Journal of Theoretical and Applied Information Technology
15th April 2023. Vol.101. No 7

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOFTWARE/HARDWARE TASK MIGRATION BASED ON A CHECKPOINT FOR MULTIPROCESSORS EMBEDDED SYSTEMS

KAMEL SMIRI[1], FAYCEL EL AYEB[2]

[1]Manouba University, Higher Institute of Multimedia Arts Manouba, Manouba University Campus, 2010, Tunisia.Carthage University, Tunisia Polytechnic School, SERCOM-Lab., Tunis, Tunisia
[2]Unit of Scientific Research, Applied College, Qassim University, Saudi Arabia.
GRIFT research group, CRISTAL laboratory, National School of Computer Sciences, Manouba University, Tunisia.
E-mail: [1]kamel.smiri@isamm.uma.tn, [2]F.ElAyeb@qu.edu.sa

ABSTRACT

Traditionally, the decision of implemented particular task in hardware or as software program running is typically taken at design time. This partitioning is static which has huge effect to the performance of a system. In the related work on Hardware/Software Co-design, the Hw/Sw partitioning problem is solved offline by mapping each task to dedicated core (Hard or Soft) with respect to certain objectives such that throughput, power consumption. But at now days and with the introduction of reconfiguration at runtime, the Software/Hardware partitioning problem can be solved online, thus, taking advantage of Hardware high performance and Software flexibility with the lowest possible costs. In this context, our approach focuses on these objectives. However, for a wide range of embedded application, the cohabiting software flexibility and hardware speed is a key feature to provide performing and flexible embedded systems. While in this system exit some tasks that are appropriate for a software program running on a processor and other tasks that might have highly throughput requirements that can be executed by dedicated hardware modules.

Keywords: *Dynamic Migration Of Soft/Hard Tasks; Embedded Systems; Checkpoint; Soclib; Gaut.*

1. INTRODUCTION

For Automotive, avionics systems and multimedia applications, massive parallelism with more 100,000 computing resources is the most viable way to meeting a high performance without an interruption. Future platforms will enroll even more computing resources to achieve this sustained performance. For each computing resources (software or hardware node) is composed of several hundreds of transistors such node will encounter a failure every 9 hours in average [1].

In general, several types of errors can be constituted a threat such as hardware failures and data corruptions. However, the execution of the treatments by purely software tasks being carried out on a given configuration of processors can't satisfy the needs of performance. In the same way, an entirely material execution presents the drawback of a high cost of design and a low flexibility. Therefore, many mechanisms and techniques have been developed to reach a high availability and reliability for distributed systems. In this context, Checkpointing is an effective methodology to cope with faults.

In this paper, we will present Checkpointing protocols known from software and hardware systems in section 1. After comparison and classification different mechanisms for software/hardware Checkpointing with respect to a series of desirable properties, we will propose an approach in which we will investigate the concept of Checkpointing known from software systems, can be utilized in hardware tasks running on reconfigurable devices.

After revealing related checkpoint mechanism, a related work on dynamic reconfiguration based software/hardware checkpoint will be presented in section 2.

In section 3, we will present a novel approach for increasing fault tolerance on the system MPSoC based on CPUs in combination with FPGA.

Typically, we will put focus on the problem of software/hardware co-design at runtime in order to

reach self-adaptive reconfigurable embedded systems.

Our approach address to migrate tasks implemented in software or hardware node from one node to another. Nevertheless, the fact of interrupting a software task and activating it in another hardware node at the same state of execution require for the availability of the control mechanisms for both hardware and software side.

In our strategy, we will introduce a software and hardware Checkpointing mechanisms for both task migration as well as reconfigurable platform.

A checkpoint is the information necessary to recover tasks. This information will be used to restore the application from the last valid checkpoint in which includes the global consistent (correct) state of the application. Moreover, our contribution consist to identify the valid checkpoint and to find its corresponding in the node in which receive a task migrated.

2. DESCRIPTION OF CHECKPOINT MECHANISM

During the last decade in all aspect of our daily lives have prevailed utilization embedded systems. In fact, the improvements in deep *Submicron technology* (using transistors of smaller size with faster switching rates) as well as predicted by Moore's law [2]. But, the downside of these technologies is a crisis of complexity design [3] that leads to decrease the efficient and performance systems.

Traditionally, the development of the embedded systems has been released by two groups. The first group develop the embedded software whereas the second staff prepare a hardware architecture can be elaborated and implemented onto SoC (System on Chip). Provided that this design space is the optimal or near optimal implementation. However, a hardware/software co-design methodology cope this increasing complexity of embedded systems. Moreover, the task migration and the hardware/software Morphing can solve this issue by giving the possibility to optimize the task binding or to reconfigure both the software and the hardware of the systems. These solutions based on the process checkpointing. A checkpointing technique is further used for the task migration which saves internal states during this process. In addition, task morphing is based on checkpointing technique in which allows migrating or swapping the execution of a task between software implementation (on a CPU) and hardware implementation (on a FPGA).

In the following, we will present protocols checkpointing for both software and hardware areas.

3. RELATED WORK : SOFTWARE CHECKPOINTING

In this section, we will present three categories of checkpointing and rollback recovery protocols: 1) Uncoordinated checkpointing, 2) Coordinated checkpointing and 3) communication-induced checkpointing. These protocols have given the possibility to ensure consistently recovery for the state in a system with multiple communicating tasks.

3.1 Uncoordinated Checkpointing

At this protocol, the domino effect present certainly. Consequently, a larger amount of useful work will get lost. This issue was due for uncoordinated taking checkpoints for each process which it decides independently when to store a checkpoint.

Many check points have to be stored and that may be useless. Subsequently, the systems have to rollback from the beginning of computation.

3.2 Coordinated Checkpointing

This scheme synchronizes the moment of taking check points for avoiding the domino effect.

Consequently, the coordinated checkpointing protocol has to insure that a consistent global state will be stored. But, the down side is a latency penalty will be paid in order to take synchronizing check points.

In distributed systems, communication between tasks can be blocked when applying check pointing. The coordinator sends a check point request to all tasks that will stop processing and flush their communication channels. After that, all tasks will return the entire check-points which received successfully messages to the checkpoint coordinator.

Furthermore, the coordinated checkpointing protocol can provide unblocked communication in which messages are stamped with a checkpoint index.

3.3 Communication-induced Checkpointing

The communication induced checkpointing protocol gives the possibility to a process to take a local check point autonomously, but however, for avoiding the domino effect, this scheme allows a checkpoint index that stamps which are piggybacked on all messages.

The main difference between coordinated and communication-induced checkpointing protocols is the process checkpointing in the second scheme will be forced when receiving message. As contrary to coordinated checkpointing, which applying checkpointing autonomously at the transmission messages.

3.4 Synthesis software checkpointing protocols

In fact, it exits other techniques for restoring the system state to the most recent consistent global state that are log-based rollback recovery. But, we are interesting by checkpoint-based protocols, because, there are less restrictive and simpler to implement than log-based protocols [4].

Table 1: Comparison between three categories of checkpoint-based rollback recovery protocols.

	Uncoordinated/Checkpointing	Coordinated/Checkpointing	Communication/Induced/Checkpointing
Checkpointin/process	Several	1	Several
Domino effect	Possible	No	No
Orphan process	Possible	No	Possible
Rollback extent	Unlimited	Last global checkpoint	Possibly several checkpoints
Recovery protocol	Distributed	Distributed	Distributed

Table 1 summarizes a comparison between the different of checkpoint-based protocols. These different scheme provide tradeoffs with respect to set of properties such that the extent of rollback, protocol of recovery, freedom from domino effect and from orphan process [5]. In first column, uncoordinated checkpointing allows the process to take autonomously its checkpoints. But however, it suffers from the domino effect and a complicated recovery. As opposed to uncoordinated checkpointing, the coordinated checkpointing protocol makes easy the recovery by synchronizing the moment of taking the checkpoints. Between these two protocols, exit communication-induced checkpointing protocol, which do not require the coordination and keep away from the domino effect [4].

4. RELATED WORK : HARDWARE CHECKPOINTING

In this section, three different methods for hardware checkpointing will be presented: 1) memory mapped state access (MM), 2) scan chain based state access (SC), and 3) shadow based scan chain state access (SHC).

4.1 Memory mapped state access

Each flip flop is directly accessible by the CPU via an address and a data bus by integrating the flip-flops storing the checkpoint into a read/writable memory space of a system. For keeping the states consistent during read or write operations, the module must be blocked until a state extraction or a rollback process has been finished [6].

4.2 Scan chain based state access

Instead of moving the flip values in the address space of the memory, the flip-flops can be chained together in Scan Chain. This is known from established ASIC design techniques. A multiplexer in front of each flip-flop is used to switch between a regular execution mode and a scan mode, where all flip-flops are linked together to form a shift register chain.

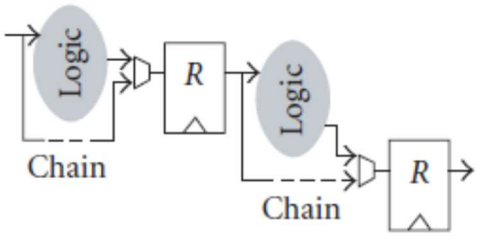

Figure 1: Hardware checkpointing using a scan chains

As depicted in figure 1, for keeping the state of the chain consistent, the output of the register chain is connected to the input forming a ring shift. So, the module can continue regular execution immediately after the checkpoint has been read.

4.3 Shadow based scan chain state access

As shown in figure2, each flip-flop of the original circuit is duplicated and connected to a chain.

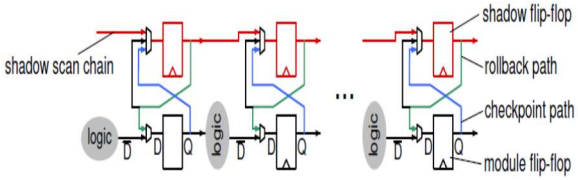

Figure 2: Hardware checkpointing using shadow scan chains

The shadow scan chain allows copying all values in one single clock cycle. Hence, it is possible to store, restore, or swap a checkpoint within one single clock cycle.

3.5 Synthesis hardware checkpointing protocols

These techniques differ in terms of the latency to extract a checkpoint (C) and hardware overhead (L) which required for saving and storing the checkpoints. Dirk Koch and al examined these

methodologies by implementing the different state extraction mechanisms in their approach. The results obtained will be present in Table 2.

Table 1: Results obtained by our approach by implementing the different state extraction mechanisms.

	#LUTs/H_L		#Flip-Flop/H_F
Original DES	2015/100%		984/100%
Scan Chain	2414/120%		1138/116%
Shadow chain	3937/195%		2025/205%
Memory mapped	2851/141%		1026/105%
	F_{max} [MHz]/P	C	L
Original DES	116/100%	-	-
Scan Chain	110/95%	10354	24979
Shadow chain	99/85%	0	16813
Memory mapped	107/92%	1306	16931

The table points out that each state extraction strategy is optimal in the sense of one of the defined properties. The shadow scan chain method leads in the case of high checkpoint rates to a higher throughput by the cost of almost doubling of the required logic resources. The simple scan chain approach demonstrates that it is possible to enhance a hardware module to be capable of checkpointing with an overhead of about 20% as compared to the original module [6].

5. RELATED WORK ON DYNAMIC HARDWARE/SOFTWARE

A first approach to dynamic hardware/software partitioning is presented in [10] [11]. There, the authors present warp configurable logic architecture (WCLA) which is dedicated for accelerating software kernels of embedded systems applications.
In [12], the authors propose a dynamic Partial Reconfiguration technology coupled with an Operating System for Reconfigurable Systems (OS4RS). In this work, they present a novel, lightweight scheduling mechanism supporting preemptable and clock-scalable hardware tasks.
A software-hardware cooperative method for multi-projector seamless tiled display system is proposed by Wang et al. by profiling the software that is currently running on an ordinary PC and the hardware image processors that are currently running on FPGAs [13]. In the FPGA domain checkpointing has been seldom investigated so far. Multi-context FPGAs have been proposed, that allow to swap the complete register set (and therefore the state) among with the hardware circuit between a working set and one or more shadow sets in a single cycle. But due to the enormous amount of additional hardware overhead, they have not been used commercially".

6. PROPOSED APPROACH

In the following, we will present a new approach which provides a novel medium for adapting to the new conditions, i.e., when the implementation task change its environments (Hardware to software or vice versa) at runtime by Dynamic Hw/Sw partitioning or Hw/Sw morphing [6]. As a matter of fact, these concepts have not been sufficiently studied [7]. Nevertheless, we investigate objectives Hw/Sw partitioning with utilizes checkpoint protocols until reaching an ideal platform featuring by high-speed and efficient dynamic reconfiguration mechanism. The objective of our dynamic Hw/Sw reconfiguration approach is not merely to change the task execution between different implementation alternatives, but also, without losing of internal states.
Our approach "Dynamic reconfiguration based checkpoint" will be integrated in an OS infrastructure which is shown in Figure 3. In which, the tasks implemented in hardware or software can be migrated from one node to another. This approach is destined to architectures based on field-programmable gate arrays (FPGA) in combination with CPUs.
We propose four main phases to realize our approach: 1) Checkpoint Identification, 2) Save and Store Checkpoint, 3) Translation process Sw/Hw and 4) Continue Execution.

6.1 Phase 1: Checkpoint Identification

Firstly, we choose a suitable software checkpoint protocol for our requirement specification. Besides, this protocol will be encoded and implemented as a plug-in in platform for virtual prototyping of MPSoC. Checkpoint Identification: in which determine where the failure intervene (during task execution, at recording data, during a transmission of a message, etc.) and identify the consistent global state.

6.2 Phase 2: Save and Store Checkpoint

In order to keep the continuity, the save checkpoint process must be triggered. And, the checkpoint would be stored into a read/writable memory space of a system.

6.3 Phase 3: Translation process Sw/Hw

As same thing for hardware checkpoint protocol, it will be written in some line of codes and integrated as plug-in in the High-Level Synthesis tool for automatic generation of hardware accelerators for FPGA. This work will be sketched in second step. Translation: the task execution can be translated between hardware and a software domain by reading a consistent state from one domain and transforming the state according to other domain.

6.4 Phase 4: Continue Execution

This phase resumes execution and maintains the continuity of its operations. The operating system again launches the execution of complete systems (application software part and application hardware part). The execution is restarted at the selected checkpoint.

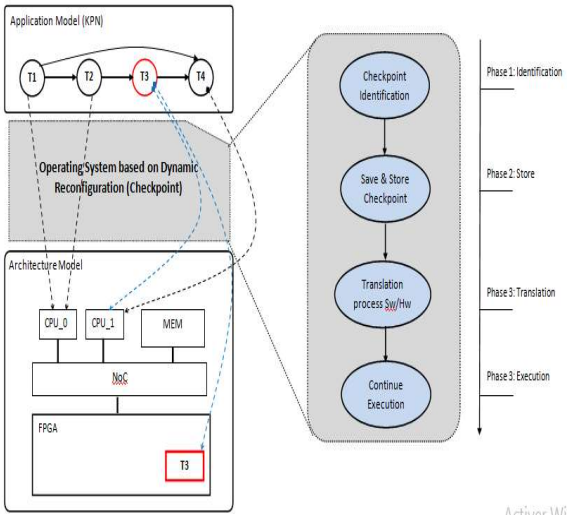

Figure 3: Proposed Approach: Software/Hardware Task Migration based on checkpoint

We have established a bibliographic synthesis in [19] [20] in order to get an idea of the new applications used in modern embedded systems. We have found that our approach is very effective for these applications. The use and implementation of our approach greatly improves the performance and robustness of these embedded systems.

7. EXPERIMENTATION

7.1 Architecture Description

The target is a ZedBoard included Xilinx chip of the Zynq-7000 family. ZedBoard is a SoC includes an ARM Cortex-A9 dual-core processor and an FPGA as well interfaces and functions required for a wide application panel. The ZedBoard card consists of two main parts:

- Programmable Logic (**PL**): contains FPGA. Due to financial constraints, we have to go through the PS section in order to program the PL. The PL configuration is carried out either by *Chips-cop pro* or either by mono *USB Debug JTAG cable*.
- Processing System (**PS**): contains processor ARM dual-core with its memory controllers and its peripherals.

7.2 Description MJPEG Decoder

The application used in the case study is decoder MJPEG. This code handles data per blocks of 8x8 pixels by using the technology of coding intra-screen. The decompression of an image JPEG (figure 4) is done in five most important stages: First actor VLD allows to analyze the input file and to decompress in blocks MCU "Minimum Coded Units". Then, each block passed through the opposite quantification and to reorder in ZIGZAG. The following task is the IDCT which makes it possible to transform the image starting from frequency field towards the space field. Finally, the conversion of the colors translates the colors of the blocks of components of an MCU to values of pixels; this task is carried out by actor LIBU [8] [15].

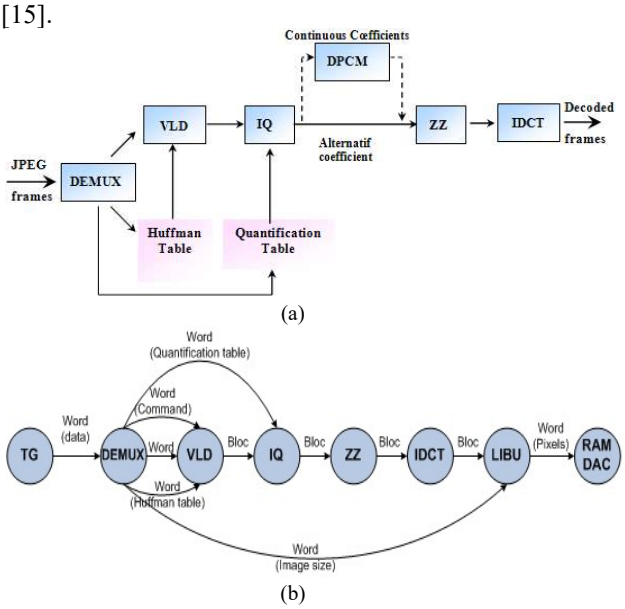

Figure 4: (a) Motion JPEG decoding principle, (b) model of Motion JPEG decoder

7.3 Tools Description

In this section, we will present tools that will be used in the experimental side. For the moment, we choose SoCLib and Gaut platforms because we familiar with these environments.

- **SoCLib**

The System on Chip Library (SoCLib) is an open framework for the virtual prototyping of MPSoC systems. This tool based on the simulation model and it respected the KPN formalism.

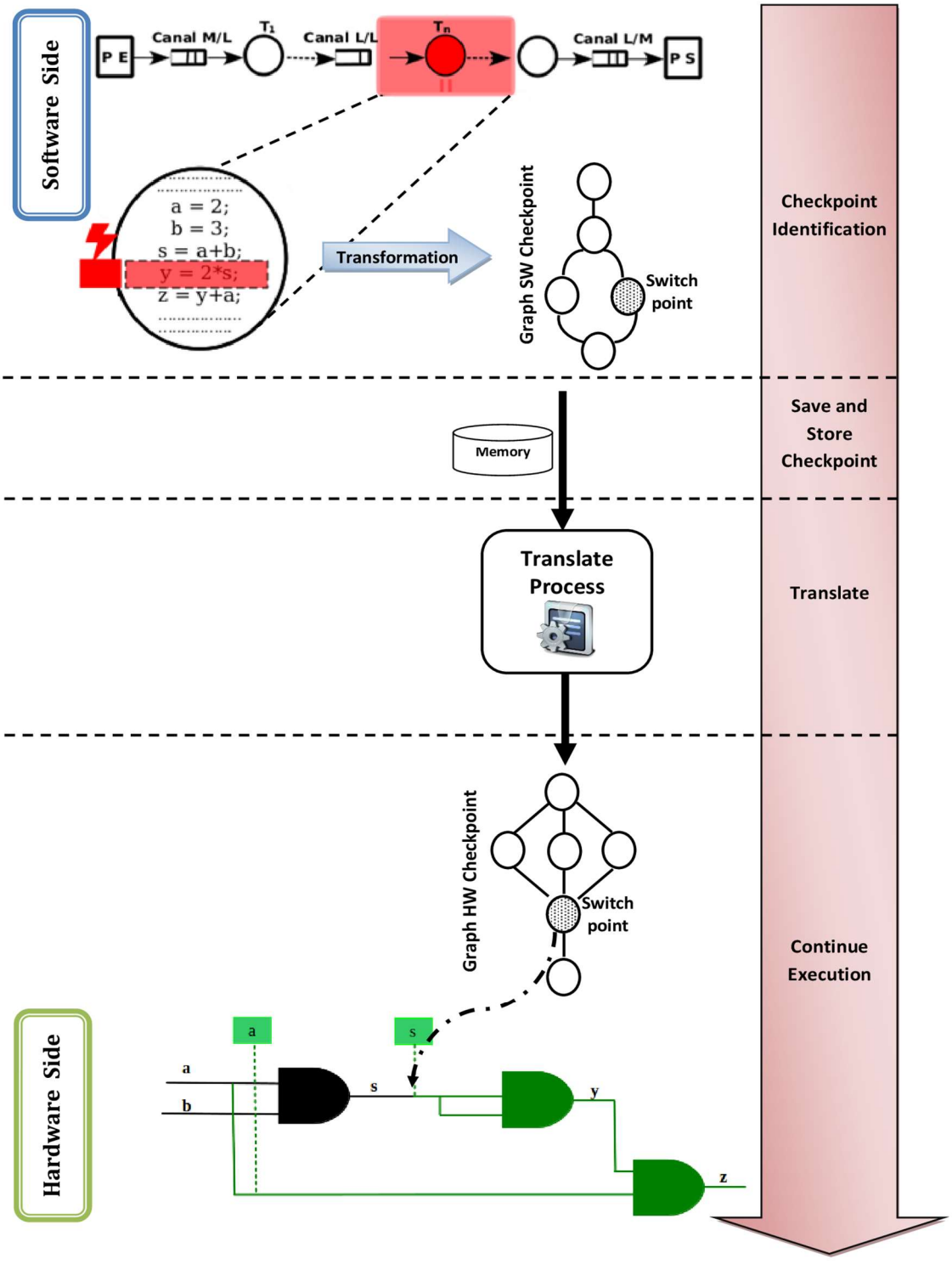

Figure 5: Design flow for dynamic reconfiguration based checkpoint

In which, the SoCLib allows the development a software application in the form of a set of parallel (allow to deal the pipeline parallelism) and communicating tasks [17]. These tasks can be implemented in hardware or software and use a Multi-Writer/Multi-Reader First-In/First-Out (MWMR FIFO) as a communication channel. Besides, SoCLib allows the simulation and the exploration of the hardware architecture via an open source library in two level abstractions (CABA and TLM) written in SystemC for the material components [14] [15] [18].

- **GAUT**

GAUT (Generator Automatic Processing Unit) is an academic and open source high-level hardware architecture synthesis tool dedicated to DSP digital signal processing applications.

With the bit-Accurate C/C++ specification, the GAUT tool automatically generates an RTL level architecture described in VHDL which can be used by logic synthesis tools on the market such as ISE (Xilinx), Quartus (Altera) or Design Compiler (Synopsis). Thus, Gaut also makes it possible to generate SystemC simulation models at the TLM and CABA levels via the virtual prototyping platform SoCLib[9].

This environment emerged in the 1990s at the LASTI research laboratory and has been pursued at LESTER since 1994. This tool currently represents a volume of around 70,000 lines of C/C++ and JAVA code.

The GAUT tool takes as input the description of the algorithm in C/C++ which must be synthesized using the CTM algorithmic class library produced by Mentor Graphics (American company founded in 1981 and its activity is in the electronic CAD field).

The functional constraints are the throughput (specified through an initiation interval which is represented by a period between the start of successive iterations) and the clock period. In addition, optional design constraints are memory mapping and timing diagram inputs/outputs.

The target model of the architectures synthesized by GAUT is composed of three functional units working in parallel: processing unit (UT), storage unit (UM) and communication unit (UCOM). The (UT) is a data path composed of logic or arithmetic cells, storage elements, driving logic and a controller (FSM). The (UM) is composed of memory banks and their associated controllers. The (UCOM) includes a synchronization processor and an operation memory which allow to have a communication interface of GALS/LIS [16].

7.4 Proposed Algorithm

The dynamic Software/Hardware online reconfiguration model is based on algorithm which will be illustrated in figure 6.

```
Algorithm Checkpoint
Variable

Begin
    For all task_tj IN Γ
        Init_task()
        Task_binary()
        Start_task()
        While (Exec_task (tj) does not finish)
            If (tj-> current_state == Running) then
                Continue (Exec_task(tj))
            Else
              If (tj-> current_state == Blocking) then
                Checkpoint_here()
                Write_checkpoint()
                t'j ← translate_task(tj)
                Find_corresponding-ckpt()
            Continue (Exec_task(t'j))
              End if
            End if
        End while
        J←j+1
    End for
End
```

Figure 6 : Dynamic Algorithm for Software/Hardware Task Migration

This algorithm illustrates the model steps. In order to identify certain states at which the execution can be switched between hardware and software which as shown in figure 6. The model begin by Init_task() for initialing task. After that, Task_binary() and Start_task() for loading binary implementation and launching the task execution. This novel model includes some sensor functions for controlling the states of tasks. If the current state is normal then the task continues its execution. When the state task is blocking then the checkpoint mechanism will be started its functions which are depicted in figure 5.

8. CONCLUSION

The computational requirements for embedded applications are increasing exponentially. This complexity, coupled with constantly evolving specifications, has forced designers to consider intrinsically flexible implementations. In this paper, we presented a first step toward a dynamic Software/hardware partitioning approach. And, we described how hardware-software morphing could be implemented. The target and the tools identified at last section with explaining a proposed algorithm. Future work will focus on

implementation of our methodology and will compare different experimental results examples.

REFERENCES:

[1] J. Dongarra, T. Herault and Y. Robert, "Fault tolerance techniques for high-performance computing", Chapter1: Fault-Tolerance Techniques for High Performance Computing, Springer international Publishing Switzerland, 2015.

[2] G. E.Moore, "Cramming more components onto integrated circuits," Electronics, vol. 38, no. 8, pp.114–117, April 1965.

[3] J. Henkel, "Closing the SoC design gap" , IEEE Computer, vol. 36, pp. 119–121, September 2003.

[4] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. 2002. "A survey of rollback-recovery protocols in message-passing systems". ACM Comput. Survey. 34, 3 (September 2002), 375-408.

[5] Akanshika, "Analysis of Rollback Recovery Techniques in Distributed Database Management System", International Journal of Modern Engineering Research (IJMER) Vol.3, Issue.3, May-June. 2013.

[6] Dirk Koch, Christian Haubelt, Thilo Streichert, Jurgen Teich., "Modeling and Synthesis of Hardware-Software Morphing", IEEE International Symposium on Circuits and Systems, New Orleans, May 2007.

[7] Ihsen Alouani, Braham L. Mediouni and Smail Niar, "A Multi-Objective Approach for Software/Hardware Partitioning in a Multi-Target Tracking System", International Symposium on Rapid System Prototyping (RSP), Amsterdam, 8-9 Oct. 2015.

[8] A. Jemai, K. Smiri, H. Smei, Task Migration in Embedded Systems: Design and Performance. Embedded Computing Systems: Applications, Optimization, and Advanced Design: Applications, Optimization, and Advanced Design, 2013.

[9] Kamel Smiri, Abdelhafidh Ben Fadhel, Abderrazak Jemai and Ahmed Chiheb Ammari, Automatic Migration of a Software Task to Hardware Component in MPSoC Systems, David publishing, Computer Technology and Application, 2011.

[10] R. Lysecky and F. Vahid, "A configurable logic architecture for dynamic hardware/software partitioning," in Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE '04), vol. 1, pp. 480–485, Paris, France, 2004.

[11] K. Jozwik, Shinya Honda, Masato Edahiro, Hiroyuki Tomiyama, and Hiroaki Takada. 2013. Rainbow: an operating system for software-hardware multitasking on dynamically partially reconfigurable FPGAs. Int. J. Reconfig. Comput. 2013.

[12] M. Wang, Y Han, R Wang, X Liu and J Sun., A software-Hardware Cooperative Method for Multi-projector Seamless tiled display system, - IEICE Electronics Express, 2015.

[13] Andy D.Pimentel, 2004. « Computer Systems: Architecture, Modelling, and Simulation », BOOK, THIRD AND FOURTH INTERNATIONAL WORKSHOPS, SAMOS 2003 AND SAMOS 2004, SAMOS, GREECE.

[14] Website SoCLib http://www.soclib.fr, Juin 2017.

[15] JPEG committee: 'Standardized in ISO/IEC IS 10918-1/2'. http://www.jpeg.org/., Juin 2017.

[16] K. Smiri, S. Bekri, H. Smei,"Fault-Tolerant in Embedded Systems (MPSoC): Performance estimation and dynamic migration tasks", in 11th International Design & Test Symposium, IDT 2016, Hammamet, Tunisia, December 18-20, 2016. IEEE 2016, ISBN 978-1-5090-4900-4 (IDT 2016), pp1-6, 2016.

[17] D. Sender Rocha dos, A. Santos, L. M. Jorge, Adaptive Intelligent Systems applied to two-wheeled robot and the effect of different terrains on performance, Advances in Science, Technology and .Engineering Systems Journal Volume: 2 Issue: 1 Pages: 1-5 Published: 2017.

[18] A.C. Ammari, A. Jemai, Multiprocessor platform-based design for multimedia, IET Computers & Digital Techniques, Volume 3, Issue 1, January 2009, p. 52 – 61, ISSN 1751-8601, 2009.

[19] Sima Afsharpour, Ahmad Patooghy, Mahdi Fazeli, Performance/energy aware task migration algorithm for many-core chips, IET Computers & Digital Techniques, 10.1049/iet-cdt.2015.0131, Print ISSN 1751-8601, Online ISSN 1751-861, Volume 10, Issue 4, pp. 165 – 173, 2016.

[20] Adel A. Elbaset, Hamdi Ali, Montaser Abd-El Sattar, Mahmoud Khaled, Implementation of a modified perturb and observe maximum power point tracking algorithm for photovoltaic system using an embedded microcontroller, IET Renewable Power Generation, Volume 10, Issue 4, p. 551 – 560, 2016.