Deliverable 4:

Technical report describing the methodology used by off-line functional/structural test

Editor: LIP6

Authors:
M. Benabdenbi
D. Refauvelet
F. Pêcheux

SUMMARY

Versions

V1.0 : 20010-03-14

1 Introduction

This document “Technical report describing the methodology used by off-line functional/structural test” is the fourth deliverable “D4” of the ADAM ANR project.

The main objective of this fourth deliverable is to define the global methodology required to reach Test & Diagnosis objectives in order to allow the construction of a Hardware Architecture Instant Map (AIM). This map of functional hardware components is then intended to be used as an entry of the functional remapping stage. The issues related to the construction of the AIM and to the remapping phase will not be discussed here; two dedicated deliverables are devoted to that.
This deliverable use as a basis the results detailed in the previous deliverables. This document refers to the current strategy of the different partners, thus ideas and principles presented here may slightly vary within the remaining time of the project.

This document includes 6 sections:
1) This introduction
2) A brief reminder of the ADAM project
3) An overview of the methodology for functional/structural Test & Diagnosis of the MP²SoC. The following sections detail the methodology.
4) A description of the main steps to be done after the chip power boot. These steps will lead to a coarse grain Test and Diagnosis of the chip subparts.
5) A description of the process leading to the election of a Master Test and Diagnosis Controller, this master being in charge to communicate with the outer world.
6) The software-based techniques allowing a fine detection/localization of faulty components.
7) A brief description of the intended reconfiguration/deactivation steps required before the remapping process.

2 Reminder of the ADAM project

ADAM is a prospective research project trying to provide solutions to the hot topic of self-adaptability of MP²SoC chips embedding applications running in presence of malfunctions such as loss of performances, hardware permanent failures or sudden temperature elevation. Architectures targeted by the ADAM project are MP²SoC containing functional homogeneous blocks (more than a thousand of computing units concurrently working on the execution of an application) but with heterogeneous behaviours (with different electrical and temporal characteristics from a block to the other). Taking into account at the same time all these aspects to allow optimized performances and high level reliability is a key issue for the next generations of embedded multi-processors systems.

As shown in Figure 1, the ADAM project can be seen as a “three stages rocket”: the online monitoring, the online test and diagnostic and the online remapping of the functional application. Actual researches address the issues related to each of these three stages but very few works yet neither propose global solutions nor specify the interfaces between the stages.

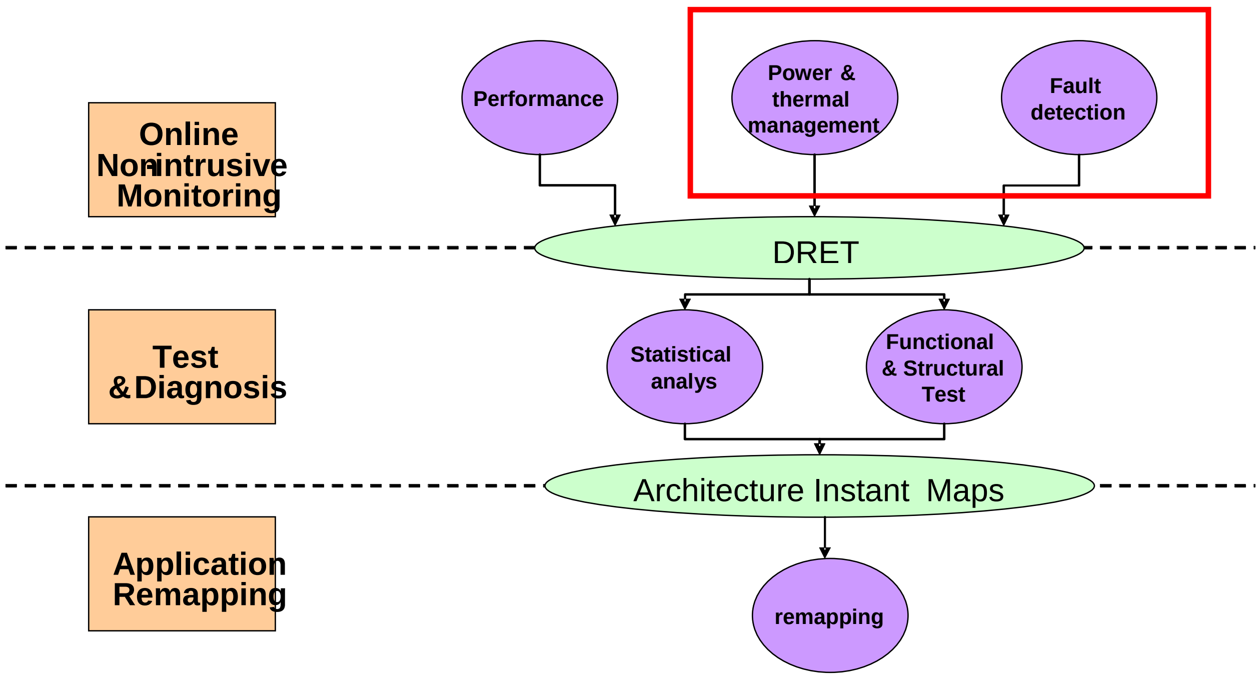
Online Non-intrusive Monitoring
Performance
Power & thermal management
Fault detection
DRET
Test & Diagnosis
Statistical analys
Functional & Structural Test
Architecture Instant Maps
Application Remapping
remapping

Figure 1: the three stages of the ADAM project.

Knowing that the hardware architectures of the three partners are basically different, the uniformity of the ADAM project comes from the willing of each partner to propose common standardized interfaces between the three stages (see figure 1). The two major results expected by the partners, that should interest the MP²SoC community, are then the data structures called DRET and AIM and their programming interface. The DRET (Distributed Raw Event Table) gives a level of abstraction allowing to manage in the same way events triggered in the system whatever the origin is, hardware or software. All the events are then processed in a software way in order to take the appropriate decision (modifying the system behaviour or even stopping the application execution).
The AIM (Architecture Instant Map) corresponds, after the diagnosis process, to the cartography of the system related to a given criteria (temperature map, map

of functional/defective hardware components, map of data volume exchanged between the embedded IPs…).

The two programming interfaces should be simple and efficient enough to let, in the last stage, the remapping application accesses easily these databases.

Several types of scenarios using these programming interfaces have already been clearly identified in the ADAM project, as shown in the deliverable 2.

2.1 Stage 1 : On-line Monitoring

The online monitoring consists in adding to the system hardware and software sensors (architecture + embedded functional application), in order to detect and collect events related to the appearance of phenomenon within the MP²SoC (unexpected temperature rise, power consumption or load of a processor over a certain threshold…). This monitoring must be non-intrusive and dedicated to reading and storing sensor values in the local memories, in order to take the appropriate decision for the system adaptation. The monitoring application can be an independent application or it can be directly embedded in the micro-kernel allowing the functional application to be executed.

2.2 Stage 2 : Diagnosis

Online diagnosis is the stage responsible for making thorough analysis, once events corresponding to alteration/malfunction have been detected in the previous stage. This stage interprets and formats the raw results, logs them into efficient data structures like databases and manages their history. Diagnosis also performs intrusive tests, like functional or structural tests on IPs, computes an annotated representation of the running architecture, and finally builds a database of audited architecture views. These views, or maps, represent an instant picture of the architecture showing the exact physical locations of the analyzed phenomenon occurrences. The diagnosis stage, according to the level of reactivity needed, may or may no stop the running application. For performance issues, it is not required to stop the application but in the case of the structural test of a component, the running application must be stopped and totally replaced by the test application. In other words, an event map actually represents the audited architecture with respect to the monitored event.

2.3 Stage 3: Application remapping, System adaptation

Online constrained application remapping exploits the database of event maps, and possibly its history, to determine how and under what conditions the application graph can be remapped on the architecture. The instant map is used to constrain the placement of the monitored application graph. Different placement strategies for the application graph are possible, from a centralized scheme that statically assigns threads to processors once for all to a distributed and dynamic placement algorithm that allows task migration/replication and local optimization.

The ADAM project addresses a major part of the issues related to MP^{2}SoC self-adaptability and aims at determining the common hardware and software mechanisms needed for the three stages. As a proof of concept, and to validate the whole work, 3 distinct applications will be mapped onto the three hardware architectures maintained by each partner: a telecom application 3GPP-LTE, a H264 decoder, and a mp3 decoder.

3 An Overview of the Test/Diagnosis/Reconfiguration Steps

3.1 The main goals

As stated in section 2.2, the main objective of the diagnosis stage is to build a cartography of faulty/functional hardware components.

3.2 The assumptions

The ADAM project focuses on the detection and diagnosis of hardware permanent faults. We do not intend to manage failures due to transient faults such as single event upsets (SEUs) which require some different detection/correction techniques out of the scope of the project.
Detecting and locating permanent faults require thorough onchip analysis and thus cannot be done online. The test and diagnosis (T&D) process is then intrusive and cannot be executed while the chip runs the functional application.
That's why in this study we propose an off-line methodology, which execution follows the online monitoring phase responsible for the stopping of the functional application after malfunction detection.
The T&D phase starts with the power boot of the MP2SoC. This process can be not only run at fabrication testing time but also all along the chip's life when embedded in its functional environment. This T&D process can be considered as a periodic in the field test/diagnosis/repair technique providing a kind of fault tolerance.
Two questions coming with the T&D phase are "what do we test ? and consequently what kind of faults do we target ?" This raises the problem of the granularity of the T&D objectives.
Another issue is that the chip embeds a limited amount of memory and is generally connected to an external memory with much more capacity.
To answer the both two last issues (granularity and limited embedded memory) we propose to split the T&D process in two successive steps: the coarse grain and the fine grain T&D phases.
The first one starting with the chip power boot will target the detection of a malfunction of a component as a whole (microprocessor, memory bank, peripheral, NoC router, …). Testing functionalities will come from the on-chip resources and are consequently limited. Details of this step are given in the next section.
The second one will start with the access to the external memory. As more complex and bigger test data/programs are available, the T&D process can be more accurate allowing localizing the faulty subparts of a component (a segment of a memory bank for example). What we propose to implement this step is detailed in section 6.
To reach the chip fault tolerance relying on the here described T&D phase, we choose to build our methodology on Software-Based-Self-Test techniques

(SBST). This mainly consists in executing, through the reuse of the embedded computing resources, software programs targeting the detection/localization of hardware faults. The main benefits, compared to dedicated hardware techniques, are the following:

- o Avoidance of a costly area overhead
- o Full scalability with the growth of the chip size
- o Fast test development
- o Adjustable fault coverage and accuracy
- o Requirement of a minimal test equipment (interesting since we intend to provide in the field testing)
- …

As we have two different goals, coarse grain and fine grain T&D objectives, we define two sets of SBST programs. The first set of programs must be very small since the embedded memory required to store them is expensive (mainly ROM and flash memory). These programs include all the testing steps from the chip booting to the moment when a microprocessor accesses the outer world.
The second set of programs is not size limited since it is stored in the external memory.
We assume also in this T&D phase that the Operating System, the boot loader and the functional application are not embedded in the chip but located in the external memory, and this for two main reasons: first because of their size, too big to be embedded, and second because of the potential degradation of the chip hardware through its lifetime. Indeed, in our approach the functional application remains the same while the chip hardware gracefully degrades. Thus, the functional application, must be, at each T&D execution, remapped on the functional hardware.
Moreover, as the T&D process can be executed many times, the chip must register, from one boot to the other, the previous functional hardware cartography, in order to skip the test and diagnosis of known faulty parts. In addition to the initial one, the current Architecture Instant Map (AIM) must then be stored in a flash memory each time it is computed.

Another issue guiding our thoughts is that as the T&D phase begins with the power boot, no assumption on what is functional and what is faulty can be done. For that reason, we propose for the coarse grain T&D step, an incremental and distributed approach (detailed also in the next section).
Finally, in this work, we do not make assumptions concerning the duration of T&D phase since this one is executed off line and depends on the test accuracy and quality targeted. However, the shorter, the better.

3.3 The targeted MP²SoC architecture

The target platform is a MPSoC (Multi-Processor Sytem on Chip). It is composed of clusters interconnected together thanks to a 2D mesh interconnection network (see figure 1). We implement here shared memory and NUMA schemes.

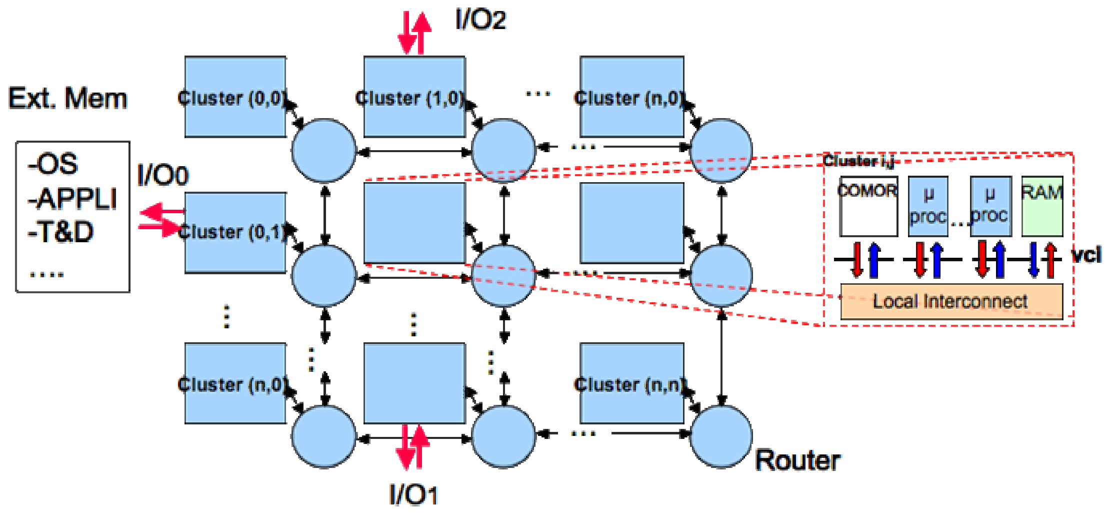

Figure 1. Targeted MPSoC architecture

Each cluster contains up to four processors, with their associated caches, one ram bank, one flash ram, containing data that are not erased in case of reset of the system. It also contains a Network Interface Controller (NIC), which is in charge of transferring request from/to the cluster.

Each cluster also contains a specific component, called configuration/monitor (COMOR), which is in charge of collecting data from the monitored hardware and modifying the configuration of the hardware. The software for both monitoring and configuration of the platform drives this component. Its purpose is not only to provide an interface between the monitoring application and the monitored hardware but also to allow the reconfiguration of the cluster component (including the router).

3.4 The required steps to reach the remapping phase

Our objective is to detect a hardware failure, localize the faulty component and then deactivate this one through an efficient configuration of the platform. When all these steps are correctly processed the functional application can be remapped on the known good remaining components.
The starting point of our methodology is the power boot of the platform and the ending point is the reconfiguration of the required components.
Here follows the required steps to fulfil the objectives:

- Power boot: quick incremental rough testing of the main components.
  At power boot we assume that we have no confidence on the reliability of the different clusters and of the NoC. Thus, concurrently and in a distributed way, starts the self-test of the clusters and the NoC. The first one is

processed thanks to embedded software while the second is implemented through hardware techniques (BIST).

- Election of a master Test/Diagnosis/Reconfiguration controller

Once the clusters and the NoC have ended their distributed self tests, a synchronization mechanism must be implemented in order to allow the starting of the fine grain test, the diagnosis and the fitting reconfiguration. For that, the external memory must be accessed by one and only known good processor to execute the corresponding codes. A voting algorithm is implemented and executed within the chip, leading to the election of the master controller responsible for the processing of the remaining steps up to the AIM generation.

- The Fine Grain Test and Diagnosis

The Master controller initiates a distributed fine grain test/diagnosis of the main clusters through the forwarding of the corresponding codes from the external memory bank to the main clusters. The diagnosis of the NoC components is also run leading to the detection and localization of faulty communication channels and/or routers.

- The reconfiguration of the platform.

Once locating the faulty components the corresponding action must be taken. For the faulty cores, they must be deactivated. For the NoC, not only the faulty routers must be deactivated but the surrounding routers must be reconfigured to avoid sending packets to the malfunctioning router.
Once the reconfiguration of the hardware is complete, the master controller can generate the cartography of functional hardware.
This AIM is finally used as an input of the remapping phase.

4 Booting time: Built-In Coarse Grain Test&Diagnosis

After booting the platform, two concurrent processes are run: the initialization procedure of the NoC and the concurrent self test of all clusters.

4.1 Distributed NoC initialization procedure

In a previous work [1], we proposed, a reconfigurable routing algorithm for a 2D-Mesh NoC. This reconfiguration procedure can be used as long as the faulty components (routers or point to point communication channels) have been identified. To support fault tolerance, we need to solve three main problems:
A The faulty router(s) and channels must be detected & de-activated by an appropriate built-in mechanism.
B A fault-tolerant, distributed, reconfigurable routing algorithm must be implemented in all routers in the NoC (proposed in [1]).
C A robust configuration bus must be implemented in the hardware to distribute the configuration information to the remaining (good) routers.

We believe that problem ***C*** can be solved by using the NoC itself: the remaining routers can be used as a test and configuration bus, through "flooding" broadcast algorithms.
However, these broadcast mechanisms can be disturbed by the destructive NoC malfunctions caused by the faulty routers, such as self-generating fake packets. Thus, we propose to address problem **A** thanks to a fully distributed initialization procedure, including detection and de-activation of both the faulty routers and the faulty communication channels. This procedure relies on a distributed, scalable, at-speed, built-in self test (BIST) hardware support, and is sytematically executed at power-on. This initialization procedure can be executed off-line when the chip is embedded in its functional environment. The fault coverage of this BIST has been evaluated using the Stuck-at fault model (SAF).

4.1.1 **The DSPIN NoC**

The DSPIN (NoC) [2], [3], (Distributed Scalable Predictable Interconnect Network) was designed by the LIP6 laboratory and was physically implemented by ST Microelectronics to support MPSoC architecture. As shown in Figure 1.A, the DSPIN router is composed of 5 modules (North, East, South, West & Local) interlinked as a full crossbar to adapt to the reconfigurable routing algorithm [1]. In order to support the GALS approach, the adjacent borders of two neighboring routers are connected by two FIFOs: one synchronous FIFO-out, one bi-synchronous [4] FIFO-in. These two FIFOs constitute a point to point communication channel (called a channel), as shown in Figure 1.B.

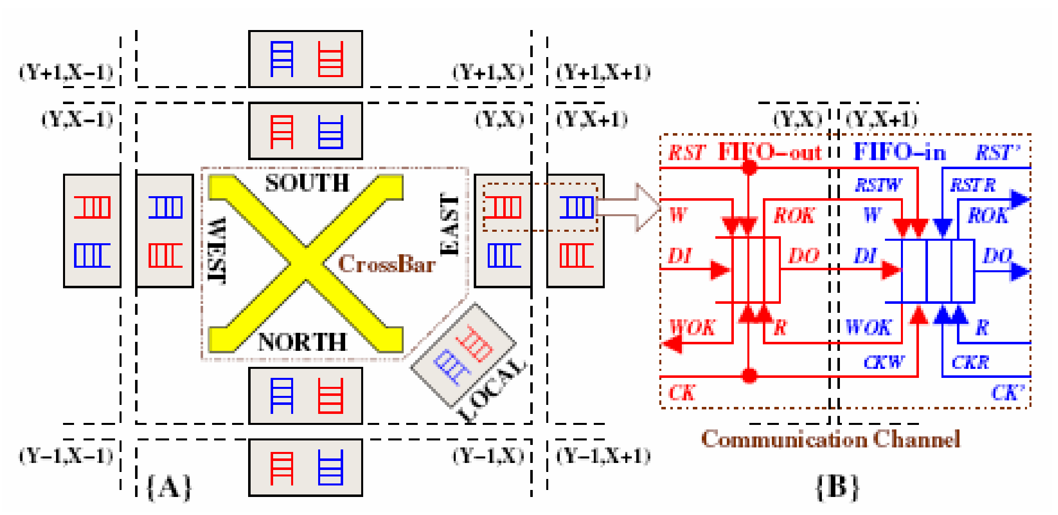

Figure 2 A generic DSPIN router and a generic communication channel

As most NoC designed for shared memory multi-processors architectures, DSPIN is a packet-switching network. There is actually two types of packets (command and response), and a packet is divided into smallest flow control units called flits. The first flit is the header flit that includes the destination address defined in absolute coordinates Y and X. The second and third flits contain the payload protocol informations. The remaining flits are payload dataflits. The trailer flit contains theendof packet (EOP) mark.
Besides thecrossbar itself, a router contains two types of hardware components:

- o Each input port implements a Routing Function (RF: combinational logic);
- o Each output port implements a Round-Robin Allocator (RRA: sequential logic).

The header flit of a packet is analyzed by the RF logic, and an output port is selected. Then the RRA builds a path from the input port to the selected output port, and the whole packet is transmitted to the target.

4.1.2 **Malfunction due to a faulty channel**

The DSPIN malfunctions caused by a faulty channel are analyzed. The Stuck at Fault (the fault model chosen here to mimic a permanent failure) can be injected on the flow-control signals (W, WOK, R, ROK) as well as on the data signals (DI, DO) as shown in Figure 2.{B}.
In this analysis, the point-to-point channel is modeled as a “Black Box”, comprising both the output FIFO(-out) in domain (y,x) and the input FIFO(-in) in domain (y,x+1).
From this analysis, not detailed here, we can conclude that most malfunctions in a single communication channel will prevent the reuse of the NoC itself for test and reconfiguration purpose. For example, in cases W/SA1, the self-generating fake packets will transmit a wrong configuration information to the good/remaining routers, or in the worst case, they will block the whole NoC.

	ADAM	4		2010-03-14	LIP6		1.0	12/22
	project	Deliverable		Date	Editor		Version	n°page

In order to avoid this destructive behavior, faulty routers and faulty communication channels must be de-activated as soon as they have been detected as faulty. This test and deactivation mechanism must be totally distributed, because it must be done locally, for each router and each communication channel, when the NoC is not yet running.

4.1.3 The distributed initialization procedure

The proposed initialization procedure is fully decentralized, and is implemented by a set of dedicated FSMs (Finite State Machine) located in each router. These FSMs are activated by the global RESET signal. There is two level of parallelism in this distributed algorithm:

- There is one set of FSMs in each router, and the boot procedure is executed in parallel in all routers.
- In each router, there is a master FSM (to test and boot the router itself), and several slave FSMs (to test and boot the communication channels).

As a communication channel is connecting two neighbor routers A & B, the slave FSM in router A must cooperate with the slave FSM in router B to test the channel (A-B).

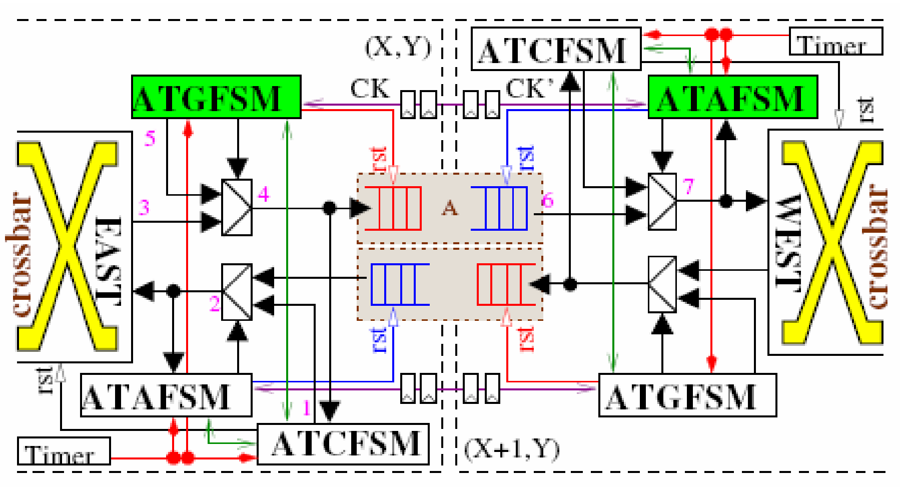

Figure 3 The initilization procedure implementation

The router (Routing Functions, Round-Robin Allocators, and the crossbar itself) is tested first, without any interaction with the neighbor routers. Then the communication channels associated to a router are tested in parallel. If the test of a router is KO, the router is considered as faulty, and all input and output channels are de-activated. If the test of a given channel is KO this channel is de-activated.

The master FSM is called ATC (Auto Test Center). The slave FSMs are called ATG (Auto Test Generator) and ATA (Auto Test Analyzer): one ATG FSM per output channel, one ATA FSM per input channel.

The ATC FSM works as a test pattern generator, and as a test pattern analyzer for the test of the router (internal crossbar and associated logic).

Each ATG FSM cooperates with the corresponding ATA FSM in the neighboring router to test the bi-synchronous channel. As these two FSMs belong to different clock domains, they communicate asynchronously through a limited number of handshaking signals, thanks to re-synchronization flip-flops.

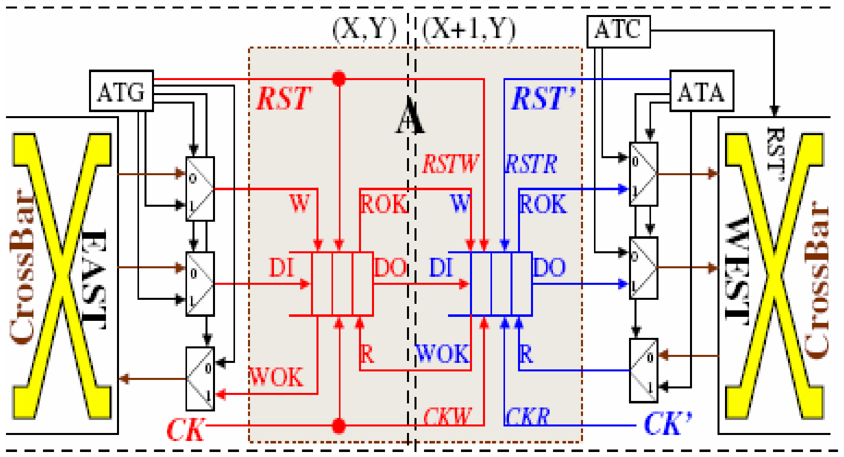

Figure 4 The router and/or channel de-activation/activation

4.1.4 De-activation/activation

The multiplexers controlled by ATG and ATA offer 2 functions: first, the multiplexers can isolate router test and channel test so as to avoid failures propagation; second, the multiplexers are used to de-activate the faulty channel.
A de-activated channel is configured to behave as a “Black Hole”. It discards any incoming data, and produces no outcoming data.
As shown in Figure 5, the channel A is de-activated or activated by the corresponding ATG/ATA couple, and the ATC FSM.
To de-activate a channel, ATG enforces W=0, RST=1, WOK=1, while ATA enforces R=1, RST’=1, ROK=0.
To activate the channel, ATG sets RST=0, selects the W, DI multiplexers with 0 and selects the WOK multiplexer with 1. ATA sets RST’=0, selects the R multiplexer with 0 and selects the ROK, DO multiplexers with 1.

4.1.5 Preparing the test and diagnosis phase

Once the free of faults routers and channels are activated, the NoC can be used since it is clean of malfunctions. As a black hole is set up to replace each faulty channel, sending packets through this channel will trigger a time out for the packet sender. This way we have a solution to inform the software that the path used for the packet transportation is not safe. The diagnosis process will be briefly discussed in section 6.

The whole initialization procedure (test and activation/de-activation) lasts about 300 cycles. When it ends, the clusters are just beginning their self-tests. When those finish their duty, the NoC is ready to be used.

4.2 Booting the MP²SoC: Intra Cluster Test & Diagnosis

The main objective of this step is for each cluster to briefly test its components and generate a local AIM. Let us describe the process for one cluster, assuming that it is replicated and executed concurrently within all the clusters.

4.2.1 **Embedded Software Based Self Test of the cluster**

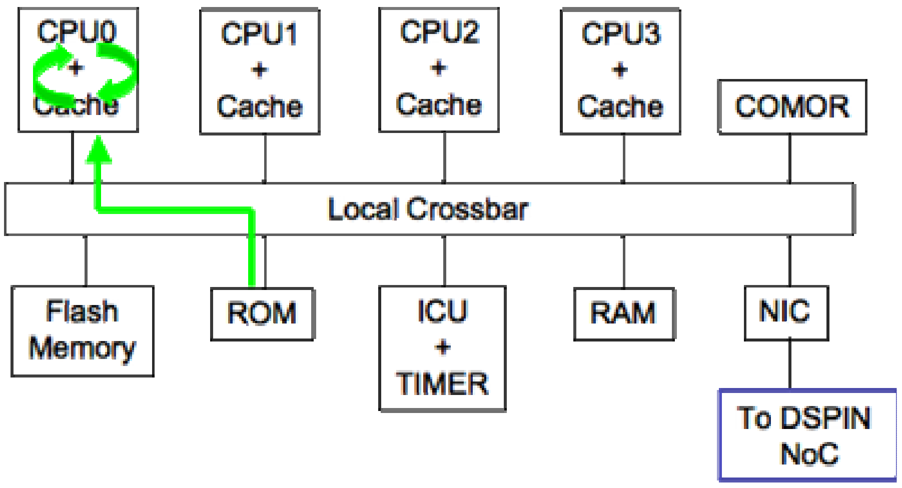

Figure 5 Testing the cluster

At booting time we consider that the NIC (Network Interface Controller) forbids the transmission of any packets. This is implemented in order to not disturb/false the T&D phase of other clusters by sending fake packets.
As the embedded memory is quite expensive, we plan to use a small sized code for this T&D step. Thus what we intend as “test” is a quick functional test compact enough to be stored in few Ko in a ROM memory. The more accurate test is stored in the external memory and will be executed later on.
First each processor loads and executes its own SBST test program located in the ROM.
Once done, the results are stored in a flash memory and we can proceed now to the SBST test of the remaining components. But for that, we need to elect a master processor, as the testing must be incremental and thus sequential.
The elected master processor is the one who passed the previous test and has the lower ID.
The remaining processors put themselves in an idle state and wait to be awakened, typically in the fine grain T&D phase.

4.2.2 Generation of a Local AIM

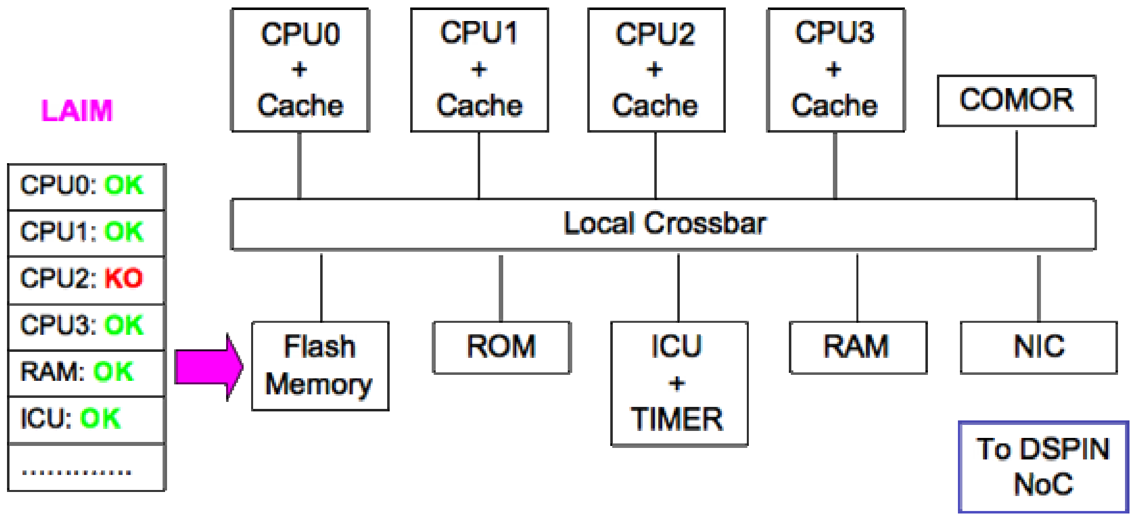

Figure 6 Generation of a Local Architecture Instant Map (LAIM)

After testing all the components with the appropriate software/algorithm test, for example a quick March Test for embedded RAMs, a table describing the status of the main components is saved in the flash memory. This table constitutes a Local AIM to be read in the next reboot of the platform, in order to skip the test of known failing components.
Once the LAIM is stored, the NIC is activated and the local master processor can candidate to the election of the Master T&D controller for the whole chip.

5 Accessing the External Memory Bank

As previously mentioned, the main objective of this step is to define a master test/diagnosis/reconfiguration controller able to access the external memory bank using a safe path through the NoC.
When beginning this step the NoC is free of failures that could lead to disturb the process. At the same time, the clusters ends or are about to end their self tests since we are in the GALS paradigm and the clusters don't run at the same frequency.

5.1 A Leader required

Here follows the requirements to fulfil the objectives:

- As the self-testing of the cluster is distributed we have to define a synchronization mechanism to continue the process.
- We need to compute at the end a unique map of what fails or not.
- As we have not hard time constraints, we do not need complex distributed algorithms to ensure the synchronization. We relaxed the timing constraints in favor to a small code size.

For that reasons, instead of implementing complex consensus based decision, we choose to proceed to the election of a processor responsible for the execution of the following steps.

5.2 A distributed election process

When a master processor of a cluster has stored its local AIM, it tries to access the external memory. If it succeeds, he is a candidate to the leadership of the whole MPSoC.
Then it tries to connect to the neighboring clusters to ensure the goodness of the links, waiting a certain amount of time if those ones have not finished yet their self-test.
Here begins the election process resulting at the end to the construction of a spanning tree over the network, a tree where the root is elected as the leader.
In fact, as depicted in picture 7, many trees begin to spread over the network since we have many candidates that can access the chip I/Os.

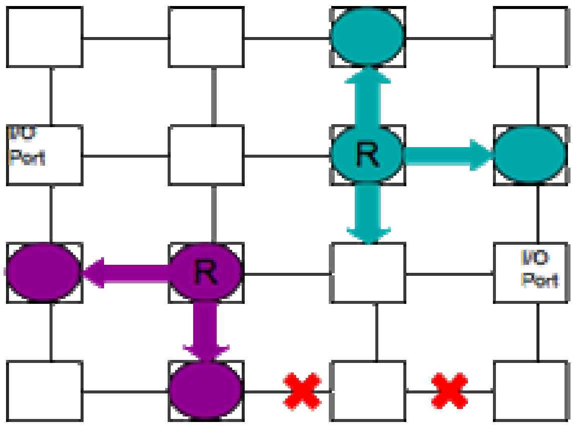

Figure 7. Two processors building their own spanning tree

The roots of the trees (candidates) send packets to their neighbors and if these ones do not belong to a tree, they are enrolled.
As shown in figure 8 the trees grow in a way depending on their priority (not detailed here).
Finally only one tree is built, whose nodes correspond to valid clusters, and whose edges refer to valid NoC links.

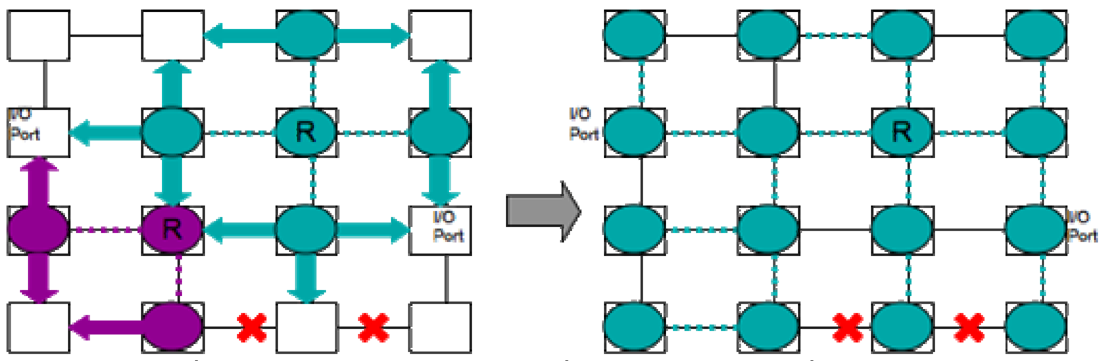

Figure 8. Concurrent spanning trees converging to one

5.3 The leader 's rôle

Once the tree is built, the leader will retrieve from the external memory the codes required to process the fine grain tests, the chip diagnosis and the reconfiguration. The leader then forwards/stores these codes within the main clusters. The leader is responsible for the scheduling and the synchronization of the different distributed tests and diagnosis. It centralizes the different diagnosis results to compute the expected AIM. It drives the reconfiguration step and finishes by executing the code initiating the remapping phase.

6 Fine Grain Test & Diagnosis

As said before, the fine grain test and diagnosis phase is controlled by the leader processor. This step consists in a more or less distributed way, to execute thorough testing of the chip and locate the component or a subpart of the component that has failed.
As done previously in section 4, we can split this phase in two: the T&D of the network on chip and the T&D of the clusters.

6.1 Fine grain NoC test & diagnosis

At this step, the NoC has been cleaned up by the initialization procedure, and has partially been tested during the spanning tree building. In order to ensure the correctness of the whole NoC components we have to test more accurately the NoC.
The NoC used, DSPIN, is in fact made of two separate subnetworks as a VCI-OCP protocol is implemented in the MPSoC: a command packets subnetwork and a response packets subnetwork. These dual networks are required to avoid deadlocks.
A NoC can be represented as a CDG (Channel Dependency Graph). Figure 9 describes what are the graph nodes and edges.

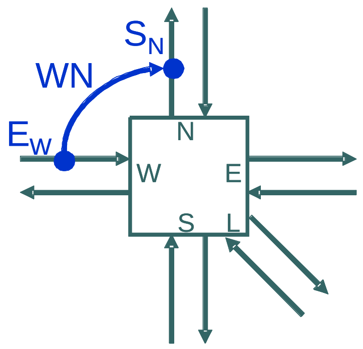

Figure 9. Representation of a router

As we can see, a node (Ew or Sn) corresponds to a communication channel and an edge (WN), a possible connection between two channels. Depending on the routing algorithm implemented, this edge is sets up or not. In our case the routing algorithm is a XY routing with X first, thus all the transition are not allowed. More details can be found in [1]

6.1.1 **Testing the NoC's paths**

For the DSPIN network, we can have for example the CDG depicted in figure 10.

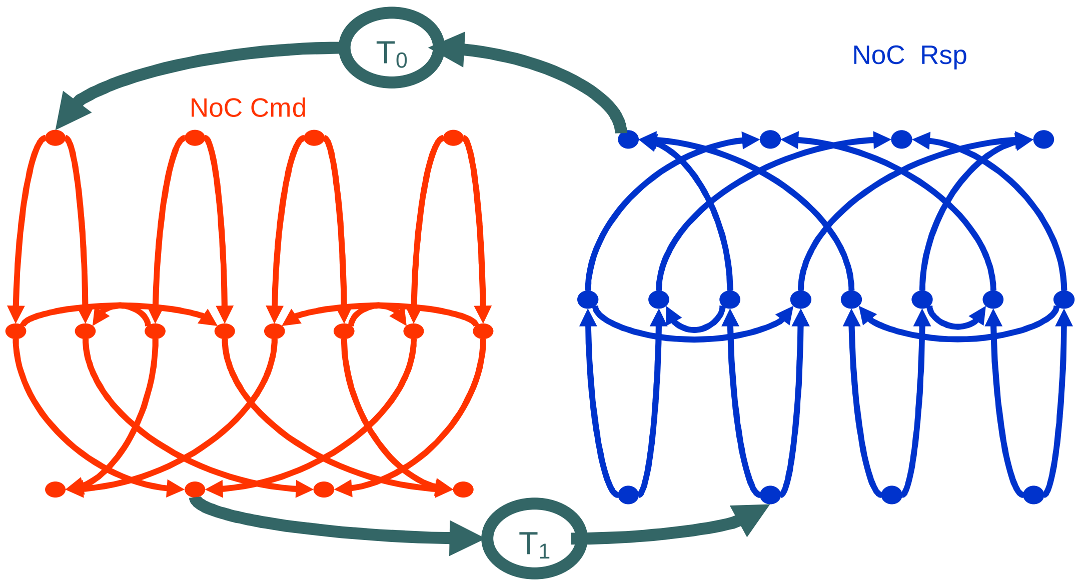

Figure 10. An example of a Channel Dependency Graph (CDG)

Testing the NoC corresponds to testing the whole CDG. Thus all CDG paths must be validated. The solution we propose is to send test packets in a distributed way to validate all the paths. In figure 10 we describe a 4X2 NoC. To is a processor injecting a packet in the NoC, T1 is a target RAM in another

cluster receiving the packet. The memory acknowledges the transfert by sending a response packet to the sender through the response NoC.
In case of a permanent failure within the paths, as we have replaced faulty channels by “Black Holes”, the sender will never receive a response and a time out information will be set up. This way, one path is tested for the two subnetworks. When all N clusters send packets to other N-1 clusters the test ends and then we can identify if a failure has occurred. The main advantage of this method is that no dedicated hardware is needed, except the one introduced for the initialization procedure.
Testing the NoC is thus a distributed software based process.

6.1.2 **NoC Diagnosis**

A path includes a set of nodes and edges. For a given cluster, each time the test of a path is done and the acknowledge is received by the sender, the associated nodes and edges are marked as fault free. At the end of the paths test, each cluster contains a database where all fault free nodes and edges are marked. Finally by crossing the databases of all the clusters, remains the faulty nodes and channels. We consider as a first approach, that if a node or a channel is faulty, the associated router must be deactivated. Thus, we can define a set of faulty routers. This corresponds to an AIM of the NoC. This information can be then reused as an entry to the reconfiguration step.

6.2 Fine grain cluster test & diagnosis

The fine grain Test & Diagnosis application is stored in the external memory. This application includes a set of programs which size and effectiveness depends on the targeted test quality. That is to say, if we want to achieve a high fault coverage, time and size consuming programs must be executed.
In our approach we distinguish two kinds of programs to be used, both based on software:

- Programs targeting functional testing of the cluster’s components.
- Programs targeting structural testing of the cluster’s components.
  Among these we can find the ones allowing the self-tests of microprocessors as described in [5]. More or less accurate software based March tests can be used to test the embedded memories.
  If the embedded cores are equipped with IEEE 1500 wrappers or IEEE 1149.1 JTAG interfaces, we can execute the software and the technique described in [6]

These programs can detect and localize failures at different level of accuracy: from a stuck at fault at a gate IOs up to a whole component. This provides great flexibility since if a failure occurs on a memory cell, it is not required to deactivate the whole memory, the corresponding segment can only be avoided when remapping the functional application on the hardware platform.
As stated previously, the Master T&D controller will forward the programs to all the clusters, and the master processor of the cluster will launch the testings.

At the end the chip controller will synthesize the diagnosis results coming from all the clusters and will compute the global AIM.

7 Reconfiguration & Deactivation of faulty components

Once having the AIM we have to reconfigure the hardware platform in order to make it usable for the remapping application.
Depending on the failing components, two types of operations must be processed:

- NoC failure

To prevent packets to be lost, the faulty routers must be deactivated and the surrounding routers must be reconfigured to avoid sending packets on the wrong way.
The deactivation of the faulty router is done thanks to the initialization procedure, by creating black holes on all the communication channels tied to the router.
Concerning the surrounding neighbors, a reconfigurable routing algorithm has been defined and implemented as described in [1]. Each router contains a four bits register to be filled with the appropriate value. The hardware/software technique to fill the routers reconfiguration registers is still under study.

- Core failure

Failing cluster's components must be deactivated in order not to generate traffic pollution on the networks that may lead to an application crash.
To prevent a core (processor, memories, coprocessors, …) from sending fake packets, we add a simple programmable component interfacing on one side the core and on the other side the local crossbar. This added component is simple since it is roughly a basic register. This component can be configured, through the use of an IJTAG port [7], by the COMOR (Configuration and Monitoring) coprocessor present in each cluster. The COMOR component is described in the first deliverable related to the online monitoring.
The validations of the reconfiguration and deactivation steps are on going works.

8 References

[1] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh Network-on-Chip," in *Proc. of DAC'08, the 45th Design Automation Conference*, 2008, pp. 441–446.
[2] LIP6. DSPIN. [Online]. Available: http://www.lip6.fr/Direction/2005- 05- 13- DSPIN.pdf
[3] I. Miro-Panades, F. Clermidy, P. Vivet, and A. Greiner, "Physical implementation of the dspin network-on-chip in the faust architecture," in *Proc. of NoCS'08, the 2nd ACM/IEEE International Symposium on Networks-on-Chip*, 2008, pp. 139–148.
[4] I. Miro Panades and A. Greiner, "Bi-synchronous fifo for synchronous circuit communication well suited for network-on-chip in gals ar-

chitectures," in *Proc. of NOCS '07,the 1st ACM/IEEE International Symposium on Networks-on-Chip*, 2007, pp. 83–94.
[5] N. Kranitis, A. Paschalis, D. Gizopoulos et G. Xenoulis. Software-Based Self-Testing of Embedded Processors. *IEEE Transactions on Computers*, 54(4) : pages 461–475, 2005.
[6] M. Tuna, M. Benabdenbi, and A. Greiner. At-Speed Testing of Core-Based System-On-Chip Using an Embedded Micro-Tester. In *25th IEEE VLSI Test Symposium (*VTS'07*)*, page 447-454, May 2007.
[7] IJTAG IEEE P1687. [Online]. Available: http://grouper.ieee.org/groups/1687/