A Hardware Preemptive Multitasking Mechanism Based on Scan-path Register Structure for FPGA-based Reconfigurable Systems

S. Jovanovic, C. Tanougast and S. Weber
Université Henri Poincaré, LIEN, 54506 Vandoeuvre lès Nancy, France
slavisa.jovanovic@lien.uhp-nancy.fr

Abstract

In this paper, we propose a hardware preemptive multitasking mechanism which uses scan-path register structure and allows identifying the total task's register size for the FPGA-based reconfigurable systems. The main objective of this preemptive mechanism is to suspend hardware task having low priority, replace it by high-priority task and restart them at another time (and/or from another area of the FPGA in FPGA-based designs). The main advantages of the proposed method are that it provides an attractive way for context saving and restoring of a hardware task without freezing other tasks during pre-emption phases and a small area overhead. We show its feasibility by allowing us to design a simple computing example as well as the implementation of AES-128 encryption algorithm which are presented in and detailed on the Xilinx Virtex FPGA technology.

1. Introduction

Essential for implementing multitasking on reconfigurable systems based on FPGA is the ability to suspend the execution of an ongoing task and to restore previously interrupted task. This mechanism, called preemption mechanism, is similar to task switch on modern CPUs and it was presented formerly in literature [1, 2].

Hardware preemption is an important mechanism which provide a mean for real time execution in the case of non-deterministic applications where the scheduling policy is not predefined like self-organizing systems, autonomic-computing, etc [3,4]. Indeed, the concept of the preemptive processing consists of suspending the execution of an ongoing task or process in order to run another high-priority task whose interruption demand was preceded and finally to restart the execution of the previously extracted task once the new one has finished.

The process steps of preemption consists of two key phases: the extraction and the restoration of the task's context which is in the registers and memories of the design. The goal of the extraction is to save the states of all the registers as well as all memories used in the circuit and to determine the starting point of the restoration process. On the other hand, the goal of the restoration is to restore the task's context of processing as it was before its interruption, in order to continue the execution previously stopped. Several techniques allowing the hardware preemption are proposed in the literature. The pros and cons of mostly used approaches we will disscus in the following.

The readback approach for context saving and storing of an outgoing task is one of the approaches which was mostly used for the FPGA-based designs. No additional hardware structures, simple implementation, no extra design efforts and no extra hardware consumption are the one of the most important advantages of this approach. On the other hand, the need for filtering state information from readback bitstream, technology dependence (extra knowledge about the configuration stream is necessary) and the freezing of the task while reading back its current state present some of the most important drawbacks. Examples of this approach can be found in [5] and [6].

Authors in [7] propose one solution of preemption for SRAM-based FPGAs and the architectures that store the configuration bitstream in SRAM cells inside the chip. This solution having been based on the readback approach gives some improvements mostly in fact that the process of the context saving of an outgoing task is combined with the process of partial reconfiguration needed for replacing outgoing task with another one.

Some other approaches similar to those mentioned above can be found in literature [8-10]. However, the common thing for all these approaches is that they are very costly and need significant resources in memory bits, chip area or logic resources. Moreover, these approaches have the same disadvantages as the readback approach.

As an alternative to all approaches (mostly readback stream based) for task's context saving and restoring are the hardware based approaches which need some modifications in a target design. All hardware preemptive structures for task's context saving and restoring need some extra interface which allows either normal mode of functioning or context saving or restoring the state information. Some of the biggest disadvantages of those

approaches are the important hardware consumption, extra design effort and in most cases, extra shutdown time. There are some solutions to cope with it like those which were proposed in [8]. However, in most cases, those solution lead to important time overhead (extra latency for context saving due to time needed for getting task in defined state also called switch point). On the other hand, the biggest advantages of those approaches are high data efficiency, independence from the used technology, no extra knowledge about bitstream. An example of hardware preemptive approach was presented in [11].

Another solution of hardware preemptive approach is the one based on register scan-path structure [7]. This paper addresses this design issue. Our proposed solution is based on modified scan-path register structure. The main advantages of this approach are high data efficiency (it reduces the amount of data to store), technology independence of used FPGA, no clock freezing during preemption phase (which allows functioning of other design's tasks and which is of main importance for the self-organizing reconfigurable systems), no extra knowledge about the size of the used registers (the register size identification part is done at the beginning of each task), small design effort (just replacing all used registers by those detailed in this paper) and relative small area overhead. The most significant area overhead is due to the hardware preemption contoller which must be included to capture all interruption demands and to switch between saving and restoring phases (see for more details Section 2).

This paper is organized as followed. Section 2 gives an overview of our hardware task preemption method, details scan-path register structure used for our approach and the hardware pre-emption controller. In this section the control flow graph of this method as well as the area overhead for our presented register scheme and the comparison with classical register are given. Section 3 proves the feasibility and describes the use of the presented preemptive approach on the sample computing example as well as on the implementation of AES-128 encryption algorithm on Xilinx Virtex FPGA technology. The results and comparison of the preemptive and non-preemtive designs for each example are also given in this section. Section 4 gives a discussion about the main advantages and drawbacks of our proposed approach. The conclusion and future work are given in Section 5.

2. Hardware Preemption based on Scan-path Register structure

2.1. Overview

In order to apply our hardware task pre-emption approach based on scan-path register structure, several tasks and modifications must be done in a FPGA design. These modifications do not need much extra design effort to yield a specific FPGA-design preemptive. We distinguish four major phases:

- First phase, defining the states of the registers of the design's tasks which must be saved in the pre-emption phase in order to have enough information at any time to restore concerned.
- Second phase, the classical registers which were supposed to capture task's states must be replaced by the preemptive flip-flop (PFF) defined later (see subsection 2.2).
- Third phase, inclusion of the hardware preemptive controller (detailed in subsection 2.3).
- Last phase, mapping of the used PFFs and the preemptive controller.

Once the necessary modifications were done, the design is ready to handle all interruption pre-emption demands. Figure 1 presents synoptic scheme which illustrates the proposed hardware pre-emption approach on a design executing k different tasks at the same time. Each task depending on occurred interruption demands can be either replaced by another one by design reconfiguration or restored from memory in runtime. This approach needs one controller to handle all interruption demands of all tasks. Figure 1 illustrates clearly the inter-connections between all tasks of the design, preemptive controller and the memory for data saving.

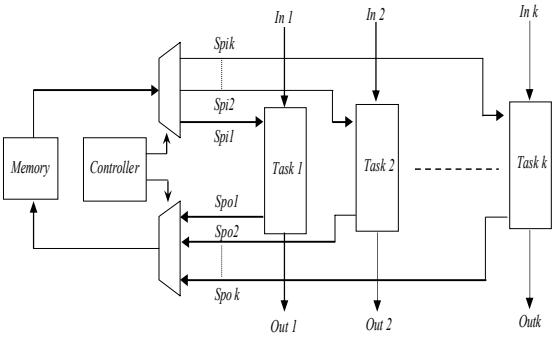

Figure 1. Synoptic scheme illustrating proposed hardware pre-emption approach

The control flow graph of this method is depicted in Figure 2. When the interruption demand of a task occurs, the hardware preemptive controller (detailed in Section 2.3) handles occurred preemption demand either by saving or restoring its context. In both cases, the concerned task can be replaced either after saving (the case of saving preemption demand) or before restoring its context (the case of restoring preemption demand). In the context saving phase the concerned task can be either replaced using partial reconfiguration of the used FPGA by another one which will take the zone being occupied

by this task and will load either a new context (new state informations loaded from outside) or already saved in memory by restoring it, or can be re-used with either new context or already saved in memory by restoring it. In the context restoring phase if a task whose context must be restored doesn't exist it must be implemented by replacing the one which is not in use any more by partial reconfiguration of the used FPGA or in other case, will be loaded with already saved context in memory. In this paper, the aspect of replacing a task by partial reconfiguration of the used FPGA is not considered.

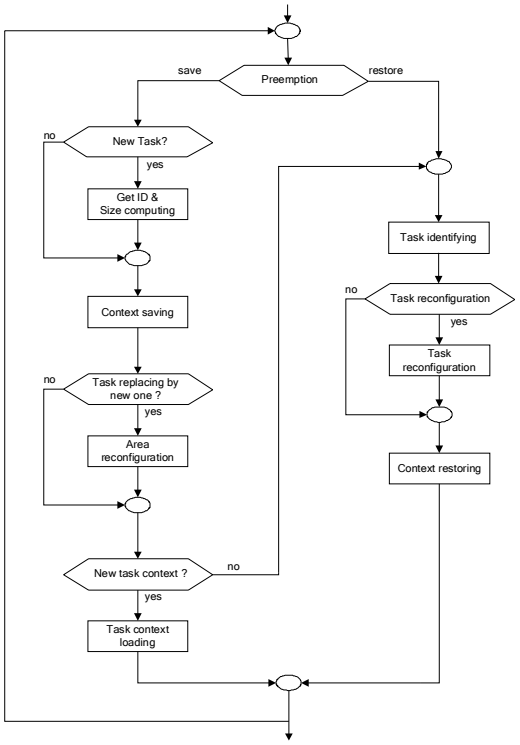

Figure 2. Control flow graph of proposed method

2.2. Preemptive flip-flop (PFF)

Our preemptive solution is based on configurable the scan path register structure. We distinguish two types of PFFs : Identifier PFF (IPFF) and classical PFF (CPFF). Figure 3 depicts a n-bits IPFF and CPFF as the part of this one.

Each task has one IPFF which is placed at the beginning of the scan-chain of the concerned task and whose main role is to allow the total register's size identification. This is a 2-bit register which is loaded at the initialization time with the vector "11" while all other registers which are PFF are reseted. The propagation of this vector allows the hardware preemptive controller to recognize the real size of all chained PFFs whose contents will be either saved or replaced by the one restored from memory during the pre-emption phase.

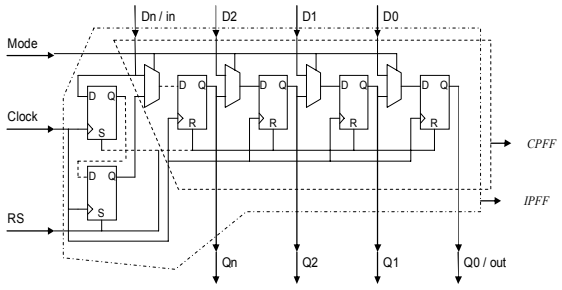

Figure 3. n bits IPFF and CPFF

The CPFF differs little to classical register. The main difference is the added combinational logic which allows two different modes of functioning: a normal or preemption mode. In the normal mode (*Mode = 1*), the CPFF behaves like a classical parallel register: the entries are propagated towards the exits via standard D flip-flops which are synchronized to the rising clock edge. On the other hand, in the preemptive mode (*Mode = 0*), the CPFF captures the current register's content (in the case of context saving the CPFF captures the content which will be saved, in the case of context restoring the captured register's content is not important) and becomes a shift register which allows propagation of the register's contents towards the exit which is associated to the least significant bit. The most significant bit of CPFF is associated to the least significant bit of previous CPFF and so on which presents a kind of scan-chain structure of all used registers in the design. The pre-emption mode allows context saving of all register contents through their serial propagation via all PFF to output (*Q0/out* in Figure 3) toward external memories or context restoring by serial propagation of all stored information from the input (*Dn/in* in Figure 3).

The added combinational logic does not contribute significantly to the area overhead. On the other hand, the evident performance decreasing is much less expressed in very large designs (see Section 4). Table 1 details the results of the comparison of 128-bits classical register (CR) and 128-bits CPFF which were implemented on Xilinx technology 4VFX12FF668.

Table 1. Comparison of the implementation results of the 128 bits classical and CPFF register

Comparison		IOs	CLB Slices	Dffs or Latches	FG	GB
CR		257	64	128	0	1
CPFF		260	66	128	132	1
Device utilization [%]	CR	80.3	1.04	0.97	0	3.13
	CPFF	81.3	1.07	0.97	1.07	3.13
Performance [MHz]	CR	3413.0				
	CPFF	**1043.3**				

2.3. Preemptive controller

The hardware preemtive controller is realized as finite state machine (FSM). Figure 4 details this FSM and describes the behaviour of the hardware preemptive controller. It consistes of 5 states: {*inititialization* state, *wait* state, *context saving* state, *context restoring* state and *get task's ID* state}.

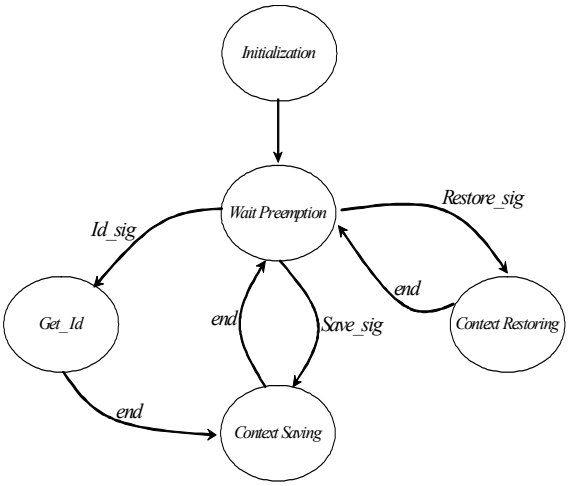

Figure 4. FSM of the hardware preemptive controller

After having been passed the initialization state, the preemptive controller is in the wait state where it waits for the interruption demands. When the interruption demand occurs, the controller examines the task's demand. If it is a new task and the context saving demand occurrs, the controller passes into the get ID state where it gathers all usefull information about the new task concerning this preemptive approach, that means: task's ID and total register's size. If it is an old one and the context saving demand occurs, the controller passes directly into the context saving state. After having finished the context saving state, the preemptive controller passes into the wait state where it waits for other interruption demands. Obviously, the context restoring demand cannot arrive for a task which is not previously saved in memory. In this case, the preemptive controller identifies the task to be restored, locates its position in memory and starts the restoring phase. At the end of this phase, the preemptive controller passes into the wait state.

3. Applications and implementation results

We prove feasibility of our proposed approach on two examples: on a sample computing example and on a more complex and larger design as the implementation of AES-128 algorithm. Those examples are presented and detailed in Xilinx Virtex technology.

3.1. Simple Computing Example

Figure 5 shows the synoptic sheme of this computing example which is already modified for the proposed preemptive approach. It contains 3 pipelined addition operators and 7 PFF (one IPFF and 6 CPFF) and one hardware preemptive controller. The classical registers are replaced with CPFFs, except the first one which is replaced with IPFF. The mapping of all PFFs, the controller and the rest of the design are done.

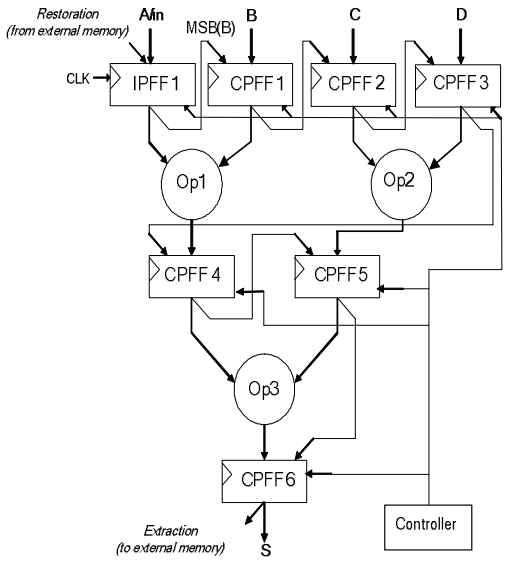

Figure 5. Proposed approach applied on simple arithmetic computing

Table 2 shows the comparison of the required logic resources for implementations with and without the applied preemption approach on Virtex-II technology 4VFX12FF668 [12].

Table 2. Comparison of the implemention results of the computing example from Fig. 5 with and without the applied presented pre-emption approach on Virtex-II technology.

Comparison		IOs	CLB Slices	Dffs or Latches	FG	GB
without (I)		41	28	56	24	1
with (II)		53	93	116	185	1
Device utilization [%]	I	46.6	10.94	7.22	4.69	6.25
	II	60.2	36.33	14.95	36.1	6.25
Performance [MHz]	I	319.9				
	II	208.7				

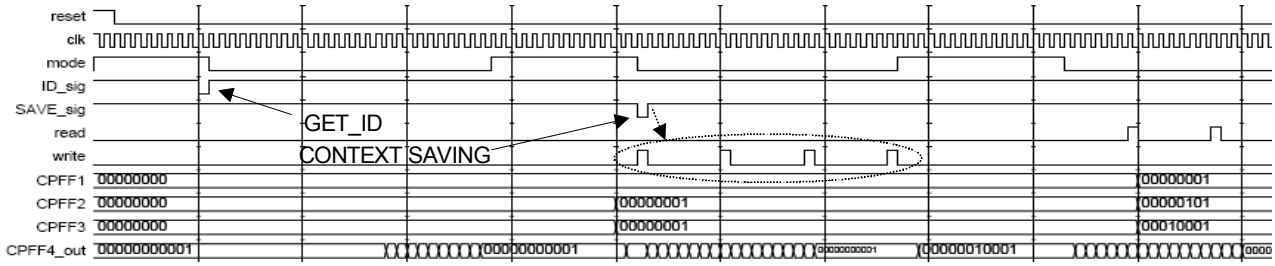

Figure 6. The snapshot of the simulation results

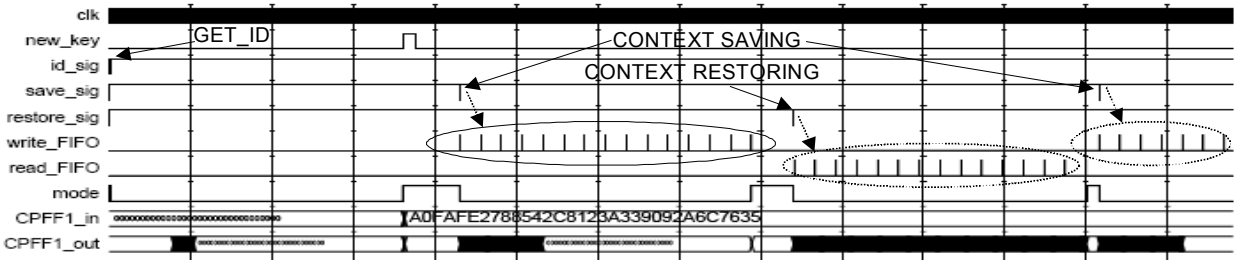

Figure 7. The snapshot of the simulation results of the implementation of the AES-128 algorithm using the proposed pre-emption approach

As was mentioned in Section II, the preemptive implementation requires additional resources. Indeed, there is a small area overhead due to the replacing of the classical registers with PFFs and a more significant area overhead due to the hardware preemptive controller implementation. The performance results are also done from which we can clearly remark design's performance degradation of, in particular, maximal authorized frequency about 30 %. In Figure 6 is presented the snapshot of the simulation results. The case of the context saving and context restoring are presented as well as the task's identification phase.

3.2. Implementation of AES-128 encryption algorithm

Table 3. Comparison of the implemention results of the AES-128 algorithm with and without the applied pre-emption approach on Virtex II technology 2V2000BF957

Comparison		IOs	CLB Slices	Dffs or Latches	FG	GB
without (I)		389	2439	4878	2822	2
with (II)		533	3183	6365	4827	2
Device utilization [%]	I	62.34	22.68	20.87	13.1	12.5
	II	85.42	29.60	27.23	22.5	12.5
Performance [MHz]	I	255.0				
	II	199.8				

This approach is applicable not only to the pipeline designs as is presented on the simple computing example in previous section but on all others containing the registers and needing pre-emption handling. The proposed preemption approach is implemented in the case of the AES-128 encryption algorithm (Advance Encryption Standard). This application is a good choice for our approach because this algorithm includes a data-path and controller parts. The main components of the AES architecture are the control module, the RoundKey generator and the encryption module. We can find more details about this algorithm in [13]. The implementation results on Xilinx Virtex II technology 2V2000bf957 are given in Table 3. From the performance results we can remark design's performance degradation of maximal authorized frequency about 20 %. If we compare with the performance results shown in Table 2, we see that this degradation is less important for more complex designs. In this implementation of AES-128 algorithm we can prove one of the main advantages of this approach: **no freezing of other tasks during the pre-emption phase.** The implementation of AES algorithm is divided into two parts: the first one which computes the keys needed for the data encryption (generator key) and the second one which encryptes data using the keys previously calculated (cipher part). The whole algorithm is implemented in a target FPGA with just one clock. The applied scenario which proves mentioned advantage consists of the following. The generator key computes the first set of keys for data encryption then the cipher part encryptes the input data. At one moment, after having encrypted several input data, the context saving demand arrives and the context saving preemption phase starts (including the identification part). At the same time, the generator key starts generation of a new set of keys for the next data encryption. In this way, the generator key (task I) functions correctly and properly during the preemption

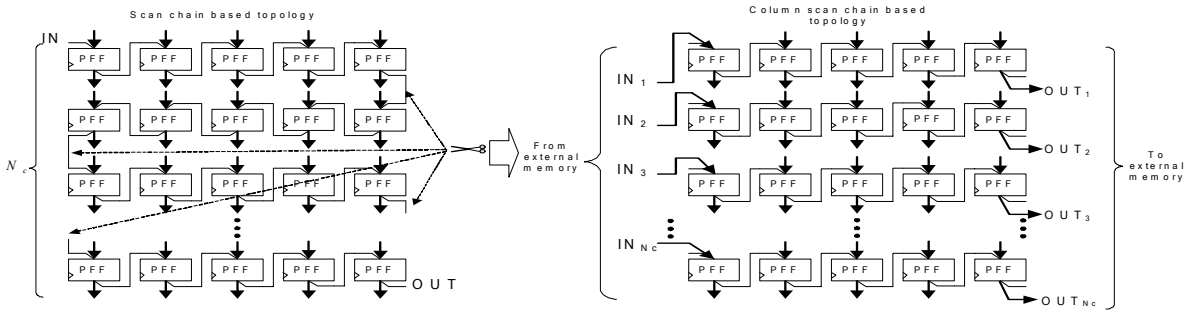

Figure 8. Schematic view of the solution for reducing the pre-emption time overhead

while the context of the cipher part (task II) is saving in the memory. Because of the sequential nature of the AES encryption, the new key computing in the generator key for a new data encryption is started before finishing the context saving phase. This fact allows us to cover the significant part of the time needed for the preemption phase. After having finished the context saving phase of the data encryption with the first set of keys, the data encryption starts with the new set. After a certain time, the new context restoring demand arrives, the context restoring phase starts and the cipher part restarts the previously interrupted task (the same task – cipher computing with the saved context). Figure 7 illustrates a snapshot of the simulation results which follow the mentioned scenario.

4. DISCUSSION

The process of context saving or restoring seems to be time consuming and depends on the total size of all used registers. If the number of registers whose states must be saved or restored is more significant, the time needed for saving the context or restoring the previous context is also more significant. The total time needed for the pre-emption phase is equal to equation (1) :

$$T_t = S_r \cdot N_r \cdot T_{CLK} \quad (1)$$

where the S_r represents the register's size, the N_r number of the registers whose states will be pre-empted and T_{CLK} the clock's period.

This is the major drawback of this approach. In order to significantly reduce this time overhead we present one of the possible solutions. The main idea is to partition off the scan register chain into N_c parts in order to obtain a column based topology of the used registers and to lead separately the ends of those parts to memory. The schematic view of this solution is presented in Figure 8. This solution increases the area overhead and makes the hardware preemptive controller more complex, but reduces significantly the total time needed for the pre-emption which is this time represented by the equation (2):

$$T_{t1\max} = S_r \cdot \frac{N_r}{N_c} \cdot T_{CLK} << T_t \quad (2)$$

where the $T_{t1\max}$ is the maximal time in the case of the N_c identical parts.

Another solution for reducing the important pre-emption time overhead without any changes in the design consists of the use of two clocks, the main fastest clock which will be used for pre- emption and will allow the fastest loading and unloading of the registers and the slowest which is derived from the main clock and will be used for the rest of the design. This solution does not increase significantly the pre-emption time overhead but has no need for additional resources.

Another major advantage of this approach (apart from the advantage of no freezing other tasks during the pre-emption phase which is detailed in the previous section) is the possibility of design automation without any changes concerning the pre-emption controller.

Moreover, the size of the used registers in the design is not important and the pre-emption controller does not take care about it because this information is obtained in the first phase of the context saving of a new task. That means the hardware pre-emption controller once implemented in the concerned design doesn't need any changes if the designer changes the size of the used register. In addition, this approach imposes just one hardware pre-emption controller by design.

5. Conclusion and future work

Our proposed solution gives an attractive and low-cost method for saving and restoring state informations of a hardware task by using the configurable scan-path register structure. Identifying total task's register size, small area overhead, easy implementation, no freezing clock during pre-emption and possibility of automating the design process are the some of the main advantages of this

approach. This preemptive scan-path based approach proved its feasibility by allowing us to design a simple computing example as well as the implementation of AES-128 encryption algorithm on the Xilinx Virtex technology.

The aspect of the replacing a task with a new one after having saved its context or before loading the saved context by using a partial reconfiguration can be developed with our approach. This aspect will need a new controller whose main role will beto take care of all demands for the area reconfiguration. The conception of this controller and its combination with the one proposed in this paper will present some of the objectives of our future work.

6. References

[1] J. Nehmar and P.Sturm, "Systemsoftware", dPunkt.Verlag, 1998.

[2] J.Ganssle and M.Barr, "Introduction to preemptive multitasking", Embedded Systems Programming, pages 55–56, 2003.

[3] Jeffrey O. Kephart, David M. Chess, "The vision of Autonomic Computing", IBM Thomas J. Watson Research Center, Hawthorne, NY, USA, Jan.2003.

[4] Soraya K. Mostéfaoui, Omer F. Rana, Noria Foukia, Salima Hassas, Giovanna Di Marzo, S. Chris Van Aart, Anthony karageorgos, "Self-organising applications : A survey", proceeding of AAMAS'2003 Workshop on Engineering Self-Organizing Applications, 15 July 2003, Melbourne, Australia. pp 62-69.

[5] H. Simmler, L. Levinson, and R. Manner, "Multitasking on FPGA coprocessors," in Proc. Of the 10 th International Conference on Field Programmable Logic and applications, Lecture note computer sciences, pp. 121–130, 2000.

[6] S.A. Guccione, D. Levi, and P.Sundararajan: JBits: A Java-based interface for reconfigurable computing. In Second Annual Military and Aerospace Applications of Pragrammable Devices and Technologies Conference (MAPLD), September 1999.

[7] Dirk Koch, Ali Ahmadinia, Christophe Bobda, Jürgen Teich, and Heiko Kalte. FPGA architecture extensions for preemptive multitasking and hardware defragmentation. In Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology, pages 433–436. IEEE, 2004.

[8] H. Kalte and M. Porrmann, "Context saving and restoring for multitasking in reconfigurable systems" International Conference on Field Programmable Logic and applications, IEEE Circuits and Systems Society, pp. 223-228, 2005.

[9] S. Trimberger, D. Carberry, A. Johnson, and J. Wong, "A time-multiplexed fpga," in Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines. IEEE Computer Society, 1997, p. 22.

[10] K. Danne, "Operating systems for FPGA based computers and their memory management," in ARCS, 2004, pp. 195– 204.

[11] Kyosun Kim, Ramesh Karri and Miodrag Potkonjak, "Micropreemption Synthesis: An Enabling Mechanism for multitask VLSI Systems", IEEE Transactions on Computer-aided design of integrated circuits and systems, vol.25. No 1, January 2006

[12] Xilinx Incorporated, "Datasheet of Virtex series FPGA".

[13] Joan Daemen, Vincent Rijmen, "A Specification for The AES Algorithm", Dr. Brian Gladman, V3.11, 12th Sept 2003. http://fp.gladman.plus.com/cryptography_technology/rijndael/aes.spec.311.pdf