Advances in Science, Technology and Engineering Systems Journal Vol. 2, No. 3, 1188-1195 (2017)

www.astesj.com

Special Issue on Recent Advances in Engineering Systems

ASTESJ

ISSN: 2415-6698

Fault-Tolerant in Embedded Systems (MPSoC): Performance Estimation and Dynamic Migration Task

Kamel Smiri* [1, 3], Habib Smei [1, 2], Nourhen Fourati [1, 3], Abderrazak Jemai [1, 4]

[1]*Université de Tunis El Manar, Faculté des Sciences de Tunis, Laboratoire LIP2, 2092, Tunis, Tunisie*

[2]*Direction Générale des Etudes Technologiques, Institut Supérieur des Etudes Technologiques de Rades, Rades, Tunisie*

[3]*Université de Manouba, Institut Supérieur des Arts Multimédias Manouba, Campus Universitaire Manouba, 2010, Tunisie*

[4]*Université de Carthage, INSAT, B.P. 676, 1080 Tunis, Cedex, Tunisie*

A R T I C L E I N F O

Article history:
Received: 09 April, 2017
Accepted: 24 June, 2017
Online: 20 July, 2017

Keywords:
Hybrid Model
Fault tolerance
Dynamic migration Task
Multiprocessor Systems-on-Chip (MPSoC)

A B S T R A C T

Multiprocessor Systems-on-Chip (MPSoC) allow the implementation of heterogeneous architectures with a high integration capacity. In recent years, computational requirements MPSoC are increasing exponentially. This complexity, coupled with constantly evolving specifications, has forced designers to consider intrinsically flexible implementations. Deploying applications typical of multimedia domains is difficult, not only due to the heterogeneous parallelism in the platforms, but also due to the performance constraints that typify these systems. An application can be modeled as a set of cooperative tasks. A task can be implemented in software or in hardware depending on its complexity and the involved cost. Our proposal is a fault tolerance approach which combines the results of a performance model and a technical's fault tolerance. We interest of the dynamic migration task to resolve the Fault-Tolerant for Multiprocessors Embedded System. We exploited an example of multimedia application (MJPEG decoder) to find optimal Fault tolerance systems. Our aim in this paper is to exploit the classic technique of fault tolerance. The solution chosen is the transformation of software processing into hardware processing. And also, exploitation of hybrid models (simulation/analytics). The goal is to have a Fault Tolerant Embedded System.

1. Introduction

This paper is an extension of work originally presented in 11th International Design & Test Symposium, IDT 2016 [1].

Multiprocessors System-on-Chip (MPSoC) is a system that contains several heterogeneous processors interconnected around advanced communication architecture as a Network on Chip (NoC). Nowadays, Multimedia applications use MPSoC architectures in order to achieve high performance. These architectures may include different types of computation units (DSP, Microcontroller...) and use different diagrams of communication.

We distinguish in the literature [2,4] three design approaches for MPSoC: (1) platform based design, (2) components based design and (3) synthesis system. In this paper we are going to focus on the first approach that targets the implementation of an application on configurable prototype architecture.

*Corresponding Author: Kamel Smiri, Université de Tunis El Manar, Faculté des Sciences de Tunis, Laboratoire LIP2, 2092, Tunis, Tunisie
Email: smiri_kamel@yahoo.fr

We follow along our work the MPSoC hardware/software design flow proposed by [5]. The software and the hardware are then gradually refined and simulated on four levels of abstraction. These levels are System Level, Virtual Architecture, Transaction Accurate and Cycle Accurate. System Level (SL) is the highest level of design flow. It provides an environment for parallel execution. The application is described as a set of processes that exchange data only through the FIFO blocking point to point. Virtual Architecture (VA) level is more detailed than the standard SL. The major difference is that it includes information on the architecture. For abstracting the underlying hardware architecture, the code of tasks using APIs known as HdS (Hardware dependent Software). Primitives of these APIs (ex. Send/Receive) access to explicit communication components. Transaction Accurate (TA) level is the second intermediate level. In this level, the tasks are linked to an Operating System (OS) and a library of communication to implement the communication and management services tasks. This software layer uses primitives of Hardware Abstraction Layers (HAL). The data transfer uses explicit addresses. The hardware platform is more detailed, including a communications network, the devices accessed by HAL-API, and an execution model of abstract processor. Cycle

Accurate (CA) level is the lowest level. At this level, the HAP-API is implemented. This implementation is done by adding a HAL software layer and a processor for each sub-system (for example, we create an implementation and a model for ARM, MIPS, and SPARC...).

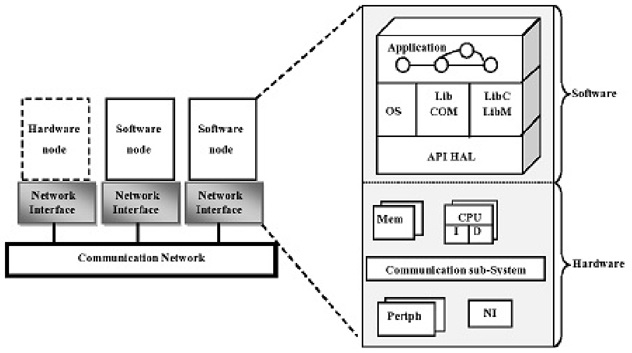

Figure 1 Typical architecture of MPSoC systems

Figure 1 present a typical architecture of MPSoC which is composed by software and hardware nodes connected by a communication network. Hardware nodes are component which does not have the ability of programming. A communication network connects all the nodes together. Software nodes provide the execution environment for applications tasks.

The software node is structured into several layers:

- Application Layer, this layer represents a set of tasks of the application which are carried out in parallel, in order to profit to the maximum of the parallelism offered by architecture multiprocessors. This layer communicates with the software node by calling on primitives of the operating system (OS).
- The Operating System OS, the operating system makes it possible to provide an interface between hardware equipment and the application.
- Libraries, the objective to use the library of communication is to call on the channels of communication and to use the various features of the library C standard and the library of mathematics.
- Hardware Abstraction Layer, this layer makes it possible the operating system to interact with the material peripherals on a level abstract rather than on a detailed material level.

The motivation of the paper is to explore the design space (Design for Space Exploration DSE) by early in the design cycle. We present in this paper a methodology for the migration of a software task in hardware component of a multimedia application at Transaction Accurate level and at Cycle Accurate level. The chosen application to illustrate our approach is the MJPEG decoder that is characterized by its high treatment density and important data exchange. This paper is organized as follows. Section 2 presents the related work. Section 3 gives an overview of Fault-Tolerant in Embedded Systems (MPSoC) approach. Section 4 presents the experimentation of MJPEG Decoder, with the objective to show the efficiency of our approach.

We propose in this paper an extension of the paper conference (IDT 2016) [1]. Mainly, we have added the following parts: a) An advanced synthesis on related works, b) A detailed description of our dynamic migration approach (based on Fault-Tolerant in Embedded Systems (MPSoC)), and c) A series of experiments to illustrate the effectiveness of our approach. We have added some experimentation that shows the effectiveness of our approach. The MJPEG decoder (which is a refresh application) is chosen to get me the dynamic migration efficiency. The IDCT task is chosen to show the gain during dynamic migration of the software to the hardware.

2. Related works

Due to the complexity of multiprocessor systems, the probability of failure is all the more important that it requires special consideration. Indeed, during a failure, a part of the application state disappears and the application may pass in an inconsistent state that prevents it from continuing normal execution. To perform effectively and properly, it must withstand stresses related to the execution environment. It must be able to tolerate the failure of one or more components. Fault tolerance is the property that keeps the smooth functioning and continuity of the system during a failure. So the intensity of the fault is not proportional to the decrease of the system load operation since found contents a very small mistake can cause a total blockage in a system designed.

There are many causes for failures in distributed systems. In [2], the author focused on solving crash failures in GRID computing, their proposal aims to ease the process of recovery when system failures are detected at runtime avoiding the necessity for application restarts. Their proposal works through a set of services that performs transparent task migration over the computing nodes, hiding the complexity related with error handling when a hybrid programming model based on Open MPI and OpenCL is employed.

In [3], the author proposes scheduling algorithm of the adaptive fault tolerant tasks which is a combination of two fault tolerant algorithm TMR (Triple Modular Redundancy) and DMR (Dual Modular Redundancy) and it also benefits from EDF and LLF task scheduling algorithms for decreasing miss rate of tasks. The main goal of these algorithms is to find a proper tasks assignment on the cores and tolerate processing component failures which could either be homogeneous or heterogeneous, so this algorithm is more perform than DMR and TMR methods in average 35% which can guarantee the proper execution of running tasks and reduce the total time of task execution.

In [4], the authors proposes an offline task remapping technique to minimize the communication energy and task migration overhead of an application mapped on a heterogeneous multiprocessor system for all processor fault-scenarios. This technique involves two steps: the first is communication Energy driven Design Space Exploration (CDSE) to select an initial mapping and the second is communication energy and Migration overhead aware Task Mapping (CMTM) for different fault-scenarios. The results show that the proposed CDSE reduces space exploration time of Conception 99% and the reduced energy CMTM 35% average communication and migration over by an average of 20% for all scenarios of single and double faults compared to tolerance techniques existing faults.

In [6], the authors proposes a design flow to explore mapping strategies and shape of the NoC network to improve synchronization performance and the power consumption, then for each step of the flow they used a heterogeneous platform (PC and

FPGA) to fulfill them. The exploration flow is provided to help designers NoC to choose a technique appropriate mapping tasks on a NoC appropriate form for a particular application. The experiments demonstrate that the most appropriate task mapping strategy and the most suitable NoC shape strongly depend on the algorithm used. Depending on the timing latency results obtained and the FPGA resources used, the designer can select the appropriate task mapping strategy on the suitable shape in a short exploration time and with precise timing evaluation.

In [5], the author proposed a methodology to master the migration process at transaction level and they demonstrate the various levels of the design process to deals with the impact of task migration as an alternative to meet performance constraints in the design flow. And, they use a SDF3 tools to provide performance estimation at transaction level. Using the SDF3 tool, the authors model a multimedia application using SDF graphs. Secondly, they target an MPSoC platform. The authors take a performance constraint to achieve 25 frames per second.

Table 1: synthesis works of fault tolerant

	[3]	[4]	[5]	[6]	[7]	[8]
GC	++	- -	- -	- -	- -	--
MPs	- -	++	++	++	++	++
AM	- -	- -	- -	- -	++	--
SM	- -	- -	- -	- -	++	--
OS	- -	++	+	+	++	+
SMt	++	- -	++	++	++	--
DMt	++	- -	++	++	++	++
IPb	- -	- -	- -	++	++	--
FT	++	++	++	++	- -	+

GC: Grid computing   SMt: Static Migration task

AM: Analytic Model   DMt: Dynamique Migration task

SM: Simulation Model   IPb: IP blocks

OS: Operating Systems, FT: Fault-Tolerant, MPs:MPSoC systems

In [7], the team of Tahir maqsood are conducted a detailed quantitative evaluation of the selected dynamic task mapping algorithms for NoC based MPSoCs for a wide range of mesh sizes with varying task and communication loads. Comparisons are conducted with varying network load, number of tasks, and network size for constantly running applications. Moreover, they propose an extension to communication-aware packing based nearest neighbor (CPNN) algorithm that attempts to reduce communication overhead among the interdependent tasks. The results indicate that proposed mapping algorithm reduces communication cost, average hop count, and end-to-end latency as compared to CPNN especially for large mesh NoCs. And the proposed scheme achieves up to 6% energy savings for smaller mesh NoCs. Finally, results of formal modeling indicate that proposed model is workable and operates according to specifications.

Synchronous Data Flow SDF is a particular case of the data flow and Petri network. Formalism SDF is often used for the entire predictable systems, in which the number of samples of data consumed and produced by each node with each execution, is specified a priori. Graphs SDF constitute the central element of modeling by SDF3 [8]. Moreover, SDF3 tool integrates several commands, for each one has a precise function. Indeed, the command sdf3flow is most interesting of tool SDF3 since it makes it possible to deploy an application graph with a constraint on architecture graph [9].

System on Chip Library (SoCLib) is an open platform for the virtual prototyping of MPSoC systems. SoCLib is a tool based on the simulation which it allows the development of platforms of virtual prototyping and facilitates the exploration of architectures by means of software tools to conceive embedded applications and of library Open Source is made up of models of high level simulation written in SystemC for the material components, which allow fast simulations [10].

Following our synthesis established in Table 1, we propose to use the contributions proposed in the paper (4) and paper (5), which we will be useful to propose a new fault tolerance approach based on dynamic migration for Embedded Systems.

3. Fault Tolerance Approach for Embedded Systems

Fault tolerance approach for embedded systems multiprocessor (Multiprocessor System on Chip MPSoC), combines the results of a hybrid model (proposed by LIP2 laboratory, in [6]) and a technical fault tolerance that is based on the dynamic migration. This approach provides an effective solution to the current problems of modern embedded systems.

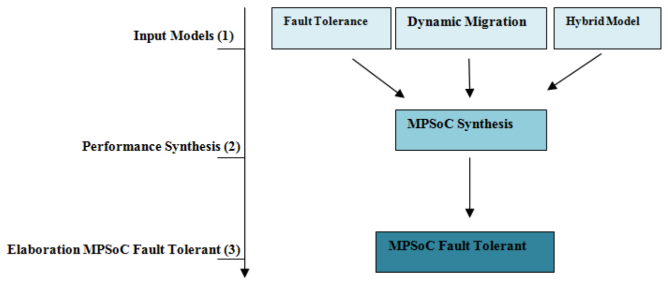

Figure 2 Fault tolerance approach for Embedded Systems

Our Fault tolerance approach for Embedded Systems is structured in three steps: 1) implementation of input models (Hubrid model, Dynamic migration technique, Fault tolerance technique), 2) performance synthesis and 3) Elaboration MPSoC Fault Tolerant.

3.1. First step: Input Models

The first step shows the essentials components of our approach citing hybrid model and fault tolerance techniques based on dynamic migration. The hybrid model (MH) resulting hybridization of two performance estimation methods: Analytical Method (MA) and Method based on simulation (MS). This model allows the exchange of information between the two methods MA and MS through an intermediate bonding layer whose goal is to treat various types of systems and reduce the time estimation and evaluation of performance and a low error rate. So, the aims of the MH model, is to combine the strengths of both MA and MS methods.

Note that the analytical method is characterized by the speed of optimal solution. It is based on the exploitation of SDF graph, this formalism to model the software applications, hardware and software requirements for deploying tasks on hardware resources. The analytical model used in this approach is a synchronous data stream which is a particular case of the data stream (Data Flow) and Petri network, to generate the mapping of the software application on the hardware platform, not only but also the usage rate of hardware resource which can be regarded as performance indicators.

The simulation-based method is characterized by the precision of the results. So in this model the designer must perform three tasks: Define the structure of the software application, Set the hardware architecture and Check the deployment of the software application on the hardware platform. Concerning technical fault tolerance that is based on dynamic migration, we propose to use migration software / hardware in a multiprocessor architecture, as a solution for fault tolerance.

3.2. Second step: performance synthesis

For the second step is the synthesis step we will make a complete architectural exploration of which will choose a multiprocessor hardware architecture by specifying these settings, A software application and we will follow the simulation and implementation of this application on the architecture while retaining control over the brutal mistakes that makes system failures.

3.3. Tree step: Elaboration MPSoC Fault Tolerant

The last step of our approach that will get a MPSoC fault tolerant. This step involves a component fault monitor CCP noted that the state's objectives of monitoring the system running. This hardware component in two missions which should take care of: detecting hardware failures and handling. CCP's interest is to eliminate any dependency on the rest of the system, consequently control failures will be independent of the processing part or in cases of failures remains functional and it keeps the communication is with the rest of system or with the outside (network). At the end of our proposed approach to fault tolerance in embedded systems multiprocessor allows to restructure the system into two subsystems: the first subsystem dedicated to the execution of the application and the second a dedicated subsystem fault tolerance.

Migration is the solution for fault tolerant problem in MPSoC. The dynamic migration is structured on four steps: (1) modeling software by KPN; (2) mapping and execution; (3) performance evaluation migration.

- *Modeling software by KPN*

First of all, we have modeled a software application on a target multiprocessor architecture. The application is a set of tasks interconnected by communication channels (FIFO) each architecture used contain one or more hardware accelerators where migration can be activated for critical tasks. And, we also fix the performance constraints that must be satisfied.

- *Mapping and execution*

In the second step, we make mapping of the software part on the hardware platform that contains the hardware component. We rely on profiling results of the application to detect the task requiring migration. This task will be transformed into hardware coprocessor to accelerate the processing of the complete system.

- *Performance evaluation of Migration*

To migrate a task from the software to a hardware accelerator, we propose three steps that need to be gone through: (1) Hardware description, it is to design a hardware component and integer throughout the MPSoC system; (2) The instantiation of the task as a hardware task; (3) The deployment of the task as hardware task.

When you decide to execute hardware a task, we start the backup from its current state including the control and the task selected in the shared memory information. Then the task migrated will be deleted or suspended on the source tile.

4. Experimentation: MJPEG Decoder

The figure 3 present Our architecture is formed by two processors ARM, a coprocessor for the TG task, a coprocessor for the RAMDAC task and a coprocessor for the migrate task.

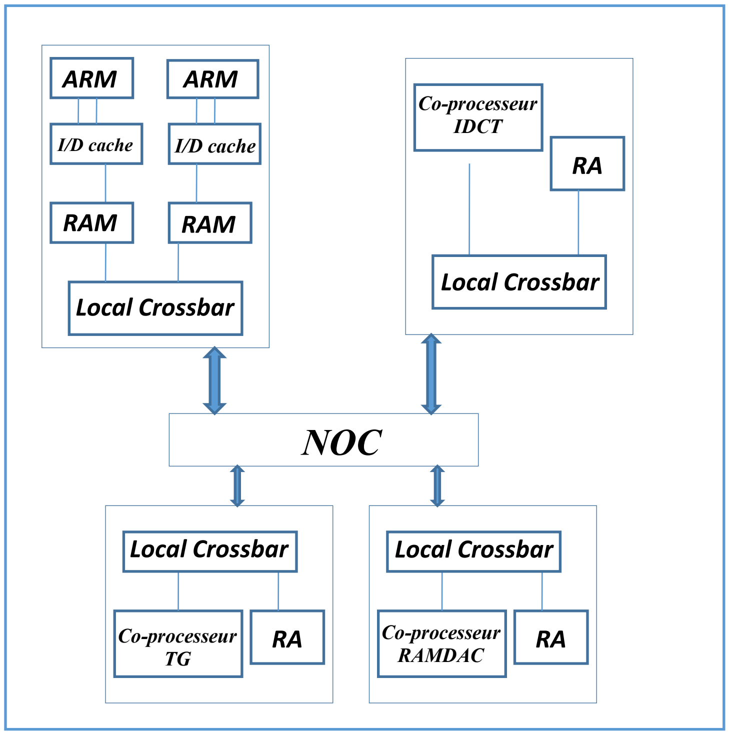

Figure 3 Architecture Hardware Platform for Embedded systems

One has validated the Fault Tolerance Approach for Embedded Systems using a Motion JPEG decoder. One wants to find an implementation of the MOTION JPEG decoder realising 25 frames per second (fps) as a functional constraint and using 50MHz processors as a non-functional constraint. The MOTION JPEG decoder reads a stream of JPEG images from an input peripheral: a traffic generator named TG and writes pixels on an output peripheral: a digital-to-analogue converter named RAMDAC. The first functional bloc DEMUX dispatches the input stream to the other blocs. The decoding chain (Fig. 5.a) begins with the decompression bloc (VLD), then the zigzag rearrangement bloc (ZZ), followed by the inverse quantification bloc (IQ). Finally, the stream passes to the inverse discrete cosine transformation (IDCT).

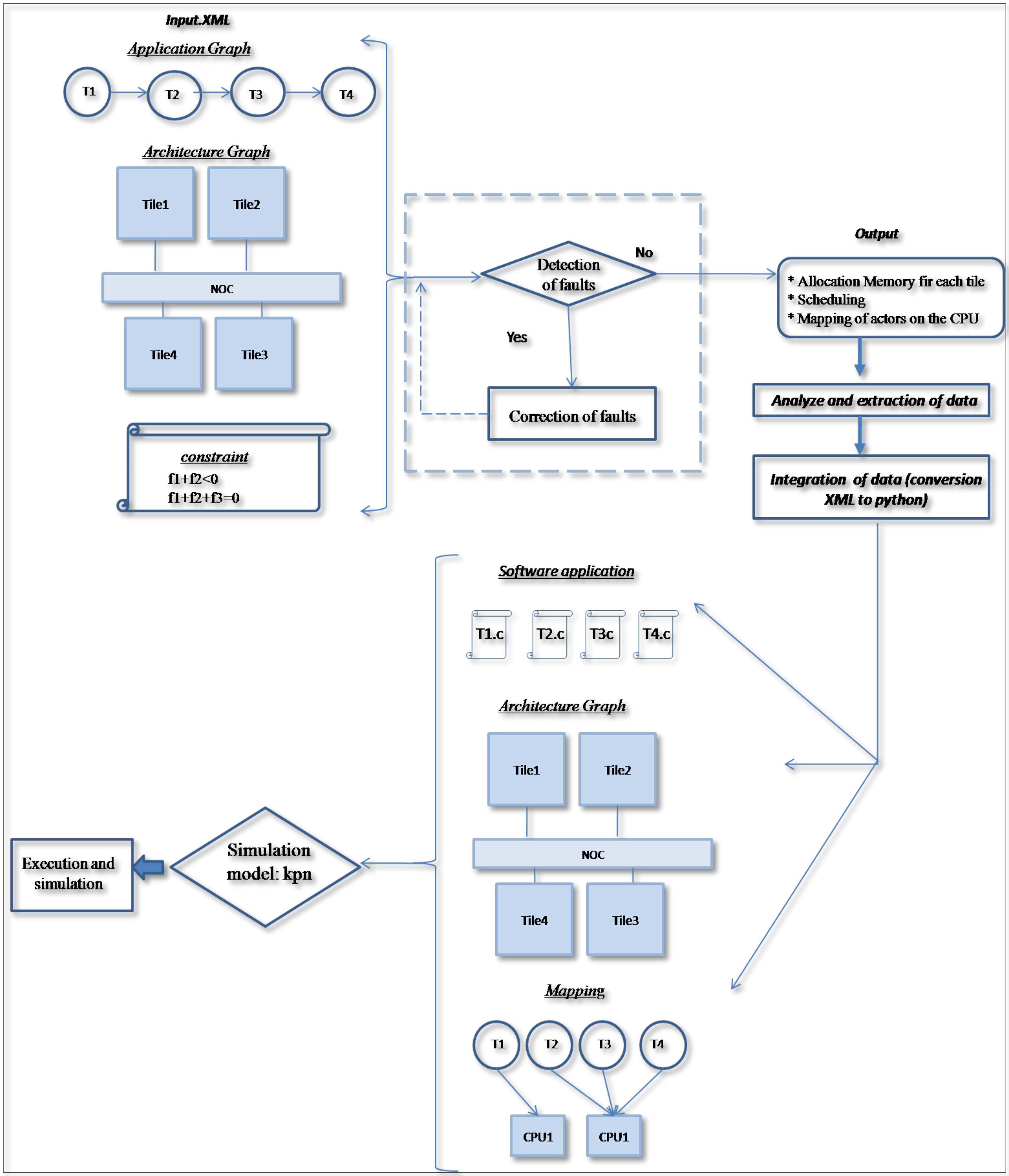

Figure 4 Fault Tolerance Approach for Embedded Systems

One has validated the Fault Tolerance Approach for Embedded Systems using a Motion JPEG decoder. One wants to find an implementation of the MOTION JPEG decoder realising 25 frames per second (fps) as a functional constraint and using 50MHz processors as a non-functional constraint. The MOTION JPEG decoder reads a stream of JPEG images from an input peripheral: a traffic generator named TG and writes pixels on an output peripheral: a digital-to-analogue converter named RAMDAC. The first functional bloc DEMUX dispatches the input stream to the other blocs. The decoding chain (Fig. 5.a) begins with the decompression bloc (VLD), then the zigzag rearrangement bloc (ZZ), followed by the inverse quantification bloc (IQ). Finally, the stream passes to the inverse discrete cosine transformation (IDCT).

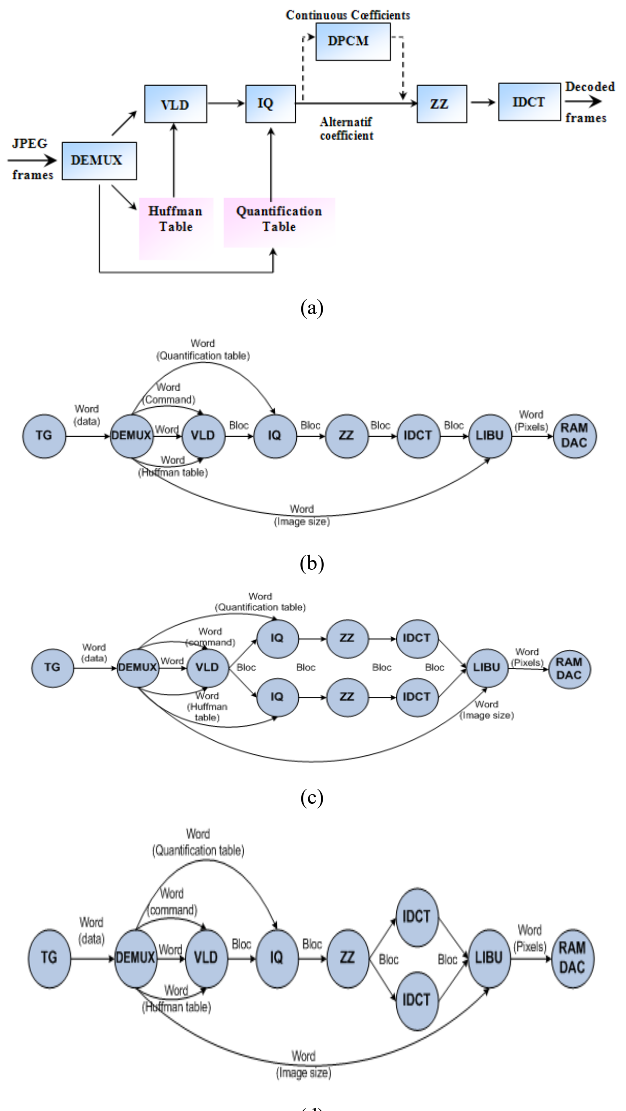

Figure 5: (a) Motion JPEG decoding principle, (b) model of Motion JPEG decoder, (c) 1st parallelization solution, (d) 2nd parallelization solution

Based itself on migration diagram presented in Fig. 4, we transform the software task IDCT to hardware component. The validation of hardware IDCT is carried out in two stages. The first is the validation of behavioral IDCT; by comparing the results of the implementation software IDCT in C with the equipment in SystemC. The second step is the validation of the communication interface that interconnects IDCT block with the rest of MPSoC system. The integration is made by adding a software/hardware channel between VLD-IDCT and a hardware/software channel between IDCT-LIBU. We limit ourselves to adapt to existing protocols KPN channels to have a coherent refinement. We have structured hardware IDCT component on three layers: (1) an adapter for easy connection to the rest of MPSoC system, (2) the interface module that manages the exchange of data between SystemC module and the rest of system, and (3) the true behavior of hardware component. The interface module is composed of two SystemC modules *Write* and *Read*, and two FIFO with finite sizes (64 KBytes). *Write* module written in the first FIFO data from the VLD. *Read* module permits to read the data processed by IDCT and sends it to the LIBU. Primitives wait (t_{Write}), wait (t_{Write}) and wait (t_{IDCT}) allow modeling of execution time modules respectively *Write*, *Read*, *IDCT*.

For aim of determining the timing of hardware IDCT and functions Write and Read on target technology (Xilinx, Altera ...), we explored GAUT tool, which is among architectural synthesis tools under constraints. For a cadence = 100ns and period = 10ns, the execution time of the IDCT is t_{IDCT} = 100ns for the treatment of Macro-Block (8 × 8 pixels). We estimate that the execution time of each module Write and Read are equal and is $t\_{Write} = t\_{Read}$ = 10ns.

2) Performance migration at CA level

The architecture that we synthesize at Cycle Accurate (CA) level is relative to the constraints laid down at the beginning of the problem. The first constraint is the functional constraint on the decoding speed of 25 frames per second (25 fps). The second constraint is the non-functional constraint on the platform that uses MIPS processors running at 50 MHz. The Operating System *Mutek* is a multiprocessor multithread kernel supporting POSIX APIs. This *Mutek* kernel proposes FIFO based communication layer allowing KPN communications. It includes three different schedulers. In the Symmetric Multiprocessor (SMP) kernel, there is only one scheduler allowing the KPN processes to migrate between processors. The Asymmetric centralized Scheduler (Non_SMP_CS) is only one scheduler affecting statically processes to processors. The Asymmetric distributed scheduler (Non_SMP_D) instantiates one scheduler to every processor and the processes are statically affected to processors. The design space exploration synthesis of kernel scheduling policies and software/hardware mapping are presented in Table 1. This table summarizes the performance of implementation of the MJPEG decoder in number of frames per second (fps).

Table 2: Performance implementation of MJPEG decoder (fps)

proc	SMP-CS				Non SMP-CS		SMP - CS		
	Seq	KPN	ZZ + IQ + IDCT	Double IDCT	Double IDCT	ZZ + IQ + IDCT	IDCT hard	IDCT+VLD hard	IDCT+VLD+IQ hard
1	12,2	6,72	---	6,78	---	---	7,7	17	24,12
2	12,61	12,01	12,16	12,86	---	12,68	11,49	24,07	32,45
3	12,63	14,1	17,65	18,63	20,49	---	11,55	36,73	32,56
4	12,6	13,86	22,35	22,09	25,53	---	---	---	---

We decide to transform IDCT task in hardware component, which allows us to achieve a speed decoding of 24.07 fps with 2 MIPS processors and 36.73 fps with 3 MIPS processors on a simulation platform. The optimal architecture will use 2 MIPS processors configured with 4 KB memory cache, RAM memory, Pi-Bus and an interruption controller. The 2 MIPS processors are dedicated to execute DEMUX, VLD, IQ, ZZ and LIBU processes with an SMP scheduling policy. In addition, an input peripheral is needed for reading video stream TG and an output peripheral for display RAMDAC. For achieving the 25 fps functional constraint, hardware implementation of the IDCT processes is necessary. If the performance is meadows of 25 fps (for example 24 fps), one accepts the solution under condition to optimize the codes of application until to get the performance of 25 fps.

3) Performance migration at TA level

The modeling of the IDCT in hardware, adding time performance metrics (t_{Write}, t_{Read} and t_{IDCT}) and the integration of the component with the rest of system, allowed us to perform simulations for three different input images (see Fig. 6.). We note that the best performance is obtained for a configuration of two CPU and hardware IDCT. This configuration agrees well with the results obtained at CA level.

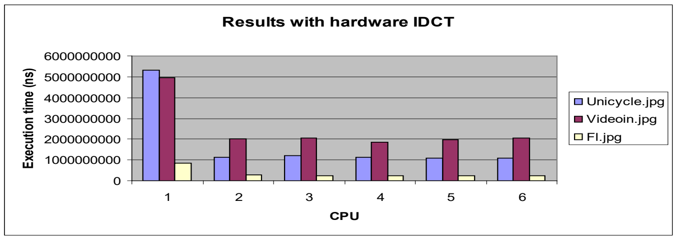

Figure 6 Comparison of simulation Results with IDCT hardware for three different images

The first stage of our approach is analytical modeling. The latter provide the estimation performance in earlier stage of MPSoC design. In our methodology, analytical modeling is based on the exploitation of graph SDF. This formalism makes it possible to model an application graph, an architecture graph and, it allows to model the constraints (we interested in the throughput constraint as an indicator of performance) under which the deployment will be run (mapping the software tasks on the processors) via SDF3 tool. In order to calculate the throughput constraint, we multiply the number of frames per second by a cycle time.

Throughput = 1/(frequency)(number of frames)*

The first stage of the diagram is the modeling of the application on a target platform at Transaction Accurate level. The Fig. 7.a presents the structure in layers of the software part and the Fig. 7.b shows an example of hardware architecture of a target platform. The application is modeled via the Khan Process Network (KPN) at TA level. The software stack is composed of a set of tasks, HdS-API, Operating System (OS) and HAL-API. Communications at this abstraction level is done through specific addresses by the primitive types *Read()* and *Write()*.

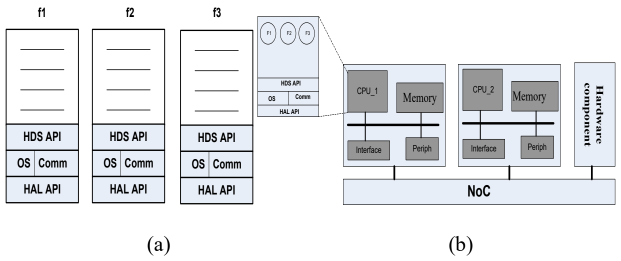

Figure 7 Transaction Accurate level: (a) software layers, (b) model of the target platform

We are going to try in this section to validate our hybrid model of estimation of the performances for the design of the embedded systems. We begin with the first part which is the analytical modeling and the data, which will be used by this model, can be obtained via tools of profiling either from the traces of a tool of simulation (we choose this method).

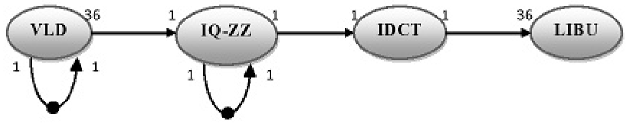

Figure 8 Graph of the MJPEG decoder application

The figure 8 represents the SDF model of the decoder MJPEG which is used during the estimations of performance. The actor VLD produces 36 tokens which correspond to 36 macros block of the image "plane.jpg" with a size of 48 pixels by 48. These are then transmitted to the actor IQ-ZZ, who reorganizes pixels according to the coding inverse zigzag then transmits it to the actor IQ-ZZ, who realizes the inverse quantification. The macro block is then treated by the actor IDCT who makes the inverse discreet transformed in cosine. Finally, the actor LIBU handles the pixels of the image to adapt them to the ring peripheral of exit, it is for it, he requires 36 macro blocks to reconstitute the whole image. Auto-arcs on knots VLD and IQ-ZZ represents respectively the tables of Huffman and quantification auto-passed on executions after execution. Once the actors are modeled, it is necessary to determine the requirements in terms for memory and the execution times of the actors of the graph. These properties are also described in an XML file. This code makes it possible to carry out actor IQZZ on the processor mips0_0 with 174763 cycles like an execution time for this task.

We add the attribute name for the element memory to facilitate the assignment or the allowance of a segment of memory for each actor in the SoCLib simulation tool. The latest information needed for the application description in its XML file is the throughput constraint which must be respectful when the decoder MJPEG mapped on the target platform. This following code presents how can define the throughput constraint in description file.

```
<timeConstraints>
    <throughput>0.0000005</throughput>
</ timeConstraints>
```

In this pseudo code, the objective of constraint is to reach a rate of 25 frames per second. So, the Application must be performed 25 iterations per second and the processor being clocked at 50 MHz (the time unit = 0.2 nanoseconds).

5. Conclusion

In this paper, one presented the technique of Fault-Tolerant in Embedded Systems (MPSoC). We interest of the dynamic migration task to resolve the Fault-Tolerant for Multiprocessors Embedded System. Multiprocessor Systems-on-Chip (MPSoC) allow the implementation of heterogeneous architectures with a high integration capacity. In recent years, computational requirements MPSoC are increasing exponentially. This complexity, coupled with constantly evolving specifications, has forced designers to consider intrinsically flexible implementations. Deploying applications typical of multimedia domains is difficult, not only due to the heterogeneous parallelism in the platforms, but also due to the performance constraints that typify these systems. An application can be modeled as a set of cooperative tasks. A task can be implemented in software or in hardware depending on its complexity and the involved cost. In this paper, our proposal is a fault tolerance approach which combines the results of a performance model and a technical's fault tolerance. Mainly our approach deals with three techniques: 1) Performance estimation in embedded systems, 2) Exploitation of hybrid models (which are based on technical simulation and analytical model), 3) Fault tolerance (based on duplication of software/hardware processing).

This approach has been tested for a Motion JPEG case study, to find out an optimal hardware and software implementation. However, the actual design of MPSoC Fault-Tolerant lacks for a high level performance evaluation model and a technical's fault tolerance. We propose a Fault-Tolerant approach based on the use of three techniques: 1) performance estimation, 2) exploitation of hybrid models (simulation / analytics) and 3) fault tolerances. The goal is to have a Fault Tolerant Embedded System. The limit of our approach is that it will only be applied in the MPSoC design approaches platform based design.

References

[1] 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.

[2] 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

[3] S. Shiravi, M. E. Salehi, Fault tolerant task scheduling algorithm for multicore systems. In Electrical Engineering (ICEE), 2014 22nd Iranian Conference on (pp. 885-890). IEEE.

[4] A. Das, A. Kumar, B. Veeravalli, Communication and migration energy aware task mapping for reliable multiprocessor systems. Future Generation Computer Systems, 30, pp 216-228, 2014.

[5] 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.

[6] K. Pang, V. Fresse, S. Yao, O. A. De Lima, Task mapping and mesh topology exploration for an FPGA-based network on chip. Microprocessors and Microsystems, 39(3), pp 189-199, 2015.

[7] T. Maqsood, S. Ali, S. U. Malik, S. A Madani, Dynamic task mapping for Network-on-Chip based systems. Journal of Systems Architecture,61(7), pp 293-306, 2015.

[8] K. Smiri, A. Jemai, "NoC-MPSoC Performance Estimation with Synchronous Data Flow (SDF) Graphs", Autonomous and Intelligent Systems (AIS'2011), June 22-24, Burnaby, BC, Canada, pp. 406-415, 2011.

[9] Ghamarian A.H., Stuijk S., Basten T., Geilen MGW. And Theelen B.D., "Latency Minimization for Synchronous Data Flow Graphs", Published in Digital System Design Architectures, Methods and Tools, 2007.

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