A New Self-Managing Hardware Design Approach for FPGA-based Reconfigurable Systems

S. Jovanović, C. Tanougast, and S. Weber

Université Henri Poincaré - Nancy 1
Laboratoire d'instrumentation et électronique (LIEN)
54506 Vandoeuvre lès Nancy, France
`slavisa.jovanovic@lien.uhp-nancy.fr`

Abstract. Given the scale and complexity of today's systems, it is of increasing importance that they handle and manage system's problems on their own in an intelligent and autonomous way. To cope with all non-deterministic changes and events that dynamically occur in a system's environment, a new "self-managing based" design approaches must be developed. Within this framework, an architectural network-based approach can be a good solution for the high demanding computation and self-managing needs. In this paper, we present the basic concept of such a design approach, its main properties and uses. A case study that proves the feasibility of this approach is demonstrated and validated on a small image-processing application.

1 Introduction

In literature, we can find lots of terms and definitions relative to a system that exhibits a form of a life-like system. This system is characterized by intelligent and autonomous way of the system handling and managing itself. Within this framework, the term of self-organization appears. Finding a clear definition of this term that would satisfy everyone is extremely difficult because the concepts behind it are still only partially understood. However, one considers that the self-organizing systems are the systems that try to behave like natural systems whose behaviour emerges and evolves without an outside intervention or programming [1]. The main properties of these systems are high degree of robustness, flexibility and adaptability allowing tackling problems far more complex than any computer system ever made. They are also characterized by: interactivity, dynamic, decentralized control, increase in order of their internal structure and autonomy.

Another more restrictive and less general term relative to self-organization that we can find in literature is the term of autonomic computing [2]. The autonomic computing systems function largely independently of their human supervisors, adapting, correcting and repairing themselves whenever a problem occurs. They called them autonomic, referring to the autonomic nervous system which controls bodily functions such as respiration and heart rate without requiring any conscious actions by the human being.

Implementation of self-* properties in technical domains presents a great challenge. This work seems more probable and realistic with the advent of FPGA reconfigurable technology. To face all non-deterministic (wished or unwished) events without the external control, such systems must have possibilities to evolve and adapt their functionality by changing their hardware structure. Some advantages of the FPGA reconfigurable technology, such as a spatio-temporal allocation by a run time reconfiguration and a partial reconfiguration match very well to these dynamically changing environments [8, 7]. Within this framework, we present our new design approach which allows us to design a system exhibiting a sort of self-managing properties.

This paper is organized as follows. In the Section 2 we present some related work on architectural concepts for autonomic computing systems. Section 3 details our design approach. The basic concept of this approach and its main properties are presented. Section 4 proves the feasibility of this approach on an image-processing application. At the end, in Section 5 some conclusions are given.

2 Background and related work

The Autonomic Computing was presented as an IBM initiative to face the growing IT (***I****nformation* ***T****echnology*) complexity crisis [2, 9]. As a result of this initiative, the four main areas of self-management, as an essence of the autonomic computing, are defined:

- Self-configuration : automatically configure components according to high-level policies to adapt them to different environments
- Self-optimization: Automatically monitor and adapt resources making appropriate choices to ensure optimal functioning regarding the defined requirements
- Self-healing: Automatically detect, diagnose, and repair hardware and software faults.
- Self-protection: Anticipate, identify, and prevent from arbitrary attacks and systemwide failures

IBM has expanded these autonomic and self-management concepts with the following additional criteria [4, 5, 6]:

- The system must perform something like healing - recovering from routine and external events
- The system continually optimizes, never maintaining the status quo
- The system requires self-knowledge of the whole and the components of the system

The basic building block of an autonomic system is the *autonomic element*. It consists of a managed element and its autonomic manager that controls it in accordance with defined policies. The managed element can be either a hardware or software resource or a combination of both. Essentially, an autonomic element consists of a closed control loop which can theoretically can control a system without an external intervention.

In [12], the authors describe behavioural properties of the autonomic elements, their interactions and how to build an autonomic system starting from a collection of the autonomic elements. They define some required behaviour of the autonomic elements. The autonomic element must handle problems locally, whenever possible. Its services must be defined accurately and in a transparent way to the other neighbouring elements. It must reject any services that would violate its policies or agreements. They also define different forms of policy at different level of specifications that the autonomic elements must respect in order to ensure desired behaviour. An overview of the policy management for the Autonomic Computing is given in [10].

In [11], the authors state that the core problems still remain. Those are the lack of appropriate standards and a precise definition describing self-managing system, the absence of mechanism for rendering self-managing systems adaptive and capability to learn . They expect the other scientific disciplines such as biology, physics and sociology to contribute towards vital concepts enabling the current systems to overcome existing problems of self-management.

An autonomic system by these definitions is highly desirable in the context of either software or hardware applications or a combination of both. For designing a hardware autonomic computing system with above mentioned properties, some aspects regarding the real-time constraints, failure-detection and fault-tolerance must be considered. Some works on failure-detection have already been done [3]. In contrast with these previous works based generally on software approaches, we focus on hardware self-management approach that realize the self-* properties without a closed control loop.

3 Architectural Design Approach for Self-Management

3.1 Introduction

We apply a classic reductionist approach for a better understanding of functioning of how the whole system functions. Let us suppose that there is a system S which is composed of N modules E_i, $0 < i < N$, that can communicate and exchange data with each other. Each module E_i carries out a function e_i at a given time. Each module's function presents a part of a set of the available functions (or services) that could be accomplished by the given module, S_i, and contributes to the system's global function e_g. Defined that way, the system can be described by the following set of expressions:

$$e_1 \in S_1, S_1 = \left\{e_{1_1}, e_{1_2}, ..., e_{1_{n_1}}\right\}$$

$$e_2 \in S_2, S_2 = \left\{e_{2_1}, e_{2_2}, ..., e_{2_{n_2}}\right\}$$

$$\vdots \tag{1}$$

$$e_N \in S_N, S_N = \left\{e_{N_1}, e_{N_2}, ..., e_{N_{n_N}}\right\}$$

$$e_g \in S_g, S_g = \left\{e_{g_1}, e_{g_2}, ..., e_{g_{n_g}}\right\}$$

$$e_g = \cup_{i=1}^{N} e_i$$

where $n_g = n_1 n_2 \dots n_N$ is the maximal number of different functions that this system can perform with a guaranteed service quality. Some functions of the system can be performed with a smaller number of modules to the detriment of the system's performances.

Let us also suppose that for each module's function e_i exists a set of the other modules' functions $\{e_{i_1}, e_{i_2}, \dots, e_{i_r}\}$ ($i \neq i_1 \neq i_2 \neq \dots \neq i_r$ and $r \geq 3$) that can at the same time successfully replace the function concerned and perform their own functions, as it presented by expression 2:

$$e_i \subset (e_{i_1} \cup e_{i_2} \cup \dots e_{i_r}) \tag{2}$$

Without a loss of generality, we limit the set of the modules' functions to three and consider that the modules whose functions are in the given set of functions present direct neighbouring modules. This means for each group of 4 neighboring modules of the system, the statement described by the expression 2 is always valid. For a rectangular structure of modules $n_c \times n_r$, where n_c and n_r are numbers of columns and rows respectively, and for a module E_{ij} with (i, j) coordinates $(0 < i < n_c, 0 < j < n_r)$, there exist 4 sets of functions which can replace functionally the concerned module (see Figure 1).

3.2 Modules awareness - Flux

To make all modules aware of their neighbours and their states, we have introduced a dynamic data stream called *flux*. This is the main originality of our work. The main objective of this data stream is to gather information about the states of the modules and to inform the modules about the states of their neighboring modules.

The flux circulates through the module, gathers information about its function, current functioning mode, states and then informs the other modules about it. The flux is not a control structure. It does not impose any decisions on the module. Its main role is gathering and informing. On the other hand, the flux helps modules to make decisions taking into account agreements with their direct neighbours and their states.

Neither of the modules can make a decision before it informs its direct neighbours and gets from them a sort of agreement for the wished decision. Whenever a change of some parameters occurs, each module "proposes" its solution and through the flux informs other modules about the change and action that it is going to perform. Other modules analyse the flux, become aware of the changes

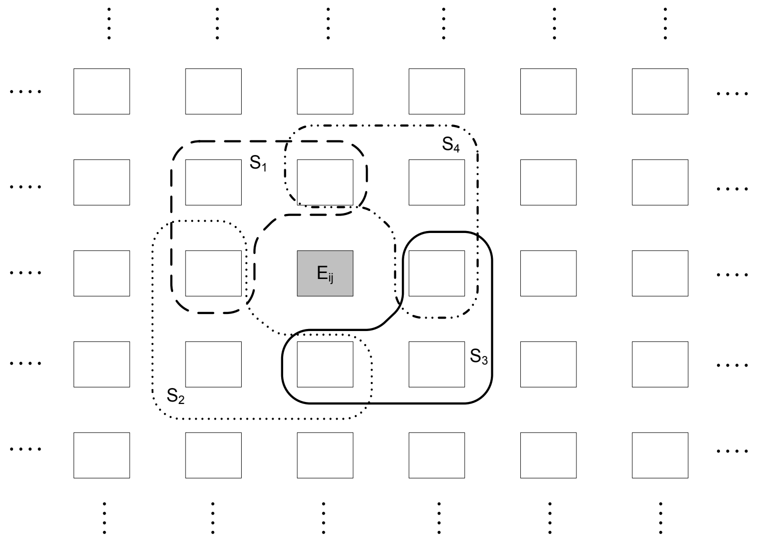


Fig. 1. For module E_{ij} there exist 4 sets of functions with which it can be replaced

and give a (dis)agreement for the proposed actions and eventually if they agree with the actions that will be taken, complement them with their own actions. Each module updates the flux about its intentions and actions that it will take.

If we compare our module and the *autonomic element* as defined in autonomic computing [2, 9], the "module's *manager*" makes decisions in accordance with the neighbouring modules. This is the consequence of its continuous monitoring and controlling by the other modules through the flux. That way the module cannot make a decision that would violate its own and policies and agreements of the other modules.

Moreover, the flux allows the system to handle the problems locally, whenever it is possible. For example, it can happen that a module discovers that its neigbouring modules which is on the way of the flux cannot perform the demanding services. In that case, it tries with the other modules affected by the same flux, to resolve the existing problem. If they do not succeed in it, there are three more fluxes comprising the same module and their modules will try to resolve the problem as well. In the worst case, if the problem cannot be solved locally, the rest of the system will become progressively aware of it. This is presented in Figure 2. The direct neighbouring modules of the module E_{ij} with the problem are shaded and surrounded by the first dotted circle. If they do not succeed in solving of the problem, the other modules (between two dotted circles) become aware of it and so on for the rest of the system.

3.3 Flux structure

Figure 3 details a structure of the flux affecting 4 modules. The number of fields corresponds to the number of modules affected by the flux and it can be extended. Each module has its field in the flux. Each module's field contains several subfields. The number of subfields is not limited and it can be extended depending on designer's concepts. Each subfield refers to a module's parameter. These parameters give information about the functional correctness, the intention of module removing (for reconfigurable systems), the module's current functioning mode and wished functioning mode. The current functioning mode is the function which the module executes at given time. The wished functioning mode is the future functioning mode that will be executed by the module and it presents a result of an analysis of functioning modes of other modules affected by the same flux. This is more detailed in the next section. In Figure 3, the first subfield denotes the functional verification information. If this subfield is set, the module executes correctly its function otherwise its function is corrupted and it must be replaced. If a module must be replaced because of its corrupted function or from another reason (i.e. defragmentation of a reconfigurable area), the second subfield called "Presence" indicates these information. The current

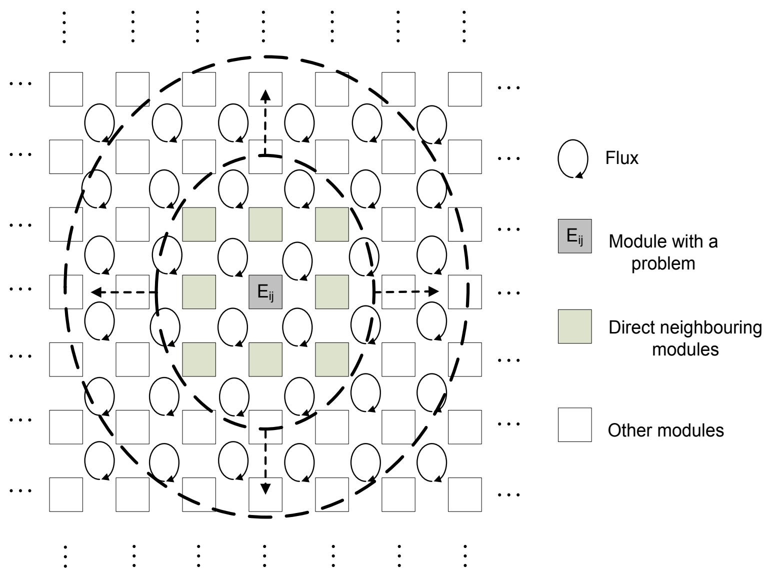


Fig. 2. The system solves the existing problem locally, if it is possible, if not, the rest of the system becomes aware of it. The first neighouring modules, if they cannot solve the problem, through the fluxes inform progressively the rest of the system. The arrows indicate the sense of the system's awareness for the existing problem

functioning mode is contained in this subfield whereas the wished functioning mode is placed in the last one (see figure 3).

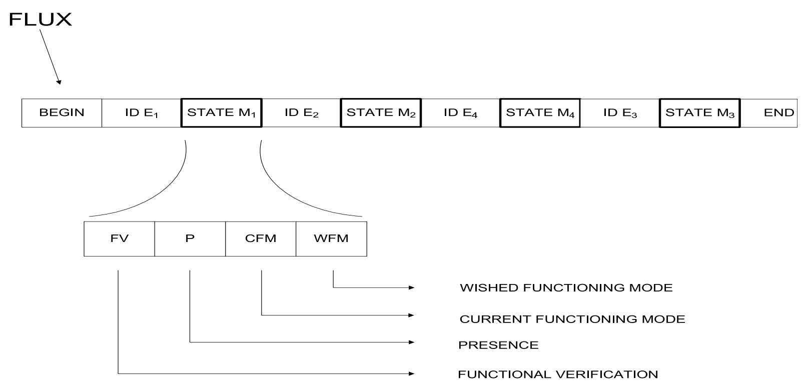


Fig. 3. An example of the flux in the case when it covers a group of 4 modules

3.4 Modules descriptors

To ensure establishing and maintening relationships with the other modules through the flux, the module uses *descriptors*. The descriptor is defined as a static structure that is used to describe the module in an accurate way.

Each descriptor gives a faithful image of the module. It contains module's functioning modes, services that it can deliver, the ways how the services can be replaced and all information that could be useful for the other modules.

The descriptor is created at the system's designing phase but can be updated at run-time, if the concerned module would like to add some additional services to its basic set of services. The descriptor of a module is placed in its first neighbouring module in the way of circulation of the flux. That way, in the case of the complete failure of the concerned module, the neighbouring modules have the "recipe" to recover its services and thus the global system function. This mechanism is presented in Figure 4b on the example of 4 modules. Each module contains its own descriptor and the descriptor of its direct neighbour. In Figure 4a is presented the descriptor which contains only functioning modes of other modules. The module's descriptor can be changed at the run-time, if the concerned module changes its structure or gets some "new functions" that are not previously described by the descriptor.

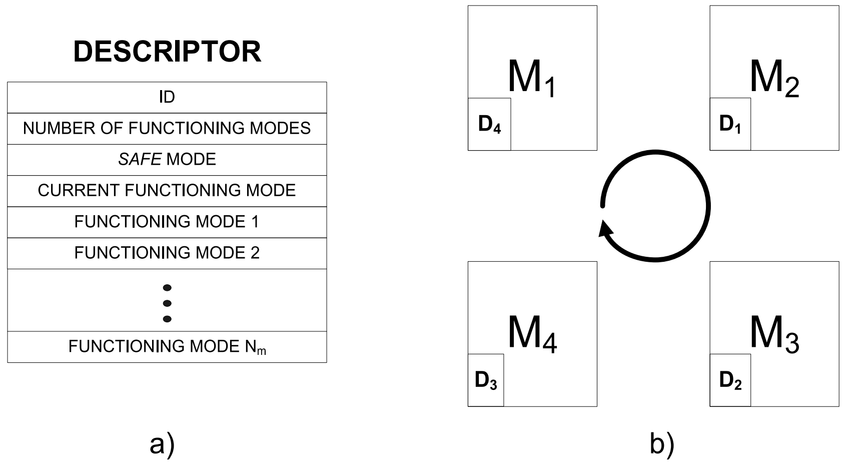


Fig. 4. a) Descriptor b) Placement of descriptors in a group of 4 modules affected by the same flux

3.5 Self-Management through the flux verification

Figure 6 shows the control flow graph of the flux verication block. Each module contains this block in which the received flux is analysed and treated. Firstly, each module verifies the states of other modules affected by the flux. Secondly, it updates its own states taking into account the states of other modules. For example, in the case of 4 modules covered by the flux, each module verifies the states of 3 others (see Figure 5).

The flux verification part is presented in Figure 6 with the dotted block called "Verification". Each subfield of each module's field is analysed. For instance, if the functional verification field of a module refers to a functional corruption of the module, the module that analyses the flux must take some actions and must adapt its functioning to the occurred problem. If it has not the descriptor of the faulty module, it waits for it from the direct neighbouring module placed in the sense of circulation of the flux. If the module has the descriptor of the faulty module, it will send it with the flux to other modules. Once the descriptor is received, the module compares its own descriptor with the one of the faulty module and it takes a decision. It chooses another functioning mode in which it will cover its own function and a part of the function of the faulty module. This phase is presented with the "Decision making" block in Figure 6. After having verified all modules' states, the flux verification block of the module updates its own states and sends the flux and eventually the descriptor to other modules.

Each module except the faulty one carries out the same procedure and after one flux circulation tour is finished, the faulty module module should be replaced by other modules. Of course the modules take also into account the changing of the flux circulation path which will not take anymore the faulty module.

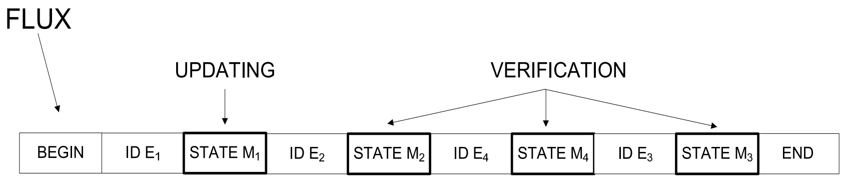


Fig. 5. Each module verifies the fields of the flux corresponding to the states of other modules affected by the flux and updates only the field corresponding to its own states

This self-managing of the modules is possible if we have only one failure or another demand at the time. This approach can be simply extended to cover other cases but that will cause large system area overheads.

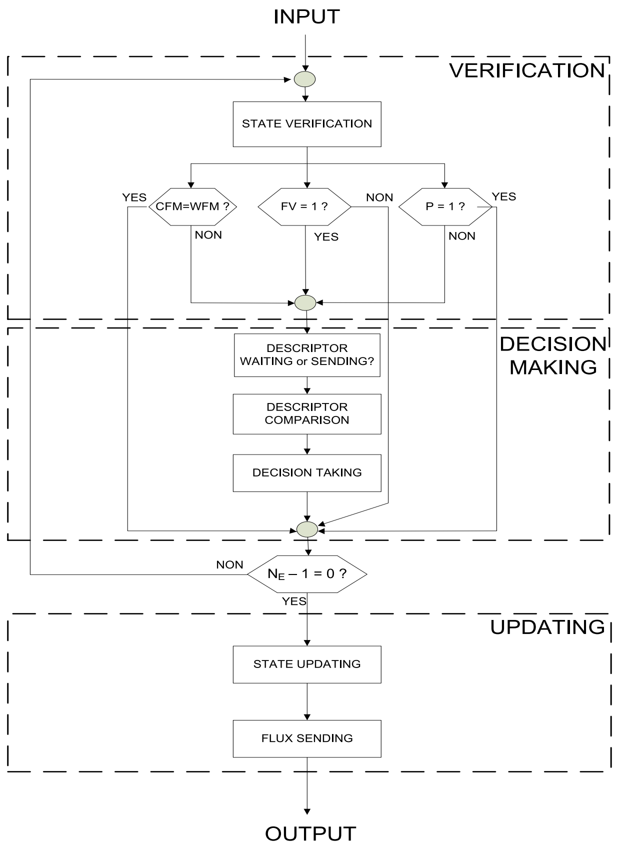
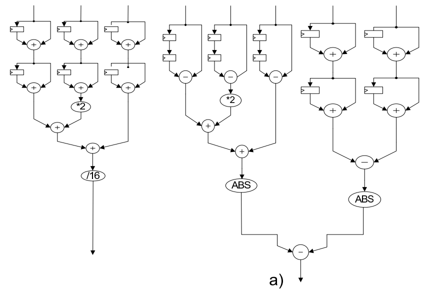


Fig. 6. Flux verification



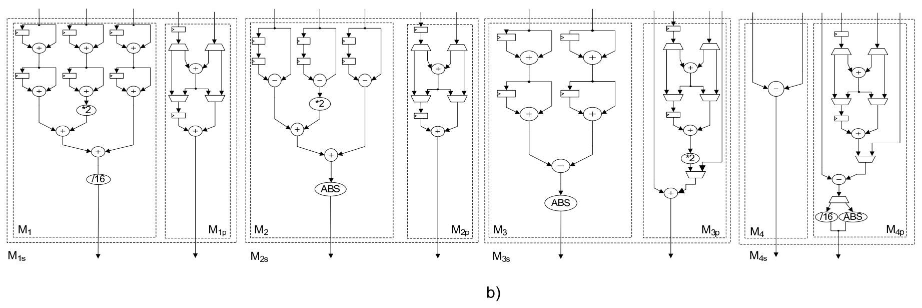


Fig. 7. Case study: an edge-detection image-processing application implemented on the CuNoC using proposed design approach: a) DFG of the application before and b) after applying proposed design approach

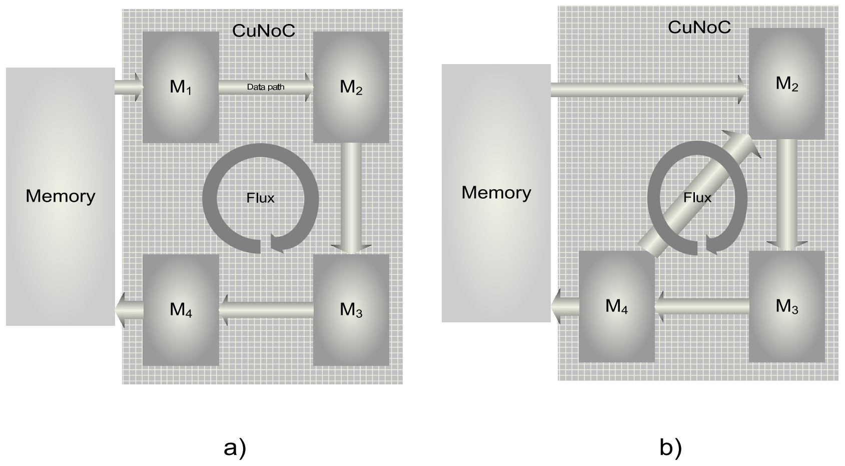


Fig. 8. Simulation cases before (a) and after (b) the failure of the module M_1

4 Case study

We have implemented the flux on the example of an edge-detection image-processing application. This application is composed of an averaging filter followed by two spatial edge detectors and a final gradient computing [13]. The data-flow graph of this application is divided in four parts, each part is carried out in one module (see Figure 7a). For each module we have added some additional resources in order to respect the statement 2 from Section 3. That means, each of the 4 presented modules can be replaced with 3 others in the detriment of system's performances. These additional logics are presented with the M_{1p},M_{2p},M_{3p} and M_{4p} respectively for the M_1,M_2,M_3 and M_4 modules (see Figure 7b). We have used the descriptors which contain only the modules' functioning modes. As a communication medium for the modules, we have used the 4×4 CuNoC communication approach [15, 14]. The flux also circulates via the CuNoC. In this case, the flux is implemented as a data packet and its structure is presented in Figure 8c.

We have simulated the complete failure of each module of the system at different times. Figure 7a illustrates the applied procedure on the failure of the module $M1$.

At a given time, the problem occurs and the other modules ($M2$, $M3$ and $M4$) through the flux become aware of it and make some decisions. They chose another functioning mode which covers a part of the M_1 function. After a while when the flux accomplished the circulation tour and all modules became aware of the problem, all services of the module $M1$ are replaced. Then the global function of the system is established again. The system's response time to occurred problem is equal to a time needed to flux to accomplish the circulation tour. During this time, all modules keep running their previous services till they become aware of existing problem of one of the modules. For the M_1 failure simulation case, instead of the previous data path comprising the modules $M1$, $M2$, $M3$ and $M4$, the new data path comprises only of $M2$, $M3$ and $M4$ but in the sequence $M2 - M3 - M4 - M2 - M3 - M4$. The first sequence of the modules $M2$, $M3$ and $M4$ is used to replace the function of the module M_1, and second one for their own functions. This is the main reason for the system's response time after reestablishing the system's global function which is much bigger than before.

We have implemented the system on Xilinx Virtex IV technology [16]. The implementation results show an area overhead of about 40 % with regard to the system without self-managing properties. This overhead is mostly due to additional logics which were implemented (M_{1p},M_{2p},M_{3p} and M_{4p}) in order to respect the statement 2 from Section 3. The area overhead due to flux verification block per computing module is insignificant.

5 Conclusion

In this paper, we presented a new hardware design concept making a system self-managing adapted for FPGA based reconfigurable systems. The originality of our approach is that the self managing is based on decentralized control from a scalable stream data which inform only the specification of the new interactions between system's element. The basic concept of this design, as well as its main characteristics are both presented. We have defined the dynamic and static structures called respectively *the flux* and *the descriptor* which present the main mechanisms of our design approach. These mechanisms are validated on an edge-detection image-processing application. As an ongoing work, some learning mechanisms that allow learning from past experience and remember effective responses are about to be considered as well as failure-detection and fault-tolerance mechanisms.

References

[1] Wolf, T. De, Holvoet, T.: Emergence versus self-organisation: different concepts but promising when combined. Engineering Self Organising Systems: Methodologies and Applications. Lecture Notes in Computer Science. Springer Verlag. **3464** 2005 1–15

[2] Kephart, J. O., Chess, D. M.: The Vision of Autonomic Computing. Computer **36-1** 2003 41-50

[3] Nordstrom, S. G., Shetty, S. S., Neema, S. K., Bapty, T. A.: Modeling Reflex-Healing Autonomy for Large-Scale Embedded Systems. IEEE Transactions on Systems, Man, and Cybernetics **36-3** may 2006

[4] Norman, D., Ortony, A., Russell, D.: Affect and machine design: Lessons for the development of autonomous machines. IBM Syst. J. **42-1**, 2003 33–44

[5] Pacifici, G., Spreitzer, M., Tantawi, A. : Performance management for cluster based web services. IBM Tech. Rep.2003

[6] Boutilier, C., Das, R., Kephart, J.: Cooperative negotiation in autonomic systems using incremental utility elicitation. Proc. 19th Conf. Uncertainty in Artificial Intelligence (UAI-2003) Acapulco Mexico aug 2003 89–97

[7] Xilinx XAPP290: Two Flows for Partial Reconfiguration: Module Based or Difference based. www.xilinx.com

[8] Compton, K., Hauck, S.: Reconfigurable computing: a survey of systems and software. ACM Computing Surveys (CSUR) **34-2** 171–210 (2002)

[9] Horn, P.: Autonomic Computing: IBM's Perspective on the State of Information Technology. IBM Corp. http://www.research.ibm.com/autonomic/research/papers/AC_Visi-on_Computer_Jan_2003.pdf 2003

[10] Agrawal, D., Lee, K.W., Lobe, J.: Policy-based Management of Networked Computing Systems. IEEE Communications Magazine 2005

[11] Herrmann, K., Mühl, G., Geihs, K.: Self Management: The solution to Complexity or Just Another Problem? IEEE Distributed Systems Online **6-1** jan 2005 1541–4922

[12] White, S. R., Hanson, J. E., Whalley, I., Chess, D. M., Kephart, J. O.: An Architectural Approach to Autonomic Computing. Proceedings of the International Conference on Autonomic Computing (ICAC'04) 2004

[13] Luk, W., Shirazi, N., Cheung, P.Y.K. : Modeling and Optimizing Run-time Reconfiguration Systems, Proc. IEEE Symposium on FPGAs for Custom Computing Machines (1996)

[14] Jovanovic, S., Tanougast, C., Bobda, C., Weber, S.: CuNoC: A Scalable Dynamic NoC for Dynamically Reconfigurable FPGAs. FPL Amsterdam Netherlands aug 2007

[15] Jovanovic, S., Tanougast, C., Bobda, C., Weber, S.: A Scalable Dynamic Infrastructure for Dynamically Reconfigurable Systems. ReCoSoC07 Montpellier France jun 2007

[16] Virtex-4 Family FPGAs. www.xilinx.com