Reliability Support for On-Chip Memories Using Networks-on-Chip

Federico Angiolini[†], David Atienza[‡#], Srinivasan Murali[‡⋆], Luca Benini[†], Giovanni De Micheli[‡]
[†] DEIS, University of Bologna, Bologna, Italy. [‡] LSI/EPFL, Lausanne, Switzerland.
[⋆] Stanford University, Palo Alto, USA. [#] DACYA/UCM, Madrid, Spain.

***Abstract*— As the geometries of the transistors reach the physical limits of operation, one of the main design challenges of Systems-on-Chips (SoCs) will be to provide dynamic (run-time) support against permanent and intermittent faults that can occur in the system. One of the most critical elements that affect the correct behavior of the system is the unreliable operation of on-chip memories. In this paper we present a novel solution to enable fault tolerant on-chip memory design at the system level for multimedia applications, based on the Network-on-Chip (NoC) interconnection paradigm. We transparently keep backup copies of critical data on a reliable memory; upon a fault event, data is fetched from the backup copy in hardware, without any software intervention. The use of a NoC backbone enables an efficient design which is modular, scalable and efficient. We proceed to demonstrating its effectiveness with two real-life application case studies, and explore the performance under varying architectural configurations. The overhead to support the proposed approach is very small compared to non-fault tolerant systems, *i.e.* no negative performance impact and an area increase dominated by that of just the backup storage itself.**

I. Introduction

Thanks to advances in process technology, an ever increasing amount of functionality can be integrated onto a single silicon die, leading to complex heterogeneous Multi-Processor System-On-Chip (MPSoC) architectures. However, as the geometries of the transistors reach the physical limits of operation, it becomes increasingly difficult for the hardware components to achieve reliable operation. The variability in process technology, the issue of thermal hotspots and the effect of various noise sources, such as power supply fluctuations, pose major challenges for the reliable operation of current and future MPSoCs [33], [1]. Failures may be temporary (for example if due to thermal effects) or permanent. Key MPSoC components that are affected by sub-micron technology issues are the on-chip memories [33], where errors can flip the stored bits, possibly resulting in a complete system failure. Current memories already include extensive mechanisms to tolerate single-bit errors, *e.g.* error-correcting codes such as Hamming codes [34], [13]. However these mechanisms are expensive and the overhead in area, power and delay to recover from multi-bit errors would be very high [33]. Hence, with the increasing uncertainty of device operation, an effective system-level support to memory fault tolerance will be mandatory to ensure proper functionality at a reasonable cost.

Simultaneously, market trends envisage consumer devices containing several integrated Intellectual Property (IP) cores, which communicate with each other at very high speed rates [20]. Efficient, scalable interconnection mechanisms among the components will be needed. Networks-on-Chip (NoCs) have been proposed as a promising replacement for traditional interconnections [6]. This technology extrapolates concepts from computer networks to provide scalable on-chip communication backbones. The use of NoCs helps designers to overcome the reliability issues of future technologies, for example because their high flexibility allows for the addition of redundant cores in the same chip (*e.g.* backup memories) without largely increasing the design complexity. The fault tolerance of NoCs themselves has been tackled by several previous papers. For example, noise and coupling phenomena on the links are faced in [9] and [10], where mechanisms for tolerating such interference issues are thoroughly presented. Soft errors can happen, but can be fixed by retransmission of corrupted packets; to this effect, error detection circuits can be coupled to schemes such as [7]. We assume these works as complementary to the present paper, and leverage upon them.

To understand how to cope with increasing physical-level unreliability, the characteristics of the target MPSoC software applications need of course to be studied in detail. Key drivers in this respect will be various multimedia services, such as scalable video rendering, videogames, *etc*. We will show that, for large classes of these applications, many types of data corruptions can be tolerated without perceivable service degradation, while only some small parts of the memory storage (which include the code segments) are really critical enough to require additional safeguards. We will validate this assumption by presenting case studies on two real-life multimedia applications.

As a major contribution of this work, we address the design of a reliable integrated memory subsystem for a NoC-based chip. The key idea is to automatically keep backup copies of critical data on a reliable memory; upon a fault event, data is transparently fetched from the backup copy in hardware, without any software intervention. To achieve this, we present a novel hardware solution that utilizes NoCs as the back-end. At the software level, we characterize the application data into two different types (**critical** and **non-critical**) and focus on the first category. We handle intermittent and permanent memory faults in the main memory; upon any occurrence of them, the NoC is dynamically reconfigured to switch all critical transactions to the backup memory. For transient failures, when the main memory recovers, the NoC switches back to the default mode of operation.

The use of NoCs to provide fault tolerance is key for several reasons. First, it guarantees modular and scalable designs. Backup devices can be added with minimal increases in design complexity; bandwidth can be added as needed to avoid performance bottlenecks due to the replication traffic. Second, dynamic fault tolerance can be supported in a way which is transparent to the software. Processors are unaware of any memory failures, thus only a limited effort by the application

1-4244-9707-X/06/$20.00 ©2006 IEEE

designer is required. Third, the NoC paradigm makes it very easy to place the main and backup memories far away in the chip floorplan; this is a key point to counter failures due to phenomena such as thermal hot-spots. Finally, our NoC architecture natively enables the decoupling of the frequency of the interconnect from those of the attached cores, allowing for clocking backup memories at a lower frequency. Therefore, the reliability of backup memories can be increased without the need for additional clock conversion logic.

We implement the proposed fault tolerance mechanism on top of our existing NoC platform [7], modeled in cycle-accurate SystemC, and integrate it within a NoC mapping flow [8]. We perform experiments on realistic multimedia applications, which show a negligible performance penalty. We present several experiments to explore various parameters that impact the performance and area overhead of the proposed mechanism. We synthesize the additional hardware components that are added to the NoC to provide the fault tolerance support. The silicon overhead is less than 10% the area of an extra backup memory bank itself (assuming a 32 kB size), which represents the baseline requirement for any replication-based fault tolerance strategy.

II. Case Study: MPEG4 Video Texture Coder

New multimedia applications cover a wide range of functionality (video processing, video conferencing, games, *etc.*); one of their main common features is that they process large amounts of incoming data in a streaming-based way (*e.g.* a continuous flow of frames). We can observe that certain parts of these streams are essential to produce a correct output, while others are not so critical and only partially affect the user-perceived quality. In many multimedia applications, it is possible to distinguish critical from non-critical data because each type is stored in different data structures within the applications. Let us briefly illustrate these characteristics in the implementation of a real-life multimedia application that is used as one of our case studies in Section VI, *i.e.* an MPEG-4 Video Texture Coder (VTC). VTC is the part of the MPEG4 standard that deals with still texture object decoding. It is a wavelet transform coder, which can be seen as a set of filter-banks [16] sent in a stream of packets. Each packet represents a portion of an image in different sub-bands, *i.e.* at different resolutions. The first packet of the stream includes the basic elements of the image, but at low resolution. This part is called the DC SUB-BAND of the wavelet. If the data that represents the DC sub-band is lost, the image cannot be reconstructed. As typical of critical data in streaming applications, it is very small in size (few kBytes for 800x640 images) and is stored in a dedicated variable and class within the VTC code. The following packets of the stream are called AC OR SPATIAL LEVELS and contain additional details about the image. They have a much larger size than the DC sub-band, but they only refine the image represented by the DC sub-band. If data representing these levels is lost, the user still sees an image, just at a lower resolution. Moreover, whenever a new frame arrives, the previous (faulty) picture is to be updated with the newly received information. Hence, any low resolution output only lasts a very limited amount of time.

From this example, we can derive fault tolerance requirements for typical multimedia applications. Only a small part of the data set is critical to the quality of output as perceived by the user, while most of the data to be processed is actually of little importance in this respect. Therefore, it is essential to preserve correct copies only of the former structures, while faults in the latter may be safely accepted.

III. Related Work

A large body of research exists on building MPSoCs that satisfy different performance and energy requirements for different applications [20]. Proposed solutions range from selection and customization of processing cores [5] to application-specific management techniques for on-chip memory hierarchies [21], [26]. One key issue for MPSoCs that has been addressed recently is the use of NoCs [6] to solve possible scalability problems with existing bus-based designs, such as AMBA [4] and CoreConnect [19]. Several NoC architectures have been presented [17], [7]. NoC fault tolerance has been tackled by several previous papers. For example, the authors of [9] describe a way of designing reliable NoC links by comparing them to radio channels, while link architectures can be tuned [10] to tolerate timing errors of up to 50% of the reference clock period. Data retransmission schemes are well known in wide area networks and have been applied to NoCs, for example in [7]; coupled with error detection circuitry, they allow for soft error recovery upon NoC links.

Regarding fault tolerance mechanisms at the micro-architectural level, reliability work on soft errors is presented in [1]. Redundant components can be used to increase processor lifetime and system reliability [28]. At the system level, dynamic fault-tolerance management [30] is shown to improve system reliability in embedded systems. Different metrics are proposed to estimate the effect of soft failures with particular attention to energy efficiency, computation performance and battery lifetime trade-offs. An interesting approach to simultaneously achieving SoC reliability and high efficiency is explored by [11] and [12]. There, the SoC is aggressively configured to comply with "typical case" constraints, thus delivering high performance and low power; in worst case conditions, which rarely occur, errors appear, but are transparently corrected either by a built-in checker or by timing error-tolerant circuitry.

The test and repair of SoCs, and more specifically of their memories, has been extensively explored [2], [35]. To provide reliable operation, the use of SINGLE-ERROR-CORRECTING-MULTIPLE-ERROR-DETECTING (SEC-MED) codes is already integrated in many on-chip memories [13]. Recent studies show that different program behavior patterns can be identified, and can be used to generate various custom error correction mechanisms for different memory portions [14]. A very important research area is represented by the development of memory cores with built-in self-test logic and spare storage resources [24], [18], [15]. While all these approaches have been demonstrated to be robust, they necessarily come at an area cost. In this paper, we propose to first split the application data traffic into logically distinct flows; subsequently, only the critical portion of data, which is expected to be comparatively small, may be backed up onto one of the above mentioned robust memories. Therefore, we believe our approach to be synergistic with those mentioned above.

Separate units (such as for example pre-existing microcontrollers [31]) have also been deployed in SoCs to supervise the status of on-chip memories. With respect to these techniques, we choose to leverage a built-in support in the underlying SoC communication infrastructure to minimize the silicon

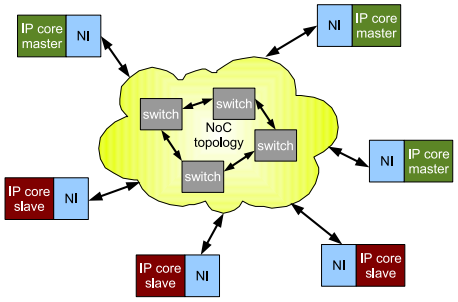


Fig. 1. General view of a NoC

overhead. Additional advantages of this choice are complete transparency to the software designer and the avoidance of any performance disruption upon fault occurrences. By leveraging NoCs as the communication backbone, the approach also guarantees maximum scalability.

IV. Baseline SoC Architecture and Extensions

A. SoC Template Architecture

The reference SoC that we consider is composed of computation cores, a communication backbone implemented by means of the ×pipes NoC, and a set of system memories.

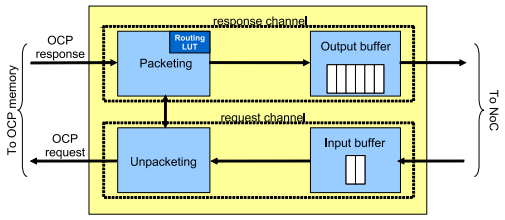
A typical NoC is built around three main conceptual blocks: Network Interface (NI), Switch (also called router) and Link (Figure 1). NIs perform protocol conversion from the native pin-out of IP cores to NoC packets; routers deliver packets to their recipients; and finally, links connect the previous blocks to each other, handling propagation delay issues. For the present paper, we illustrate our approach within the ×pipes NoC [7], by extending its building blocks to support reliability-aware features.

For our reference system, we assume the availability of two classes of memories: "error-detecting" and "reliable". Error-detecting memories, which can be commonly found today, are not capable of error correction but are at least capable of detecting faults, for example by Cyclic Redundancy Check (CRC) codes. We also postulate the availability of memories with much higher reliability for backing up critical data. This assumption is motivated by ad-hoc circuit level solutions and strengthened by three design choices we enable for these memories: (i) small capacity, (ii) lower-than-usual clock frequency (in this paper, we assume one half that of regular memories), (iii) during typical system operation, smaller workload than regular memories. We assume the existence of *main* memories having error detection capability; normal SoC operation leverages upon them, including storage of critical and non-critical data. We add smaller spare *backup* memories, featuring higher reliability, to hold shadow copies of critical data only. Each main memory requires the existence of one such backup, although a single storage device can hold backups for multiple main memories.

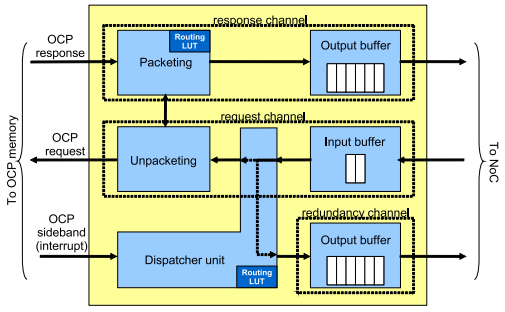
To identify the critical data set, we assume that the programmer defines the set of variables to be backed up, and maps them to a specific memory address range. This address range is then used to configure our NoC, either at design time or at runtime during the boot of the system. The accesses to this particular memory region are thereafter handled with our proposed schemes, improving the fault tolerance of the MPSoC design. Application code is assumed to be a vital resource too. Therefore, instructions are always treated in the same way as the critical data; in the remainder of the paper, we will not mention this distinction for the sake of simplicity. Note that the classification of data into critical and non-critical can also be done using efficient compiler support. In this case, the user can mark critical data using special macros and the compiler can map the data to a specific address range. The size of the critical set will depend on the application at hand, and is impossible to predict in general. We aim this work at streaming applications, mostly in the multimedia field, for which the amount of critical information can be safely assumed to be small in percentage. These applications do represent a significant slice of the embedded device market.

B. Proposed Hardware Extensions

To implement our approach, we perform changes to the NoC building blocks. The flexible packet-switching design of NoCs ensures that these changes are transparent to the transport layer (switches and links), but NIs need to be made aware of fault events. Two NIs exist natively: *initiator NI* (attached to a system master, such as a processor) and *target NI* (attached to a system slave, such as a memory). Both follow the OCP 2.0 [23] connection protocol specification at the IP core side and perform source routing by checking the target of the transaction against a routing LookUp Table (LUT).



(a)



(b)

Fig. 2. (a) Plain target NI architecture, (b) Extended target NI architecture

The changes to the target NI can be seen in Figure 2. The original target NI is still plugged to backup memories, while the extended version is used for main memories. A plain target NI features a request channel, where requests from system masters are conveyed, and a response channel, through which memory responses are packeted and pushed towards the NoC. A third channel (redundancy channel) is now added to the extended target NI; this channel is an output, and re-injects some of the request packets back again into the NoC.

By this arrangement, critical-data accesses to the memory (*i.e.* within a predefined address range) can be forwarded to the backup storage element. Not all packets are forwarded; during normal operation, that is until a fault is detected, only writes to critical address regions follow this path. This ensures that the backup memory is kept up to date with changes in critical data, but minimizes the network traffic overhead and increases the reliability of the backup memory, which faces a smaller workload. Since the backup memory only receives write commands, it remains silent, *i.e.* it does not send packets onto the NoC. This prevents conflicts such as two memories responding to the same processor request. The resulting flow of packets is depicted in Figure 3(a). The forwarding behavior is controlled by a DISPATCHER NI block, which supervises input and output packet flows. An extra routing LUT directs forwarded packets; the LUT consists of just a single entry, since there is only one backup memory per each main memory.

The extended target NI also features an extra interrupt interface by the memory side. Whenever a fault is detected, the memory can issue an interrupt. This triggers a reaction by the dispatcher, which responds by beginning to forward critical read packets to the backup memory according to the extra routing table entry. In this way, reads that would fail due to data corruption are instead transparently forwarded to the backup memory and safely handled (see Section V for more details). Critical writes continue to be forwarded as before.

The initiator NI is also extended. First, it checks all outgoing requests for their target address. If the address falls in the specific range provided by the application designer as storage of critical data, then a flag bit is set in the packet header. This allows the dispatcher in the extended target NI to very easily decide whether to forward packets or not. A second change in this NI involves an extra entry in its routing LUT, and a very small amount of extra logic that checks the `SourceID` field in the header of response packets. The initiator NI can thus detect whether a read request it sent got a response from the intended slave or from a different one. As we will show, in our approach, upon a fault, critical reads receive responses from the backup memory instead of the main one. Therefore, noticing a mismatch is an indirect indicator of whether there was a fault in the main memory. This can trigger different actions depending on the type of error that needs to be handled, as described in Section V.

V. RUN-TIME FAULT TOLERANT NOC-BASED SCHEMES

Two types of errors can occur in on-chip memories of MP-SoC designs, namely, transient or permanent. We assume that the system is able to recognize transient errors by detecting some known combination of parameters, either upon the error event itself or even before any error appears. For example, a thermal sensor detecting that a threshold overheating temperature has been surpassed may signal a "transient error" condition before any real fault is observed. The "transient error" condition would be deasserted once the temperature returns to acceptable levels. The same prevention or detection principle could be applied to other electrical or functional parameters that may indicate that a critical point of operation is being approached. In highly fault tolerant systems, the main memory is itself equipped with error correction (not only detection) logic; any internal correction event could then be pessimistically assumed as a hint of an imminent failure. This hypothesis could be reversed after a configurable period of time, once the isolated correction event can be safely assumed to be an occasional glitch, or maybe after a (self-)testing routine. Any known-critical or unexpected events should however be treated by the system as permanent faults, and accordingly handled.

In the following subsections we describe how the proposed extensions can be used to design schemes capable of handling both transient and permanent failures. In both cases, the backup memories do not contain any data upon boot, but are kept synchronized with the main memory at runtime.

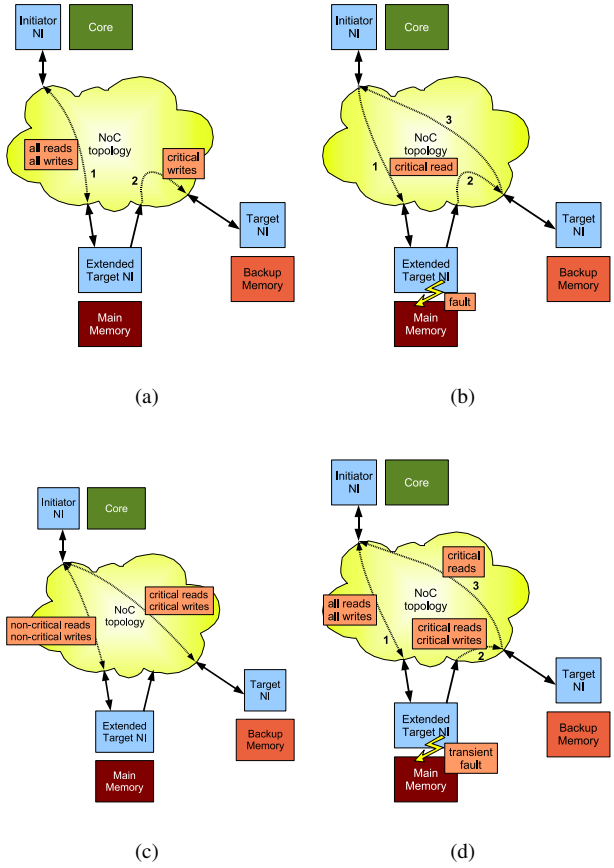


Fig. 3. Handling of packet flow in the system. (a) Normal operation with backup, (b) First phase of recovery for permanent and transient failures: read transaction handling upon fault occurrence, (c) Final operation mode after recovery from permanent failure, (d) Operation mode while a transient failure is pending

A. Permanent Error Recovery Support

As soon as a permanent error is identified, the recovery process begins. First, critical-region operations continue to be issued by the processors to the main memory as normal (see Section IV-B), but the extended target NI starts diverting both read and write requests to the backup memory. Therefore, the backup memory, which had been silent, begins to generate responses as a reaction to the read requests, while the main memory becomes silent for accesses into the critical address range. The `SourceID` field of request packets is kept unchanged, so that the backup memory automatically sends its reply to the system master that had originally asked for it without any lookup conversion. Figure 3(b) shows the handling mechanism of critical reads upon a fault.

Since going through the main memory and then the backup memory to fetch data is time consuming, the second phase of our recovery process for permanent faults tries to minimize the performance impact of this three-way handling of critical reads. The extended initiator NI (Section IV-B) is able to identify whether the source of read responses is the main or the backup memory. The first critical read after the fault occurrence triggers a mismatch detection, which in turn forces the initiator NI to access a different entry within its routing lookup table. Hence, all following memory reads within the critical address range are directly sent to the backup memory after the fault. This clearly improves latencies for the remaining operations. The resulting flow of packets is shown in Figure 3(c).

The approach does not introduce any data coherency issue. During normal operation, the forwarding of write transactions guarantees that critical data is always consistent among the main and backup memories. Writes are forwarded just before hitting the main memory bank, not after having been performed; in this way, a faulty main memory has no chance of polluting the backup copy of the data. The contents of the backup memory are updated after a slight delay, but this causes no issue as the sequence of packets is strictly maintained. Upon a fault occurrence, transactions are initially directed to the main memory, and only afterwards, when needed, are routed to the backup device; this arrangement avoids skipping transactions and guarantees that all pending transactions (reads and/or writes) are completed on the correct copy of the data. Therefore, proper functionality is strictly maintained when introducing the extra storage bank.

Similarly, when adding the backup memory to the NoC, deadlock issues do not arise given a proper design of the NoC routing scheme. In this respect, the NoC designer must accommodate for one extra IP core and some extra routing paths during the deadlock-free NoC mapping stage. We provide a streamlined way of handling the issue by integrating the discussed reliability enhancements within a complete NoC mapping flow called SUNMAP [8]. Briefly summarizing the flow, given an application task graph as an input, SUNMAP can map it onto an optimal NoC topology. Libraries which model the area and power consumption of NoC components are used to drive the optimization functions. An important property of our mapping tool is that its exploration algorithm automatically generates deadlock-free NoCs, removing the complexity of manually handling extra system components.

B. Intermittent Error Recovery Support

In the case of transient errors (*e.g.* due to overheating detection), the first phase of the recovery process is as seen above; critical-region read transactions are automatically forwarded to the backup memory, which automatically responds to the initiator. However, the second phase differs due to the nature of transient failures, where the main memory is supposed to recover complete functionality at a certain moment in time. All traffic, including the critical one, continues to be sent from the processor to the main memory. The extended target NI, being aware that a fault condition is pending, diverts all critical reads towards the backup memory, but lets critical writes be performed towards both the main and backup locations. When the main memory detects that it is able to return to normal operation (*e.g.* after a temperature decrease), it is allowed to issue a different interrupt to indicate so. The extended target NI then resumes normal operation.

The main assumption is that updates to the critical data set in main memory can be successfully performed even during the "transient fault" state. This might be allowed, *e.g.* by choosing conservative temperature thresholds to assert the fault warning. If this solution is not acceptable and the designer does not want to consider the fault permanent, we assume that a higher-level protocol will transfer the safe backup copies of critical data back to the main memory after its return to full functionality.

VI. Experimental Results

To assess the validity of our approach, we employ two different benchmarks from the multimedia domain. The first one is the MPEG-4 VTC application already described in Section II. As a second test, we use one of the sub-algorithms of a 3D Image Reconstruction algorithm [27], 3DR for short (see [32] for the full code of the algorithm, 1.75 million lines of C++ code), where the relative displacement between every two frames is used to reconstruct the third dimension. Similarly to the VTC benchmark, the amount of critical data that stores control information about the matching process (*e.g.* 160 kB for images of 640x480 pixels) is much smaller than the overall input data per each 2-frame matching process (2 MB at the same resolution), and is stored in two data structures which are easily identifiable by the application designer.

In our experiments, we run the 3DR and the VTC benchmarks on top of three reliability-enhanced topologies, as shown in Figure 4. Both benchmarks are implemented using 10 processing cores and a single main memory. The first topology is a NoC crossbar, the second is a star, and the third is a mesh. The topologies and benchmarks are chosen to illustrate different situations of performance penalty for adding reliability support, since the applications demand different features. In fact, 3DR tends to saturate the main memory bandwidth, while VTC is less demanding. The NoC is simulated within a cycle-true simulation environment. We clock the NoCs at 900 MHz, twice the frequency of the cores and memories [3].

A. Performance Studies

We run the benchmarks in five different setups. The first two are reference baselines, the remaining ones represent our proposed scheme.

- *Reference-Unreliable*: our reference run is a system without reliability support at all, where accesses are to a fast (450 MHz) main memory. No faults are injected.
- *Reference-Robust*: we model the same system with a reliable main memory running at a lower frequency, therefore minimizing error occurrences [12] and accounting for the overhead of extra circuitry. System performance is obviously impacted, but robust operation can be assumed.
- *Proposed-Replication*: we create a system with a fast main memory and deploy a slow backup memory, but we do not yet inject any fault in the system. As a result, the overhead for the backup of critical data can be observed. We assume the backup memory to be clocked at half the clock speed of regular memories, for the same reasons outlined in the previous setup.
- *Proposed-Permanent*: we create a system with a fast main memory and deploy a slow backup memory, then inject a permanent fault right at the beginning of the simulation. This enables the evaluation of the impact of accessing the backup copy of critical data.

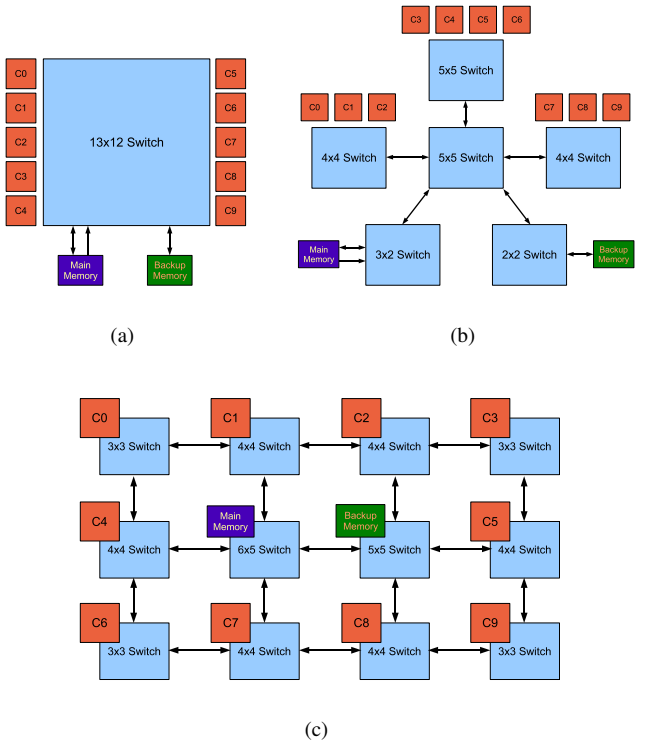


Fig. 4. The three topologies under test: (a) crossbar, (b) star, (c) mesh

- *Proposed-Transient*: we create a system with a fast main memory and deploy a slow backup memory, then inject a transient fault right at the beginning of the simulation, and never recover from it. This analysis helps to understand what happens to system performance during the period where the main memory is accessed first, but critical traffic needs to be rerouted to the backup memory.

Figure 5 reports performance, measured in completed transactions per second, for our test setups. The system throughput of most of the scenarios is close, with *Reference-Robust* being much worse than average and *Proposed-Permanent* performing much better, at least in the 3DR case, than even *Reference-Unreliable*. We explain these major effects by observing that both benchmarks, like most multimedia applications, place heavy demands in terms of memory bandwidth; this is a logical consequence of parallel computing on a 10-core system. In *Reference-Robust* the available memory bandwidth is decreased to provide more reliability, which causes performance to worsen dramatically: throughput drops by about 24% in VTC and by as much as 43% in 3DR, which is even more demanding. For the same reasons, the *Proposed-Permanent* scenario, where critical data is stored in a separate device, actually guarantees a performance boost related to load balancing among the two memories; the boost is up to 40% for 3DR. Under less demanding applications, we expect both scenarios to perform more similarly to the baseline. The *Proposed-Replication* scenario exhibits a minimal penalty compared to the unreliable case, since the traffic overhead is well handled by the NoC. VTC rarely accesses critical regions, so no penalty is noticeable; in 3DR the throughput decreases 1% to 9%. Finally, the *Proposed-Transient* case exhibits a performance level close to *Reference-Unreliable*, because non-critical traffic behaves exactly as in the base scenario, but several effects related to critical traffic have to be accounted for. On the one side, critical traffic creates NoC congestion and incurs a latency overhead. On the other hand, the main memory does not have to process critical reads, therefore the non-critical transactions can be executed with less delay. In VTC, the overall balance is roughly even. In 3DR, where a larger amount of critical reads (*e.g.* instruction cache refills) takes place, the main memory benefits from large latency gains.

Experimental results show that, in order to improve system reliability, deploying a single highly fault tolerant main memory (*Reference-Robust*) may not be a wise choice in terms of performance within complex multimedia systems. In our proposed architecture, the main memory is left running at a high frequency, and a slower secondary memory bank is added. This choice incurs minor throughput overheads both during normal operation and after fault occurrences. These results justify the feasibility of deploying our architecture even in throughput-constrained environments.

The gains we outline for the *Proposed-Permanent* scenario suggest that always mapping critical information to a separate reliable memory, without inter-memory transactions, may be a simpler yet efficient approach, due to load balancing. However, such a choice does not improve reliability as much as our backup mechanism. First, having two copies of critical data is certainly more reliable than having a single one. Second, using the main memory as the default resource permits a lower workload for the backup memory during normal operation (only write transactions need to be processed), which further increases its reliability. Since the focus of this paper is high fault tolerance, we feel that a redundant data mapping is justified, and our aim is simply to verify that performance is not seriously impacted as a result. Performance optimizations through reduction of local congestion can always be achieved by the system designer by tuning the memory hierarchy, which includes deploying multiple storage elements; these steps can be taken in combination with our proposed approach.

B. Architectural Exploration of NoC Features

We extend our analysis to different NoC-based hardware architectures using the same NoC backbone. We vary some parameters of our baseline topologies. First, we modify the star topology of Figure 4(b) by attaching the backup memory beyond a further dedicated switch. The total distance from the central hub is therefore of two hops instead of one. In this way we model backup memories further apart from main memories in the chip floorplan, which improves the tolerance to local overheating. Performance is unchanged under the *Reference* scenarios, where the backup memory is never accessed. In *Proposed* scenarios, where the backup storage is in fact accessed, throughput worsens by less than 0.3%. This is because the latency to go through an extra hop in the NoC is very small, provided there is limited congestion. If the latency to reach the backup memory becomes too large, the topology designer may want to add dedicated NoC links.

To test the dependency of performance on the buffer depth of the redundancy channel, we try a sweep by setting this parameter within the extended target NI from 3 to 6 stages. Our results indicate that, both in VTC and 3DR, deep FIFOs only improve system performance by less than 2%, which indicates that large buffering is not mandatory in the extended target NI.

To validate the effectiveness of the routing shortcut that is enabled in the initiator NI after permanent faults, we measure

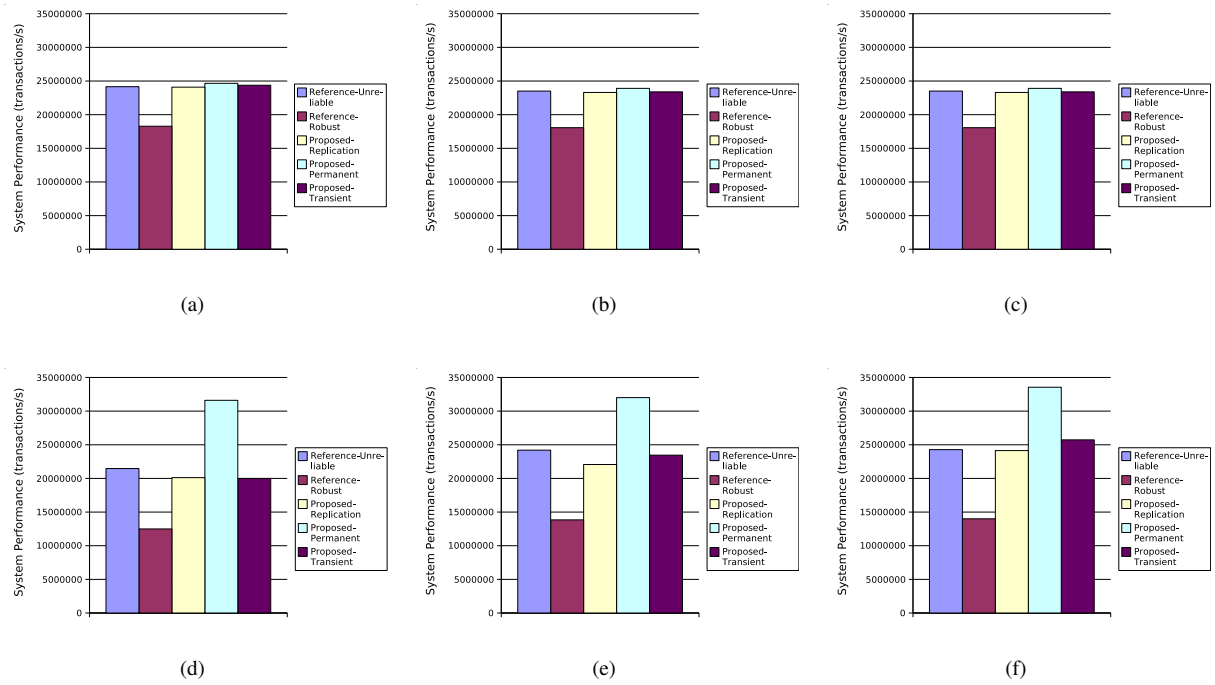


Fig. 5. Comparative performance of adding reliability support for (a, b, c) the VTC benchmark on crossbar, star and mesh topologies respectively, (d, e, f) the 3DR benchmark on crossbar, star and mesh topologies respectively

the latency of two different transactions on the star topology: (1) a critical read going from the core to a faulty main memory, bouncing towards the backup memory, and from there to the processor again and (2) a read directly towards the backup memory after the processor has updated its internal lookup tables. The minimum latency is cut from 78 to 68 (-13%) clock cycles, and the average one goes down from 103 to 95 (-8%). This metric, while topology-dependent, shows the advantage of updating the routing decisions of the initiator upon permanent faults.

C. Effects of Varying Percentages of Critical Data

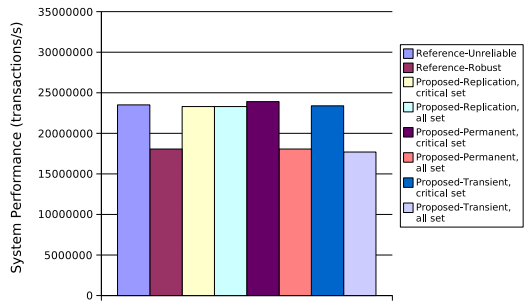
It is important to explore different reliability/performance trade-offs according to the amount of variables that are considered critical: the more data needs to be backed up, the larger the safe backup memories need to be. Since backup memories are supposed to be reliable also thanks to being smaller, slower and relatively little accessed, the effect of having large backups upon reliability is unclear. To shed some light onto the performance side of the issue, we analyze the behavior under different rates of possible critical *vs.* non-critical data in Figure 6. The star topology is taken as an example. In the plots, the *Reference-Unreliable* bar can be assumed to represent an ideal case where no data is critical. For the *Proposed* cases we protect against faults two different memory area: the actual critical set of the benchmark (the same of the studies in Figure 5, labeled "critical set"), and as an extreme bound, the whole address space ("all set"). The first interesting remark is that the *Proposed-Replication* performance, *i.e.* the system throughput before any fault, but in presence of the backup overhead, is only moderately impacted by the size of the critical data set. In the worst case of 3DR, which is severely bandwidth-limited, even backing up the whole address space incurs a penalty of just 18%. On the other hand, in case of a fault, the size of the protected memory space is a key performance parameter. While choosing a small critical set allows for very good throughput, extending the fault tolerance to the whole main memory content incurs a large penalty. This is in agreement with expectations; in both the *Proposed-Permanent* and the *Proposed-Transient* cases, all traffic is ultimately redirected to the backup memory, which is running at a lower frequency: therefore, throughput becomes similar to the *Reference-Robust* baseline.

This bracket of results frames the applicability of our approach. If the critical set of the application can be kept small, throughput penalties are minimal and advantages are clear. Otherwise, performance degrades up to a worst case equivalent to a system with a single reliable memory.

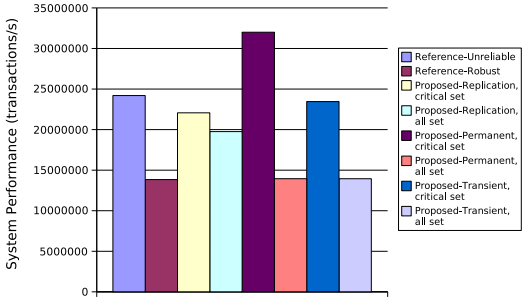
D. Synthesis Results

Regarding the modifications in the NoC to support a backup memory, four changes are needed: (i) the NI associated to the main memory must be augmented, (ii) the backup memory needs an extra (plain) target NI device, (iii) the initiator NI becomes a bit more complex, (iv) extra links and switch ports may be needed for routing data to the backup memory.

To assess the silicon cost of the proposed extensions, we synthesize the original and extended NIs with a 0.13 μm UMC technology library. Initiator NIs experience no operating frequency penalty to support the extra functionality, while area increases by about 7% (0.031 mm^2 against 0.029 mm^2). We also study extended target NIs, having 4-slot buffers in the response channel and 3- to 9-slot buffers in the extra redundancy channel. The impact on maximum achievable frequency is just of 2% to 6%, negligible in a NoC where the clock frequency is limited by the switches [3]. By adopting a 4-slot buffer identical to that of the response channel, area increases from 0.032 mm^2 to 0.039 mm^2.



(a)



(b)

Fig. 6. Impact of adding reliability support on the star, with different sizes of the critical data set, for (a) VTC, (b) 3DR

As a result, the area cost due to NI changes is 0.041 mm^2. Overall, even including other possible overheads in the NoC (*i.e.* extra ports in switches and extra links), the final overhead is still small in comparison to the area of the extra backup memory bank itself, which can take on average 1 mm^2 of area for a 32 kB on-die SRAM in 0.13 μm technology.

VII. Conclusions

With the growing complexity in consumer products, a generation of System-On-Chip (SoC) architectures with extreme interconnection fabric demands is being envisioned. One of the main challenges for designers will be the deployment of fault tolerant architectures. In this paper, we have presented a complete approach to countering transient and permanent failures in on-chip memories, by taking advantage of the communication infrastructure provided by reliable Network-on-Chip (NoC) backbones. Our design is based on modular extensions of the network interfaces of the cores, and is transparent to the software designer. The only activity required by the programmer is minimal code annotation to tell the compiler which parts of the data set are critical. The extensions are integrated within a NoC mapping flow, which transparently handles instantiation issues. Our experimental results show that the proposed approach has a very limited area overhead compared to non-reliable designs, while being scalable for any number of cores.

VIII. Acknowledgments

This work has been supported by STMicroelectronics, the Swiss FNS (Research Grant 20021-109450/1), the Spanish Government (Research Grant TIN2005-5619), the US NSF (contract CCR-0305718), the Semiconductor Research Corporation (SRC) (contract 1188).

References

[1] V. Agarwal, et al. The effect of technology scaling on microarchitectural structures. Technical Report TR2000-02, University of Texas at Austin, USA, 2002.
[2] R. Aitken, et al. Redundancy, repair, and test features of a 90nm embedded SRAM generator. In *Proc. ISDFT in VLSI Systems*, 2003.
[3] F. Angiolini, et al. Contrasting a NoC and a traditional interconnect fabric with layout awareness. In Proc. *DATE*, 2006.
[4] ARM Inc. *AMBA Specification*, May 1999.
[5] Kubilay Atasu, et al. Automatic application-specific instruction-set extensions under microarchitectural constraints. In Proc. *DAC*, 2003.
[6] Luca Benini et al. NoC: a new SoC paradigm. *IEEE Computer*, 2002.
[7] Davide Bertozzi et al. ×pipes: A network-on-chip architecture for gigascale systems-on-chip. *IEEE Circuits Systems Mag.*, 2004.
[8] Davide Bertozzi et al. Error control schemes for on-chip communication links: the energy-reliability trade-off. *IEEE Trans. on CAD*, 2005.
[9] Morgenshtein et al. Micro-modem - reliability solution for NoC communications In Proc. *ICECS*, 2004.
[10] Tamhankar et al. Performance driven reliable link design for networks on chips In Proc. *ASP-DAC*, 2005.
[11] T. M. Austin DIVA: a reliable substrate for deep submicron microarchitecture design In Proc. MICRO-32, 1999.
[12] T. M. Austin et al. Opportunities and challenges for better than worst-case design In Proc. *ASP-DAC*, 2005.
[13] M. Blaum, et al. The reliability of single-error protected computer memories. *IEEE Trans. Comput.*, 1988.
[14] Nicolas S. Bowen et al. The effect of program behavior on fault observability. *IEEE Trans. Comput.*, 1996.
[15] M. Choi, et al. Optimal spare utilization in repairable and reliable memory cores. In *Proc. Int. Workshop on Mem. Tech., Design and Testing*, 2003.
[16] Irak Sodagar et al. Scalable wavelet coding for synthetic and natural hybrid images. *IEEE Trans. on Circ. and Syst. For Video Technologhy*, 1999.
[17] Kees Goossens, et al. Aethereal NoC:concepts, architectures, and implementations. *IEEE Des. and Test of Computers*, 2005.
[18] Chih-Tsun Huang, et al. Built-in redundancy analysis for memory yield improvement. *IEEE Trans. on Reliability*, 2003.
[19] IBM. *The CoreConnect Bus Architecture, Product Specification*, July 1999.
[20] Ahmed Jerraya and Wayne Wolf. *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, Elsevier, 2005.
[21] M. Kandemir, et al. A compiler based approach for dynamically managing scratch-pad memories in embedded systems. *IEEE Trans. on CAD*, 2004.
[22] Sorin Manolache, et al. Fault and energy-aware communication mapping with guaranteed latency for applications implemented on NoC. In Proc. *DAC*, 2005.
[23] OCP International Partnership (OCP-IP). Open core protocol standard 2.0, 2003. `http://www.ocpip.org/home`.
[24] J. Ohtani, et al. A shared built-in self-repair analysis for multiple embedded memories. In Proc. IEEE CICC), 2001.
[25] Jose Oliver et al. Fast and efficient spatial scalable image compression using wavelet lower trees. In Proc. *DCC*, 2003.
[26] Preeti Ranjan Panda, et al. Data and memory optimizations for embedded systems. *ACM TODAES*, April 2001.
[27] Marc Pollefeys, et al. Metric 3D surface reconstruction from uncalibrated image sequences. In Proc. *SMILE*, 1998.
[28] P. Shivakumar, et al. Exploiting microarchitectural redundancy for defect tolerance. In Proc. *ICCD*, 2003.
[29] Tajana Simunic et al. Managing power consumption in networks on chips. In *Proc. DATE*, 2002.
[30] P. Stanley-Marbell et al. Dynamic fault-tolerance management in failure-prone battery-powered systems. In *Proc. IWLS*, 2003.
[31] Chin-Lung Su, et al. A processor-based built-in self-repair design for embedded memories. In *Proc. of ATS*, 2003.
[32] Target jr, 2002. `http://computing.ee.ethz.ch/sepp/targetjr-5.0b-mo.html`.
[33] Hua Wang, et al. Systematic analysis of energy and delay impact of very deep submicron process variability effects in embedded sram modules. In *DATE*, 2005.
[34] Sihai Xiao, et al. A generalization of the single b-bit byte error correcting and double bit error detecting codes for high-speed memory systems. *IEEE Trans. on Computer*, 1996.
[35] Yervant Zorian, et al. SRAM design on 65-nm CMOS Technology with dynamic sleep transistor for leakage reduction. *IEEE Trans. on Computer*, 1999.