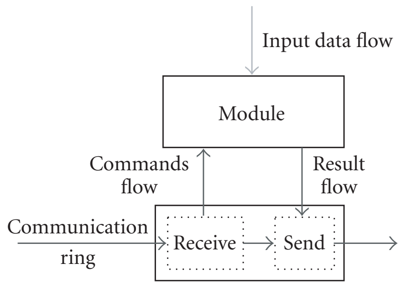

Figure 1: Model of communication flows.

2.1. *Architecture description*

Image analysis consists of extracting some relevant parameters from one or several images. Image analysis examples are object segmentation, feature extraction, image motion and tracking, and so forth [14, 15]. Any image analysis requires four types of operations:

(i) acquisition operations for image capture;
(ii) storage operations;
(iii) processing operations;
(iv) control operations for the system supervision.

In an NoC concept, the global algorithm is divided into local blocks to compose local solutions [16–18]. The local blocks are called modules. Each module handles one type of operation and all modules are connected as elements in a network-based topology. Several modules may be required for one type of operation. As an example, processing operations can be implemented on several processing modules to improve speed.

A characteristic of image analysis applications is unbalanced data flow between input and output. The input data flow corresponds to a high number of pixels (images), whereas the output data flow represents little data information (selective results). From these unbalanced flows, two different communication topologies must be defined, each one being adapted to the speed and flow of data.

For the input data flow, a parallel bus ensures high bandwidth. For the result and command flows, a new flexible communication topology needs to be identified. A dedicated bus is not suitable due to the scalability constraint. This communication topology must have an interface with an unlimited number of modules. A shared unidirectional bus is designed from the “token-ring” approach. This communication flows are shown in Figure 1.

This new communication model is a bit like the Harvard model, which has physically separate storage and signal pathways for instructions and data. Our model is based on these separated flows: the input data flow is separated from the command flow. The reduced output data flow (the result flow) is mixed with command flow.

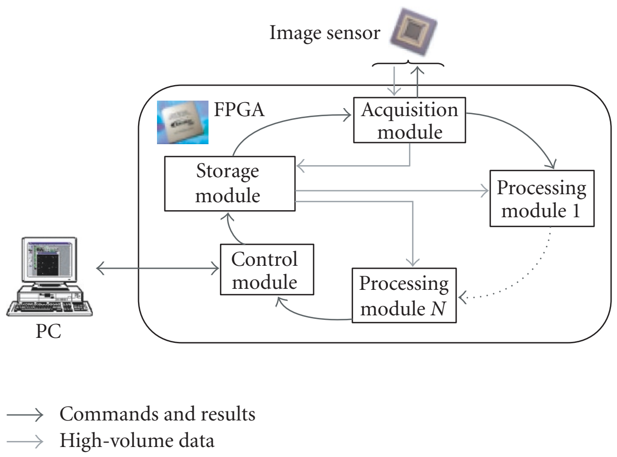

Figure 2: The proposed NoC architecture for vision systems dedicated to image analysis algorithms.

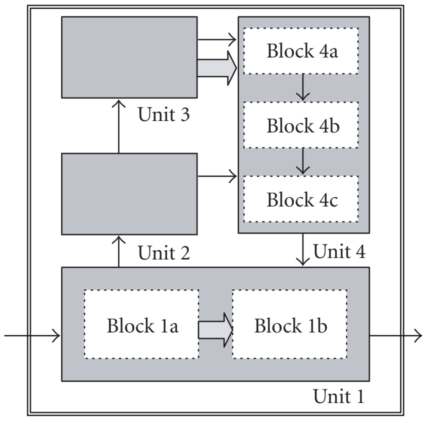

Figure 3: Module structure.

Using the modular principle and the communication ring, multiple clock domains can be defined. Some operations must run with the maximal clock frequency, other frequencies are technically constrained. For example, the acquisition operation depends on the acquisition device. With a single clock domain, increasing the speed of a part of the design leads to modification of the technically constrained parts. Using a globally asynchronous locally synchronous (GALS) approach, logic that constitutes one module is synchronous and each module runs at its own frequency. Each module can therefore be optimized separately to run at its best frequency. Communications between modules are asynchronous and they use a single-rail data path 4-phase handshake designed in the wrapper. This GALS structure allows many optimizations and an easier evolution [19–22].

All modules are inserted around the ring as shown in Figure 2. The number of modules is theoretically unlimited.

2.2. Modules description

The modular principle of the architecture can be shown at different levels: one type of operation is implemented by means of a *module* (acquisition, storage, processing, etc.). Each module includes *units* that carry out a function (decoding, control, correlation, data interface, etc.), and these units are shaped into basic *blocks* (memory, comparator, etc.). Some units can be found inside different modules. Figure 3 shows all levels inside a module.

The number and type of modules depend on the application. As image analysis algorithms require several types of operations, this architecture accepts several types of modules.

(i) The *acquisition module* produces data that are processed by the system. A prototype of this architecture is built around a CMOS image sensor to have a real SoC. But if a particular hardware characteristic is necessary, the modularity of the system ensures easily replacing the CMOS sensor by any sensor, camera, or other source of data. This module produces all CMOS image sensor commands and receives CMOS image sensor data. One part takes the 10-bit pixel data from the sensor and sends them to the storage module. A simple preprocessing can be performed in this module such as a binarization operation.

(ii) The *storage module* stores incoming images from the acquisition module. Writing and reading cycles are supervised by the control module. Whenever possible, memory banks are FPGA-embedded memories. This embedded memory is a shared dual-port memory. The acquisition module writes data into memory and this memory is shared between all processing modules for reading operation. Two buses are used for parallel memory accesses. Recent FPGAs can store more than one image having 1280×1024 8-bit pixels, and more than 5 images having 512×512 pixels, and this value will increasingly grow up in the future. If more memory space is needed, an external memory device can be used and the storage module carries out the interface between the external device and the system.

(iii) The *processing module* contains the logic that is required for a given algorithm. The result from the image analysis is then sent to the control module by means of the communication ring. To improve performance, more than one processing module can be used for the parallel operations. If several processing modules are used, the operations are distributed on all processing modules.

The number of these modules is theoretically unlimited. The control of the system is not distributed in all modules, but it is fully centralized in the single control module, which performs decisions and scheduling operations.

(i) The *control module* sends commands to each module through the communication ring. These commands activate predefined macrofunctions in the target module. The integrity and the acceptance of the commands are checked with a flag inserted in the same command frame that returns to the control module. As all commands are sent from this module, the scheduling must be specified in the control

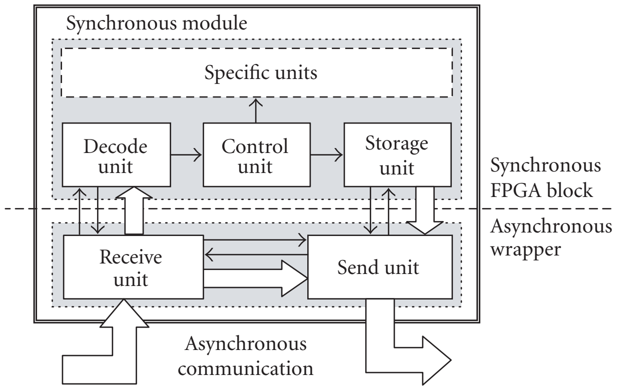

FIGURE 4: Asynchronous wrapper structure.

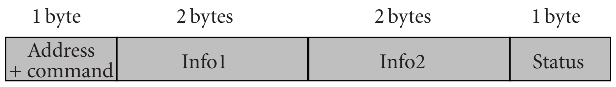

FIGURE 5: Communication frame structure.

module. In the same way, this module receives resulting data transferred from a processing module through other modules. Results are sent to the PC by a standard USB link.

2.3. Communication protocol

Each module is designed in a synchronous way having its own frequency. Communications between modules are asynchronous via a wrapper and they use a single-rail data path 4-phase handshake. Two serial flipflops are used between independent clock domains to reduce the metastability [23, 24]. The wrapper includes two independent units (see Figure 4). One receives frames from the previous module and the other one sends frames to the following module at the same time.

Through the communication ring, the control module sends frames containing command frames and empty frames. Empty frames can be used by any module to send results or any information back to the control module. So the output-reduced data flow (the result flow) is mixed with command flow in the communication ring. Each frame consists of 6 bytes (see Figure 5). The first byte contains the target module address and the command. In regular frame, different pieces of information associated to the command word are in the next four bytes. The last byte indicates the status information (error, busy, received, and executed). The target destination module sets this status according to its current state. Then the instruction is sent through the communication ring and is received by the control module. The state flag is analyzed. If an error occurs, the control module sends the command again to the target module through the communication ring, or can reinitialize the whole system if necessary.

TABLE 1: Static and dynamic modules in the adaptive FPGA-based system.

	Control	Processing	Acquisition	Storage
Parameters	Dynamic	Dynamic	Static	Dynamic
Scheduling	Dynamic	Static	Static	Static
Algorithm	Dynamic	Dynamic	Static	Dynamic
External device	Static	Static	Dynamic	Dynamic

3. ARCHITECTURE ANALYSIS

This architecture must adapt its structure to algorithm modifications. Four types of modifications are identified.

(i) *Parameters adaptation*: from a given image analysis algorithm, some parameters can vary. These parameters can be the size of full-analyzed images, the shape or the location of studied windows, and so forth.
(ii) *Parallel operations* (scheduling): for a given algorithm, the number of processing modules can vary to improve the parallelism. So the scheduling performed by the control module changes.
(iii) *Algorithm*: processing module can accept any algorithm meeting the targeted characteristics described in Section 2.1 (unbalanced data flow and parallelism).
(iv) *External devices*: any device can be replaced by another one. Acquisition devices such as cameras, CCD sensors, or other devices are interchangeable. Features of the new device may differ from the previous one and format of data as well. For each new acquisition device, the acquisition module must be adapted.

According to the type of modifications, only some modules must be changed. Modifying one module inside the architecture does not affect other modules, as modules are independent. Modules that depend on one or several modifications must be analyzed. All modules are numbered and classified into two categories.

(i) Modules that remain unchanged are *static* modules. Functional blocks are immediately reused without any modification.
(ii) Modules that are algorithm-dependent or architecture-dependent are *dynamic* modules. A dynamic module contains static and dynamic units/blocks. In this case, only the dynamic units/blocks must be changed.

A first analysis consists of identifying the static and dynamic modules in this architecture. The type of modification will determine specific static and dynamic modules given in Table 1. The reusability of this FPGA-based system is based on the percentage between static and dynamic parts. All dynamic unit/blocks have a fixed interface to avoid modifications of static blocks linked to them.

Dynamic modules can be either *predictive* or *nonpredictive*. Predictive modules are modules with resources and execution times estimated before the implementation process.

The acquisition module is camera-dependent but not algorithm-dependent. Image acquisition depends on the size

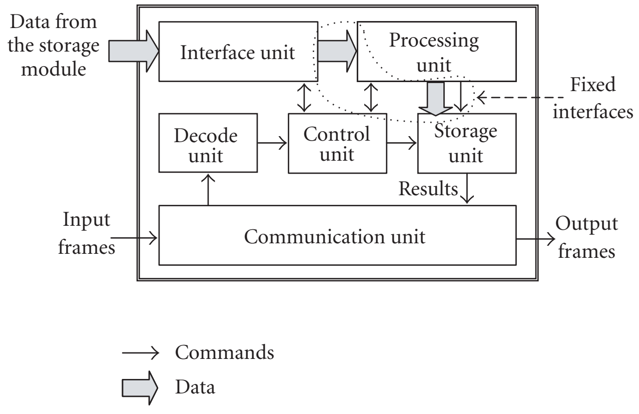

Figure 6: The processing module structure.

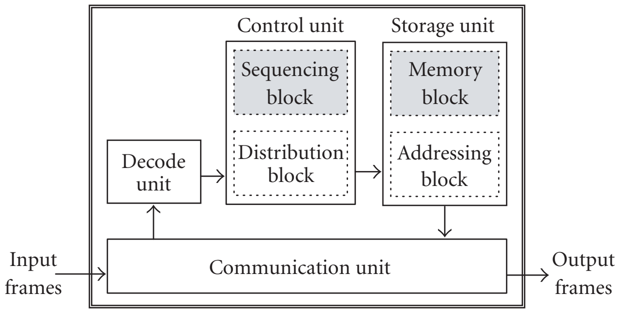

Figure 7: Control module structure.

of the grabber and the camera frequency. This module is dynamic when a new sensor/camera is used, but it is static if no replacement occurs. In an architectural point of view, changing an external device does not give any relevant information concerning our adaptive architecture. This type of modification is not described in this paper.

For other types of modifications, the storage module should be adapted to the size and the number of stored images. This module remains static for a constant size of the input image. If not, its only dynamic resource is the number of memory bits. The number of logic cells and registers would remain similar. So its evolution is very easy to predict and will not be described in this paper. The storage module can be considered as a static module.

For the dynamic modules such as the processing and the control modules, a more detailed analysis is required.

3.1. Processing module

The processing module is algorithm-dependent and parameter-dependent. This module is static if the type and the size of operations are identical; if not, it is dynamic. The processing module contains several units as shown in Figure 6. White units correspond to the static units and the grey ones to the dynamic units. Most units are static and the dynamic part corresponds to the processing itself. This unit is connected to static units by means of fixed interfaces.

For each type of modification,

(i) some algorithms are *parameterizable*. When the parameters vary, this module is *dynamic* and *predictive*. The operations remain identical, only the number and size of operations change. Therefore, it is possible to predict the number of resources for a new version from the previous implementation;

(ii) when the number of processing modules increases, the *scheduling* must be changed, but the operation inside a processing module remains identical. The processing module is then *static*;

(iii) for a *new algorithm* implementation, the processing unit is an *unpredictive dynamic* block. Resources depend on the new implemented operation. The HDL description and its implementation are necessary to find out the number of resources and the processing time. A hardware specialist is required in the design flow for a complete HDL description. A high-level development tool (DK Design Suite, ImpulseC, etc.) can be integrated in the design flow for the dynamic parts design. These tools estimate the needed resource and time, and the automatic generation of the HDL IP blocks can avoid the intervention of the hardware specialist. A first analysis of this integration has been made. Results are not optimized but remain satisfactory for most applications. The design flow for such adaptive architecture is not presented in this paper.

3.2. Control module

The control module is algorithm-dependent and scheduling-dependent. This module is therefore fully dynamic for the three types of modifications. Similar to the processing module, white units correspond to the static units and the grey ones to the dynamic units in the control module structure shown in Figure 7. The interfaces of dynamic units are fixed.

Two blocks are dynamic inside the control module. The memory block contains the command frames to send to all modules. The sequencing block dispatches operations on all modules. These blocks are predictive if the number of commands and the number of processing modules are known.

In the next section, the architecture is characterized for each type of modification. Resource and timing prediction models are presented.

4. ARCHITECTURE CHARACTERIZATION AND MODELING

The quality of result (QoR) indicates the required area and the execution time for the implemented algorithm. This QoR ensures the evaluation of the adaptive architecture and helps the designer to choose the suitable structure. Some prediction models must be provided to the designer. These models are more precisely a timing prediction model and a resource prediction model.

4.1. Resource prediction model

Global resources can be predicted by summing the resources of all modules: acquisition (AM), storage (SM), control (CM), and processing (PM). Several acquisition devices can be used: two cameras for stereovision and sometimes more than 2 for specific applications. The number of acquisition modules (N_{AM}) increases according to the number of acquisition devices. In the same way, storage modules can be multiplied (N_{SM}) to ensure concurrent memory accesses, and several processing modules (N_{PM}) can be inserted around the ring to get a better execution time. The control is centralized into one control module whatever the application is. Resources for the wrapper are included in the resources for each module. FGPA integrates several communication links that will be used for the communication ring. The resource prediction model is given in

$$R_{\mathrm{global}} = N_{\mathrm{AM}} \times R_{\mathrm{AM}} + N_{\mathrm{SM}} \times R_{\mathrm{SM}} + R_{\mathrm{CM}} + N_{\mathrm{PM}} \times R_{\mathrm{PM}}, \tag{1}$$

where R can be replaced by Lc, Rg, or Mb, respectively for the number of logic cells, registers, and memory bits.

According to the type of modification, static and dynamic resources inside the architecture change so that the resource prediction models differ. Therefore, models are presented for both types of modifications.

The first type of modification is *parameters adaptation* for one processing module ($N_{\mathrm{PM}} = 1$). When a parameter changes, only the content of some frames is modified but not the number. In a resource point of view, the control module is considered as a static module in this case. The processing module contains several units as shown in Section 3. The static units are interface unit (IU), decode unit (DU), control unit (CtU), storage unit (SU), and communication unit (CU) (wrapper). Only the processing unit (PU) is a dynamic unit that depends on the implemented algorithm. The resources for other modules are known, as these modules are static modules. In the following equation, bold parameters correspond to dynamic parts

$$\begin{aligned} R_{\mathrm{global}} &= N_{\mathrm{AM}} \times R_{\mathrm{AM}} + N_{\mathrm{SM}} \times R_{\mathrm{SM}} + R_{\mathrm{CM}} + \boldsymbol{R}_{\mathrm{PM}} \quad \text{with} \\ R_{\mathrm{PM}} &= R_{\mathrm{IU}} + R_{\mathrm{DU}} + R_{\mathrm{CtU}} + R_{\mathrm{SU}} + R_{\mathrm{CU}} + \boldsymbol{R}_{\mathrm{PU}}. \end{aligned} \tag{2}$$

For some cases, the resources for this dynamic unit (R_{PU}) can be estimated from a previous implementation. In other cases, the traditional way is the HDL description and resource estimation by means of dedicated CAD tools. The design flow for this adaptive architecture can integrate the DK Design Suite tool for the dynamic block description and resource estimation.

For the *scheduling modification*, all modules are static modules except the control module. The scheduling depends on the number (N_{PM}) of processing modules. The number of processing modules can vary but the structure of the processing modules remains unchanged. Dynamic and static parts for the control module are extracted from the previous analysis presented in Section 3. Two units are static units, the communication unit (CU) that corresponds to the wrapper and the decode unit (DU). Two other units are dynamic as they contain static and dynamic blocks. The control unit has a dynamic sequencing block (SB) and a static distribution block (DB). The sequencing block (SB) supervises each processing module, these resources depend on the number of processing modules (N_{PM}). The storage unit contains a static addressing block (AB) and a dynamic memory block (MB). The memory block stores all used frames for the algorithm. The resources correspond to the number of resources for one frame (R_F) multiplied by the number of stored frames (N_F). In this case, (1) becomes

$$\begin{aligned} R_{\mathrm{global}} &= N_{\mathrm{AM}} \times R_{\mathrm{AM}} + N_{\mathrm{SM}} \times R_{\mathrm{SM}} + \boldsymbol{R}_{\mathrm{CM}} + \boldsymbol{N}_{\mathrm{PM}} \times R_{\mathrm{PM}} \quad \text{with} \\ R_{\mathrm{CM}} &= R_{\mathrm{DU}} + R_{\mathrm{CU}} + R_{\mathrm{DB}} + R_{\mathrm{AB}} + \boldsymbol{N}_{\mathrm{PM}} \times R_{\mathrm{SB}} + \boldsymbol{N}_F \times R_F. \end{aligned} \tag{3}$$

For a *new algorithm*, (2) and (3) must be taken into account.

4.2. Timing prediction model

The global time to process one full image depends on three operations:

(i) the communication across the ring;
(ii) the memory access;
(iii) the processing itself.

Three parameters are associated with these three operations:

(i) the global communication time T_{com} depends on the number of frames to send (N_{SF}) and to a lesser degree on the number of modules around the ring;
(ii) T_{mem} is the sum of all data transfer from storage modules to processing modules through the 32-bit dispatching bus;
(iii) the processing time T_{proc} fully depends on the algorithm and on the number of processing modules.

According to the algorithm and to the configuration of the architecture (number of each type of module), some operations can be performed simultaneously. Thus the global time to process one full image (T_{global}) can be limited by a value:

$$T_{\mathrm{global}} \leq T_{\mathrm{com}} + T_{\mathrm{mem}} + T_{\mathrm{proc}}. \tag{4}$$

It is difficult to define a general model, as a lot of configurations for one algorithm exist. The timing prediction model is considered for a specific algorithm.

5. AN EXAMPLE OF IMAGE ANALYSIS ALGORITHM MAPPING ONTO THE ARCHITECTURE

As an example, a particle image velocimetry (PIV) algorithm is mapped onto this adaptive architecture [25].

5.1. The PIV algorithm

PIV is a technique for flow visualization and measurement [26, 27]. Particles are used as markers for motion visualization in the studied flow. In our application, two single exposure image frames are recorded by one CMOS sensor within

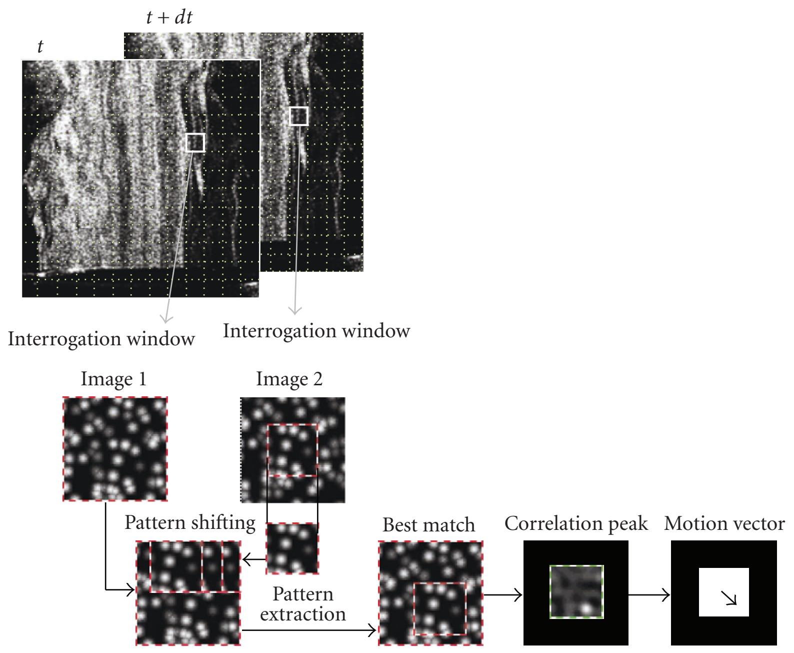

FIGURE 8: Principle of PIV algorithm.

a short time interval Δt. Recorded images are divided into 32×32-pixel small subregions called interrogation windows. From the interrogation window of the second image, a pattern (subregion) is extracted. This pattern is shifted in the corresponding interrogation window in the first image and both are cross-correlated. The diagram of this principle is presented in Figure 8. A traditional technique using grey-level images is adapted to binary direct cross-correlation to ensure an easier implementation in programmable logic devices. Multiplications usually used in grey-level representations are replaced by XNOR logical operations according to

$$F(i, j) = \sum_x \sum_y s_1(x, y)\,\text{XNOR}\,s_2(x - i, y - j), \quad (5)$$

where s_1 and s_2, respectively, represent the pixel values of the interrogation windows from images 1 and 2.

A PIV algorithm is suitable for parallel processing as the direct cross-correlation computation is highly parallelizable. Two cross-correlated interrogation areas are independent of each other. The same operation is computed simultaneously on different interrogation windows. This complex computation is therefore a good candidate for a hardware real-time implementation and well adapted to our architecture. Input data flow corresponds to 2 full binary images and the output data flow corresponds to coordinates of resulting vectors for each 32×32-pixel subwindows.

Some command frames must be sent to each module to adapt this algorithm to our architecture. The sequencing of this frames sent from the control module to other modules is shown in Figure 9.

5.2. *Prediction models for PIV algorithm*

Two types of modifications are studied for PIV algorithm.

(i) *Parameters modification*: from an algorithmic point of view, several parameters depend on the experimental environment for PIV applications. The size of images, camera frequency, and other information are tailored for a given environment as they depend on the characteristics (size and speed) of the fluid. As a consequence, it is sometimes important to change some parameters such as the size of the interrogation window. Traditional interrogation windows for PIV applications are 16×16, 32×32, or 64×64 pixels with or without overlapping windows. For different sizes of interrogation window, the number of resulting vectors varies and the size of correlation operations as well. So the processing module is dynamic. Inside the control module, only the content of commands varies, but not its number. So the control module is considered to be static.

(ii) *Scheduling modification*: the architecture accepts a high number (theoretically unlimited) of processing modules. This number depends on the specified speed given by the application. It has been shown in [25] that for a specified PIV application, the image processing designer evaluates the number of processing modules according to the speed required for the PIV application (i.e., the number of vectors per second). The image processing designer duplicates an identical processing module around the communication ring. An immediate consequence is that the number of required resources increases for each added module. Such adaptations require several models used to find the most suitable structure without any implementation. The scheduling (specified in the control module) can be changed that makes the control module the only dynamic module, all other modules being static.

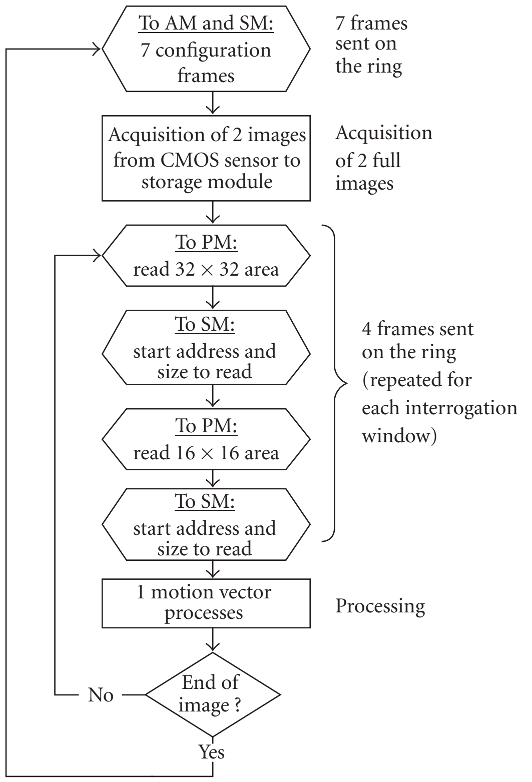
To AM and SM:
7 configuration frames
7 frames sent on the ring
Acquisition of 2 images from CMOS sensor to storage module
Acquisition of 2 full images
To PM:
read 32 × 32 area
To SM:
start address and size to read
4 frames sent on the ring (repeated for each interrogation window)
To PM:
read 16 × 16 area
To SM:
start address and size to read
1 motion vector processes
Processing
End of image ?
No
Yes

Figure 9: Command frames sequencing for PIV algorithm.

In order to find the required prediction models for these two types of modifications, a first FPGA implementation of our architecture with PIV algorithm is used to extract resources and some relevant timing. The architecture is implemented on an NIOS II board with a Stratix II 2S60 FPGA [28] and an IBIS4 CMOS image sensor with the following features:

(i) image size: 320×256 pixels. One pixel out of four is used to compose the images (viewfinder mode);
(ii) first interrogation window: 32×32 pixels;
(iii) pattern in second interrogation window: 16×16 pixels;
(iv) $N_{\mathrm{AM}} = N_{\mathrm{SM}} = N_{\mathrm{PM}} = 1$;
(v) frequencies: $F_{\mathrm{acquisition}} = 50\,\mathrm{MHz}$, $F_{\mathrm{storage}} = 100\,\mathrm{MHz}$, $F_{\mathrm{control}} = 150\,\mathrm{MHz}$, $F_{\mathrm{processing}} = 50\,\mathrm{MHz}$.

5.2.1. *Resource prediction model for PIV*

Table 2 gives the resources (logic cells, registers, and memory bits) obtained with the first FPGA implementation.

PIV resource prediction model for parameter modification

From Table 2 and (2), the resource prediction models for *parameters modification* can be defined if a model can be found for R_{PU} (resources for the processing unit). Three blocks constitute the processing unit:

(i) a memory block that stores 2 interrogation windows;
(ii) a comparison block that processes correlation and accumulation operations;
(iii) a supervision block.

The supervision block is a finite-state machine. The number of resources for this block remains identical, around 69 logic cells and 63 registers. The two other blocks depend on the interrogation windows: logic cells (Lc) and registers (Rg) resources are multiplied by 2 when the interrogation windows grow from $S \times S$ to $2S \times 2S$. For the memory bits (Mb), the storage block should contain an $S \times S$ window and its corresponding pattern whose size is $(S/2) \times (S/2)$.

The global resource parameter R is replaced by Lc, Rg, or Mb in the equations for resources concerning, respectively, logic cells, registers, and memory bits. Using the result from the first implementation, the resources for this processing unit can be estimated:

$$\begin{aligned} \mathrm{Lc}_{\mathrm{PU}} &= 69 + 186 \times \frac{S}{32}, \\ \mathrm{Rg}_{\mathrm{PU}} &= 63 + 217 \times \frac{S}{32}, \\ \mathrm{Mb}_{\mathrm{PU}} &= S^2 + \left(\frac{S}{2}\right)^2. \end{aligned} \tag{6}$$

The global resources for all 3 types of resources are given by

$$\boxed{\begin{aligned} \mathrm{Lc} &= 1038 + 186 \times \frac{S}{32}, \\ \mathrm{Rg} &= 1142 + 217 \times \frac{S}{32}, \\ \mathrm{Mb} &= 524320 + S^2 + \left(\frac{S}{2}\right)^2, \end{aligned}} \tag{7}$$

where S is the size of the interrogation window.

PIV resource prediction model for scheduling modification

From Table 2 and (3), resource prediction models for *scheduling modifications* can be defined if N_F and R_F are known. The control module stores the seven command frames used for the acquisition of the two full images. Then, four specific frames are stored in the control module to start the vector computation in a given processing module. So,

$$N_F = 7 + 4 \times N_{\mathrm{PM}}. \tag{8}$$

In the first implementation of our architecture, one processing module is used. As a consequence $N_F = 11$.

In Table 2, 24 logics cells are necessary to store 11 frames. So the number of logic cells to store one frame (R_F) is about 2 (for registers as well). Finally, the resource prediction model is

$$\boxed{\begin{aligned} \mathrm{Lc} &= 769 + 453 \times N_{\mathrm{PM}}, \\ \mathrm{Rg} &= 867 + 492 \times N_{\mathrm{PM}}, \\ \mathrm{Mb} &= 524320 + 1280 \times N_{\mathrm{PM}}, \end{aligned}} \tag{9}$$

where N_{PM} is the number of processing modules.

Table 2: Resource distribution associated with the first FPGA implementation.

Name			Logic cells			Registers (Rg)			Memory bits (Mb)		
Modules	Units	Blocks	Modules	Units	Blocks	Modules	Units	Blocks	Modules	Units	Blocks
Control (R_{CM})	Comm. (R_{CU})		278	30		265	39		32	0	
	Decod. (R_{DU})			44			42			32	
	Control	Distribution (R_{DB})		72	29		65	31		0	
		Sequencing (R_{SB})			43			34			
	Storage	Memory (R_{MB})		132	24		119	22		0	
		Addressing (R_{AB})			108			97			
Storage (R_{SM})			280			422			524288		
Acquisition (R_{AM})			264			225			0		
Processing (R_{PM})	Comm. (R_{CU})		402	33		447	34		1280	0	
	Decod. (R_{DU})			12			24			0	
	Control (R_{CtU})			49			42			0	
	Storage (R_{SU})			48			63			0	
	Interface (R_{IU})			5			4			0	
	Processing (R_{PU})	Storage		255	111		280	139		1280	1280
		Comparison			75			78			0
		Supervision			69			63			0

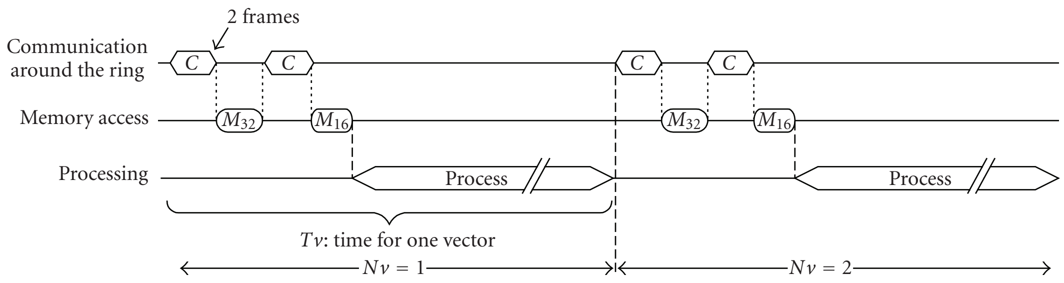

Figure 10: PIV timing diagram for $N_{PM} = 1$.

5.2.2. Timing prediction model for PIV

PIV timing prediction model for parameter modification

With one processing module for the architecture, the timing diagram corresponds to Figure 10.

In this case, the three operations defined in Section 4.2. are performed sequentially. So

$$T_{\text{global}} = T_{\text{com}} + T_{\text{mem}} + T_{\text{proc}} = Nv \times Tv, \tag{10}$$

where Nv is the number of vectors in one image and Tv is the time to process one vector.

The required time to process one vector is divided into 3 parts:

$$T_v = Tv_{\text{com}} + Tv_{\text{mem}} + Tv_{\text{proc}}, \tag{11}$$

where

(i) Tv_{com} is the communication time across the ring for one vector. This time corresponds to the number of sent frames for each vector (4 for PIV algorithm as detailed in the sequencing in Figure 9) multiplied by the time for one frame. Tv_{com} cannot be predicted before the implementation, and remains unchanged whatever the size of the interrogation window is;

(ii) Tv_{mem} is the time for the data transfer from the storage module to the processing module through the 32-bit dispatching bus. The data transfer concerns the $S \times S$ interrogation window and its $(S/2) \times (S/2)$ corresponding pattern:

$$Tv_{\text{mem}} = T_m(S) + T_m\left(\frac{S}{2}\right), \tag{12}$$

where $T_m(S)$ is the time to read the $S \times S$ interrogation window and $T_m(S/2)$ is the time to read the $(S/2) \times (S/2)$ pattern. When the interrogation windows grow from $S \times S$ to $2S \times 2S$, the data transfer time is multiplied by 4 if S is larger or equal to 32 bits. As the width of the bus is 32-bit long, the time for a data transfer is only divided by 2 when the interrogation windows decrease from $S \times S$ to $S/2 \times S/2$ if S is lower than 32 bits. This can be modelled by

$$\begin{aligned} T_m(S) &= T_m(32) \times \left(\frac{S}{32}\right)^2 \quad \text{if } S \geq 32, \\ T_m(S) &= T_m(32) \times \frac{S}{32} \quad \text{if } S \leq 32, \end{aligned} \tag{13}$$

where $T_m(32)$ is the time to read a 32×32-bit data block.

(i) Tv_{proc} is the processing time itself. No implementation is required to find this value. For a given position, the comparison between $S \times S$ interrogation windows and its corresponding pattern is processed during $S/2$ clock periods. This comparison is repeated for each possible position of pattern inside the interrogation window (i.e., $(S/2 + 1) \times (S/2 + 1)$ times). The deduced processing time is

$$Tv_{\text{proc}} = T_{\text{clk}} \times \frac{S}{2} \times \left(\frac{S}{2} + 1\right)^2, \tag{14}$$

where T_{clk} is the clock period.

All these times depend on the implementation target and the frequency of each module. These times are extracted from the first implementation, which is presented at the beginning of the section. For our application, the clock period of the processing module is 10 nanoseconds. The obtained results are

$$Tv_{\text{com}} = 5.6\,\mu\text{s}, \qquad T_m(32) = 4.9\,\mu\text{s}, \qquad T_m(16) = 2.5\,\mu\text{s}. \tag{15}$$

Adding other timing expressions gives the PIV timing prediction models:

$$\boxed{\begin{aligned} T_v(\mu\text{s}) &= 5.6 + 4.9 \times \frac{5}{4} \times \left(\frac{S}{32}\right)^2 \\ &\quad + T_{\text{clk}} \times \frac{S}{2} \times \left(\frac{S}{2} + 1\right)^2 \quad \text{if } S > 32, \\ T_v(\mu\text{s}) &= 5.6 + 4.9 \times \left(\frac{3S}{64}\right) \\ &\quad + T_{\text{clk}} \times \frac{S}{2} \times \left(\frac{S}{2} + 1\right)^2 \quad \text{if } S \leq 32. \end{aligned}} \tag{16}$$

PIV timing prediction model for scheduling modification

Without any information on the scheduling, only an upper bound can be defined using (4). For one vector, this equation becomes

$$T_v \leq Tv_{\text{com}} + Tv_{\text{mem}} + Tv_{\text{proc}}. \tag{17}$$

The processing operations (correlation operations) can be performed simultaneously inside each processing module. So this bound can be reduced by dividing the processing time by the number of processing module, as in

$$\begin{aligned} T_v &\leq \frac{N_{\text{PM}} \times Tv_{\text{com}} + N_{\text{PM}} \times Tv_{\text{mem}} + Tv_{\text{proc}}}{N_{\text{PM}}} \\ &\leq Tv_{\text{com}} + Tv_{\text{mem}} + Tv_{\text{proc}}. \end{aligned} \tag{18}$$

If the chosen scheduling is taken into account, a thinner estimation can be performed. For the PIV algorithm used in our architecture with multiple processing modules, the communication through the ring and memory accesses can be performed simultaneously. Frames for several modules are mixed (C_1, C_2, C_3, etc.) to take advantages of the two memory-buses, reducing significantly the latency. As an example, frames for processing modules 1 and 2 are alternated. In this way, reading time for the first interrogation window is finished when the second request begins. For this algorithm, mixing two processing modules gives a good result as shown in Figure 11. The time for the memory access is fully overlapped by other operations. (With longer memory access, mixing 3 or more modules could be better.)

The processing operations begin when the frames for the processing modules 1 and 2 are sent. Frames for other modules do not affect the global time. Only the time of the last sequence (the last 4 vectors) increases the global time, but it is ignored in our equations.

Therefore, the average processing time T_v for one vector can be approximated to

$$\boxed{\begin{aligned} T_v &= \frac{2 \times Tv_{\text{com}} + Tv_{\text{proc}}}{N_{\text{PM}}} \\ &< \frac{N_{\text{PM}} \times Tv_{\text{com}} + N_{\text{PM}} \times Tv_{\text{mem}} + Tv_{\text{proc}}}{N_{\text{PM}}}, \end{aligned}} \tag{19}$$

where N_{PM} is the number of processing modules.

This equation is used if $(N_{\text{PM}} - 2) \times Tv_{\text{com}} < Tv_{\text{proc}}$. The average time to process one vector does not decrease anymore if the global communication time becomes higher than the processing time.

For both types of modifications, a timing prediction model and a resource prediction model are extracted. The image processing designer can predict the execution time and the used resources according to modifications. These models are validated with an implementation. All results and comparisons are given in the following section.

6. ANALYSIS OF RESULTS AND INTERPRETATION FOR PIV

A PIV algorithm is mapped onto the architecture with several sizes of the interrogation window and different scheduling

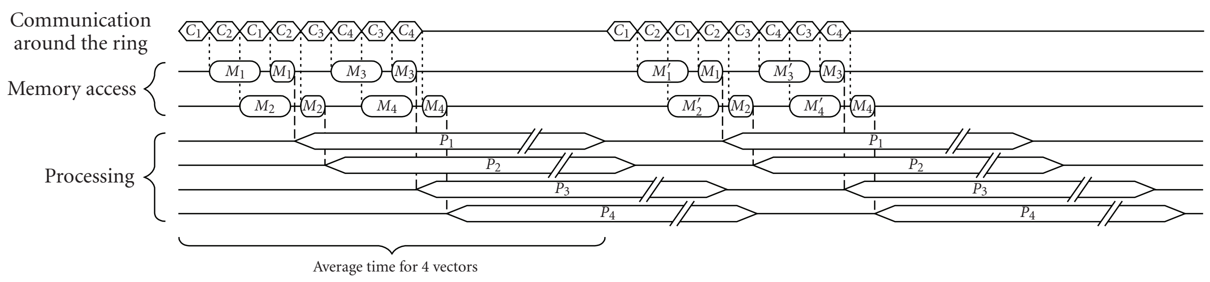

Figure 11: PIV timing diagram for $N_{\mathrm{PM}} = 4$.

Table 3: Resources distribution with several sizes of interrogation window.

	16 × 16 window		32 × 32 window	64 × 64 window	
	Result	Prediction	Result	Result	Prediction
Logic cells	1068	1131	1224	1375	1410
Registers	1211	1251	1359	1600	1576
Mem. bits	524608	524640	525600	529344	529440

Table 4: Required time to process one vector with several sizes of interrogation window.

	16 × 16 window		64 × 64 window	
	Result	Prediction	Result	Prediction
$Tv(\mu s)$	16.8	15.8	381	378.6

Table 5: Resources distribution when the number of processing modules increases.

N	Logic cells		Registers		Memory bits	
	Obtained	Predicted	Obtained	Predicted	Obtained	Predicted
2	1774	1681	1859	1851	526880	526880
4	2741	2595	2840	2835	529440	529440
6	3684	3509	3820	3819	532000	532000

and the architecture is implemented in a Stratix 2S60 FPGA. Results are given for 1 up to 6 processing modules inside the ring and for 16 × 16 and 64 × 64 interrogation windows. These experimental results will be compared with the prediction models. The obtained results are given for parameter modifications and then for scheduling modifications.

6.1. Efficiency of prediction models

6.1.1. Parameters modification results for PIV

Table 3 gives the results for 16 × 16, 32 × 32, and 64 × 64 interrogation windows with one processing module.

Table 4 gives timing estimations and results for the same parameters.

The models match with the experimental results as shown in Tables 3 and 4. These models are therefore accurate enough to estimate the resources and the timing of the chosen architecture without any implementation process.

6.1.2. Scheduling modification results for PIV

The results for the architecture up to 6 processing modules are presented with a 32 × 32 interrogation window. The predicted results are also added to enable comparisons. Resources and timing results are, respectively, presented in Tables 5 and 6.

The resource distribution model gives efficient results that are close to the implementation results. Timing models are a little bit below the experimental values. The most underestimated parameter is the communication around the ring. According to the algorithm, the sequencing is not really optimized and the control module sometimes waits for some results before sending a new frame (meanwhile, empty frames are inserted into the flow). Current work concerns the communication protocol to reduce this latency and to improve the efficiency of the communication.

In order to ensure comparisons with other embedded systems for PIV, the timing results are presented in different ways (clock frequency, etc.).

6.2. Efficiency of the architecture

Previous section presents the efficiency of resource and timing prediction models when the size of interrogation windows and the number of processing modules change. The intrinsic values of resources and processing time are not valued.

A specific and dedicated architecture gives better results than an adaptive architecture, as the VHDL description is optimized for this algorithm. However, our results are satisfactory with regard to the execution time and can be compared to the results provided by existing dedicated PIV systems. Our implemented architecture can handle high-speed constraints such as more than 80 000 vectors/s (for 32×32 interrogation windows) and can be adapted to high-speed applications.

Performing comparisons between several systems is a delicate task as each PIV system has its own characteristics, and could be more adapted to particular cases. Nevertheless, coarse comparisons between our architecture and other PIV systems are attempted. Some comparative results are summed up in Table 7.

Three systems are compared with our architecture. The first system is based on a "smart camera" as presented in [29].

Table 6: Timing results when the number of processing modules increases.

N	Average time to process 1 vector(μs)	Predicted time to process 1 vector(μs)	Reachable pixel clock frequency (MHz)	Number of vectors per second	Images 1280×1024/s.
2	31	28.7	33.0(×2)*	32258	25.2(×2)*
4	16.5	14.3	62.1(×2)*	60606	47.3(×2)*
6	11.8	9.6	87.1(×2)*	85106	66.5(×2)*

* The maximum frequency that the vision system can follow with 2 processing modules is 33 MHz if the motion vectors are processed between consecutive images (between 1 and 2, between 2 and 3, etc.). This frequency is multiplied by 2 if each image is only used in one couple (processes between 1 and 2, between 3 and 4, etc.).

Table 7: Some comparative results with other PIV systems.

	Smart camera [29]	GPUs [30]	XC2V6000 [31]	Our system [25]
Image size	1008×1016	1024×1024	1008×1008	1280×1024
Windows size	40×40	32×32	12×12	32×32
Algorithm	Discrete grey-level 12-bit cross-correlation	Discrete grey-level FFT correlation	Discrete grey-level 8-bit cross-correlation	Discrete binary cross-correlation
Overlap	50%	Yes, if multiple passes	No	No
Subpixel interpolation	3-point parabolic peak	Centre-of-mass, Gauss fit	No	No
Speed (fps)	6.25	7	40	66

This architecture can be adapted to different applications but it is restricted to a precise configuration for the PIV algorithm. The second PIV system uses a programmable graphics processing unit (GPU) [30] that makes the algorithm easier to modify, but that is the slowest. The last one [31] uses a Virtex II FPGA board for a very specific configuration of a PIV algorithm. The execution time is optimized for one precise PIV algorithm with fixed characteristics, but it is also the less evolutive architecture.

All these algorithms are grey-level scale image processing, whereas our architecture accepts only binary cross-correlation. Our results rely on the quality of the thresholding during the binarization process. Experimental results for our adaptive architecture can be only compared with other systems if a suitable binarization is used. In this case, our architecture gives a high QoR in terms of speed.

7. CONCLUSION AND FURTHER WORK

This paper presents an adaptive FPGA-based architecture for vision systems dedicated to image analysis algorithms. This architecture must adapt its structure to image analysis algorithms modification. Two approaches are used to ensure adaptivity. Using the NoC approach and major features of image analysis algorithms, this architecture has been designed by breaking down the global structure into standalone and dedicated modules. Two appropriate communication topologies are used: a communication ring and a bus for the incoming data. Using the GALS approach, all modules are inserted around the communication ring via an asynchronous wrapper. Thus each synchronous module can run at its own frequency.

This architecture contains modules that can be dynamic or static according to the type of modification. The image processing designer finds out the appropriate structure and changes the dynamic parts only. Timing and resource prediction models are given to ensure predictivity. A particle image velocimetry algorithm is mapped onto the architecture. Specific resource and timing prediction models are given for this algorithm from the first implementation. Results of several implementations (with some different parameters) are compared with these models. Resource and timing prediction models are close enough to the experimental results to help the image processing designer to choose the configuration (number of processing modules and interrogation window size) adapted to its constraints (camera frequency, size, and speed of the fluid) before the new implementation process.

Our architecture is compared with other systems. For this PIV algorithm, our architecture is well adapted to high-speed constraints. This paper shows that this architecture is well adapted to algorithm modifications. For a new image analysis algorithm, it is necessary to adapt prediction models. Using these adapted models, the image processing designer can precisely estimate the execution time and the used resources before the new implementation process. Consequently, the design flow is fast, reliable, and the implementation results are guaranteed.

Future work concerns the mapping of another image analysis algorithms in multispectral application domain. These new mappings will illustrate that the presented architecture can be adapted to other image analysis algorithms. These multispectral image processing algorithms contain several types of functions and the algorithmic complexity is interesting for the adaptive architecture in terms of communication protocol and scheduling. Another future work will focus on adaptive architectures dedicated to other classes of image processing algorithms. These architectures will be based on other NoC topologies. As a result, the image processing designer will use the topology library and adapts the structure according to the algorithm with the use of the timing and resource prediction models.

REFERENCES

[1] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-Chip Designs*, Kluwer Academic, Boston, Mass, USA, 1998.

[2] L. Benini and G. De Micheli, “Networks on chips: a new SoC paradigm,” *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[3] A. Jantsch and H. Tenhunen, *Networks on Chip*, Kluwer Academic, Boston, Mass, USA, 2003.

[4] W. J. Dally and B. Towles, “Route packets, not wires: on-chip interconnection networks,” in *Proceedings of the 38th Design Automation Conference (DAC ’01)*, pp. 684–689, Las Vegas, Nev, USA, June 2001.

[5] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, “A design methodology for application-specific Networks on Chip,” *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 263–280, 2006.

[6] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, “The Nostrum backbone-a communication protocol stack for networks on chip,” in *Proceedings of the 17th International Conference on VLSI Design (ICVD ’04)*, vol. 17, pp. 693–696, Mumbai, India, January 2004.

[7] M. Forsell, “A scalable high-performance computing solution for networks on chips,” *IEEE Micro*, vol. 22, no. 5, pp. 46–55, 2002.

[8] J. Liang, S. Swaminathan, and R. Tessier, “ASOC: a scalable, single-chip communications architecture,” in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT ’00)*, pp. 37–46, Philadelphia, Pa, USA, October 2000.

[9] M. B. Taylor, J. Kim, J. Miller, et al., “The raw microprocessor: a computational fabric for software circuits and general-purpose programs,” *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[10] C. A. Zeferino and A. A. Susin, “SoCIN: a parametric and scalable network-on-chip,” in *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI ’03)*, Sao Paulo, Brazil, September 2003.

[11] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, “SPIN: a scalable, packet switched, on-chip micro-network,” in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE ’03)*, Munich, Germany, March 2003.

[12] F. Karim, A. Nguyen, and S. Dey, “An interconnect architecture for networking systems on chips,” *IEEE Micro*, vol. 22, no. 5, pp. 36–45, 2002.

[13] D. Siguenza-Tortosa and J. Nurmi, “Proteo: a new approach to network-on-chip,” in *Proceedings of International Conference on Communication Systems and Networks (CSN ’02)*, Malaga, Spain, September 2002.

[14] M. Seul, L. O’Gorman, and M. J. Sammon, *Practical Algorithms for Image Analysis: Descriptions, Examples, and Code*, Press Syndicate of the University of Cambridge, Cambridge, UK, 2000.

[15] K. Sutherland and J. W. Ironside, “Novel application of image analysis to the detection of spongiform change,” *Analytical and Quantitative Cytology and Histology*, vol. 16, no. 6, pp. 430–434, 1994.

[16] G. Campobello, M. Castano, C. Ciofi, and D. Mangano, “GALS networks on chip: a new solution for asynchronous delay-insensitive links,” in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE ’06)* , pp. 160–165, Munich, Germany, March 2006.

[17] A. Hemani and A. Jantsch, “Network on chip: an architecture for billion transistor era,” in *Proceedings of the 18th NorChip Conference*, pp. 166–173, Turku, Finland, November 2000.

[18] R. Siegmund and D. Müller, “Efficient modeling and synthesis of on-chip communication protocols for network-on-chip design,” in *Proceedings of the International Symposium on Circuits and Systems (ISCAS ’03)*, vol. 5, pp. 81–84, Bangkok, Thailand, May 2003.

[19] A. Hemani, T. Meincke, S. Kumar, et al., “Lowering power consumption in clock by using globally asynchronous locally synchronous design style,” in *Proceedings of the 36th ACM/IEEE Conference on Design Automation (DAC ’99)*, pp. 873–878, New Orleans, La, USA, June 1999.

[20] J. Muttersbach, T. Villiger, and W. Fichtner, “Practical design of globally-asynchronous locally synchronous systems,” in *Proceedings of the 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC ’00)*, pp. 52–59, Eilat, Israel, April 2000.

[21] J. Muttersbach, *Globally-asynchronous locally-synchronous architectures for VLSI systems*, Ph.D. thesis, ETH, Zürich, Germany, 2001.

[22] S. Bhunia, A. Datta, N. Banerjee, and K. Roy, “GAARP: a power-aware GALS architecture for real-time algorithm-specific tasks,” *IEEE Transactions on Computers*, vol. 54, no. 6, pp. 752–766, 2005.

[23] E. Brunvand, “Implementing self-timed systems with FPGAs,” in *FPGAs*, W. Moore and W. Luk, Eds., pp. 312–323, Abingdon EE&CS Books, Abingdon, England, 1991.

[24] K. Erickson, “Asynchronous FPGA risks,” in *Proceedings of the 4th Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD ’00)*, Laurel, Md, USA, September 2000.

[25] A. Aubert, N. Bochard, and V. Fresse, “An adaptive embedded architecture for real-time PIV algorithms,” in *Proceedings of the 14th European Signal Processing Conference (EUSIPCO ’06)*, Florence, Italy, September 2006.

[26] R. J. Adrian, “Particle image techniques for experimental fluid mechanics,” *Annual Review of Fluid Mechanics*, vol. 23, pp. 261–304, 1991.

[27] R. K. Keane and R. J. Adrian, “Theory of cross-correlation analysis of PIV images,” *Applied Scientific Research*, vol. 49, no. 3, pp. 191–215, 1992.

[28] Altera Corp., “Altera Stratix II EP2S60 NIOS II Development board,” datasheet, 2004, http://www.altera.com.

[29] M. Leeser, S. Miller, and Y. Haiqian, “Smart camera based on reconfigurable hardware enables diverse real-time applications,” in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM ’04)*, pp. 147–155, Napa, Calif, USA, April 2004.

[30] T. Schiwietz and R. Westerman, “GPU-PIV,” in *Proceedings of the Vision, Modeling, and Visualization Conference (VMV ’04)*, pp. 151–158, Stanford, Calif, USA, November 2004.

[31] T. Fujiwara, K. Fujimoto, and T. Maruyama, “A real-time visualization system for PIV,” in *Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL ’03)*, vol. 2778 of *Lecture Notes in Computer Science*, pp. 437–447, Lisbon, Portugal, September 2003.