Shield: A Reliable Network-on-Chip Router Architecture for Chip Multiprocessors

Pavan Poluri, *Student Member, IEEE* and Ahmed Louri, *Fellow, IEEE*

Abstract—The increasing number of cores on a chip has made the network on chip (NoC) concept the standard communication paradigm for chip multiprocessors. A fault in an NoC leads to undesirable ramifications that can severely impact the performance of a chip. Therefore, it is vital to design fault tolerant NoCs. In this paper, we present *Shield*, a reliable NoC router architecture that has the unique ability to tolerate both hard and soft errors in the routing pipeline using techniques such as spatial redundancy, exploitation of idle cycles, bypassing of faulty resources and selective hardening. Using Mean Time to Failure and Silicon Protection Factor metrics, we show that Shield is six times more reliable than the baseline-unprotected router and is at least 1.5 times more reliable than existing fault tolerant router architectures. We introduce a new metric called Soft Error Improvement Factor and show that the soft error tolerance of Shield has improved by three times in comparison to the baseline-unprotected router. This reliability improvement is accomplished by incurring an area and power overhead of 34 and 31 percent respectively. Latency analysis using SPLASH-2 and PARSEC reveals that in the presence of faults, latency increases by a modest 13 and 10 percent respectively.

Index Terms—Network-on-chip, router architecture, hard faults, soft errors, mean time to failure

1 Introduction

Rapid technology scaling [1], [2] into the deep nanometer regimes has facilitated designers to fabricate billion transistor chips. The abundant availability of on-chip transistors coupled with the desire to design low power chips that either maintain the same level of performance or improve performance compared to their predecessors has led to the inception and rise of chip multiprocessors (CMPs). As a result, there has been a paradigm shift from the design of computation oriented architectures to communication oriented architectures wherein, the communication between multiple computational cores on a single chip plays a crucial role in the performance of the chip. The need to efficiently handle the strict communication requirements of CMPs has led to the inception of network-on-chip (NoC) paradigm [3], [4], [5].

While the decreasing feature size resulted in smaller gate delays, it has exacerbated the global wire delay problem [6], [7]. NoC overcomes the problems associated with this global wire delays and very limited bus bandwidth by replacing the ad-hoc shared buses with a modular and flexible interconnect structure comprised of shorter wires thereby increasing simultaneous communication and on-chip bandwidth [8], [9], [10], [11], [12], [13], [14], [15], [16]. A typical NoC is comprised of links and routers that are used for data transmission and packet routing respectively.

The continuous decrease in feature size has increased the vulnerability of the transistors and wires to various fault mechanisms [17]. Faults can be primarily categorized into permanent faults and transient faults [18]. A fault that manifests either at the fabrication time or during the operation time of a circuit and continues to affect the circuit's functionality from the time of its genesis is called a permanent or hard fault. Traditional causes of permanent faults include time-dependent dielectric breakdown (TDDB) [19], negative bias temperature instability [20], hot carrier injection [21] and electromigration [22].

On the other hand, a fault that survives typically for a period of one or at most two clock cycles and affects the functionality of a circuit only during that period is called a transient fault. Transient faults commonly occur during runtime and are mainly caused due to alpha particles from packaging material [23], [24] and thermal radiation from cosmic rays [25] and process variation [26].

A fault in the NoC could result in undesirable scenarios such as deadlock in the network, packet loss, increased packet latency, erroneous messages, all of which, result in significant consequences on the performance of the chip. Hence, it is of utmost importance to tackle the reliability of the NoC from initial design stages.

In this work, we concentrate on the issue of fault tolerance in an NoC router. Fault tolerance of the links has been addressed previously by many scientists [14], [27], [28], [29], [30], [31] and therefore it is out of scope of this paper. Within the router, we focus on its *pipeline*, as it is responsible for the smooth flow of packets through the router. We present a reliable router architecture that is capable of tolerating both hard and soft errors in the routing pipeline. The proposed reliable router architecture's hard fault tolerance methodology is based on employing techniques such as spatial redundancy, exploitation of idle cycles of existing resources and bypassing faulty resource. This hard fault tolerance methodology was first proposed in [32]. Further analysis of this methodology using Mean Time to Failure (MTTF) reliability metric and benchmark applications has

• *The authors are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721. E-mail: {pavanp, louri}@email.arizona.edu.*

Manuscript received 19 Sept. 2014; revised 24 Sept. 2015; accepted 19 Nov. 2015. Date of publication 25 Jan. 2016; date of current version 14 Sept. 2016. Recommended for acceptance by A. Gordon-Ross.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2521641

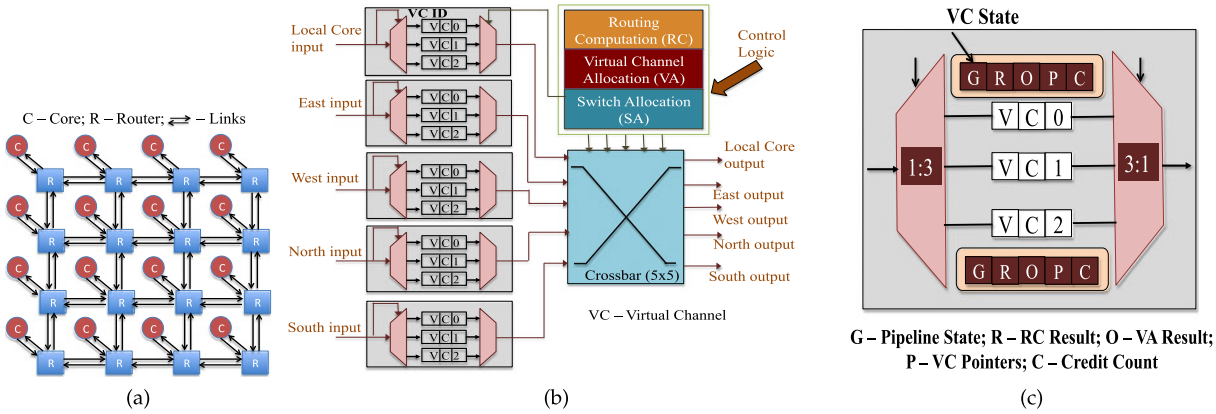


Fig. 1. (a) Mesh based NoC. (b) Details of an NoC router. (c) Details of a router's input port.

been conducted and presented in [33]. In this paper, we tackle the issue of soft errors in the routing pipeline by using selective hardening technique, and present an integrated fault tolerant router architecture called *Shield*, that can tolerate both hard and soft errors.

The remaining of the paper is presented as follows: in Section 2, we present the fundamentals of an NoC router and a brief working of the router pipeline. In Section 3, we briefly discuss about the existing fault tolerant router architectures and the motivation for proposing the Shield architecture. In Section 4, we discuss the effects of hard faults in the pipeline and summarize the techniques we use to tolerate hard errors. In Section 5, we discuss in detail the selective hardening technique used to improve soft error tolerance in Shield. In Section 6 we present the reliability analysis of Shield conducted with respect both hard and soft errors and in Section 7 we present the performance analysis in terms of area, power and critical path. Section 8 discusses the latency analysis performed on an NoC and Section 9 concludes the paper.

2 Baseline NoC Router Architecture and Pipeline

Fig. 1a shows the arrangement of sixteen cores and routers in a 4 × 4 mesh topology. Fig. 1b [9] illustrates the baseline architecture of an NoC router comprised of *five* input ports and *five* output ports with each input port comprised of *three* virtual channels (VCs). We have shown three VCs per input port for illustration purpose. This is the traditional architecture of a router in a mesh topology. Routing Computation (RC) unit, Virtual Channel Allocation (VA) unit and Switch Allocation (SA) unit are responsible for the control signals generated within the router pipeline during the traversal of a packet. The 5 × 5 crossbar (XB) connects the input ports of the router to its output ports and facilitates packet traversal.

For efficient router resource utilization, data traverses the NoC in the form of *flits*. A packet is segmented into a single *head flit*, single or multiple *body flit(s)* and a single *tail flit*. Head flit is responsible for allocating necessary resources for a packet, body flit(s) contain the payload of the packet and the tail flit is responsible for freeing the resources allocated to the packet.

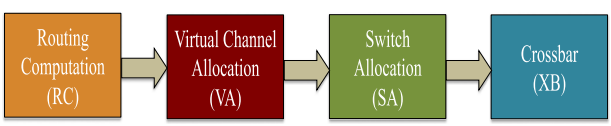


Fig. 2. NoC router pipeline.

2.1 Details of a Router's Input Port

Fig. 1c [9] shows the internal architecture of a router's input port with three VCs. Each input port is comprised of a demultiplexer, VCs (buffers) and a multiplexer. Each input VC is associated with state information comprised of five fields namely 'G' (indicates the current pipeline state), 'R' (stores the result of RC), 'O' (stores the result of VA), 'P' (read/write pointers into the VC) and 'C' (indicates the available credit count). This state information facilitates the smooth flow of a packet in that VC through the router pipeline.

2.2 Brief Overview of the Router Pipeline Stages

Fig. 2 [9] shows the four-stage pipeline of an NoC router.

Stage 1 (RC). This stage is responsible for deciding the output port at the current router through which the packet will exit and traverse towards its next hop. This decision is made based on the destination information in the head flit and the routing protocol used in the NoC.

Stage 2 (VA). This stage is responsible for allocating to the packet at the current router an empty VC at the downstream router. This stage can be performed in two sub-stages [34] (as shown in Fig. 3). Every input VC with a head flit that has just finished RC, arbitrates for an empty VC at the downstream router in the first sub-stage. In the second sub-stage, all the different input VCs that are allocated the same VC at the downstream router compete with each other to determine which input VC is allocated the empty VC.

Stage 3 (SA). This stage is responsible for determining which input VC of an input port gets to transmit a flit through the crossbar in the next cycle. This stage can be performed in two sub-stages [34] (as shown in Fig. 3). The first sub-stage chooses the winning input VC of a specific input port and the second sub-stage resolves the competition between packets in different input VCs trying to access the same output port of the crossbar.

Stage 4 (XB). Crossbar facilitates connections between the input and output ports of a router. The winning input VCs in switch allocation stage transmit a flit in this stage. The

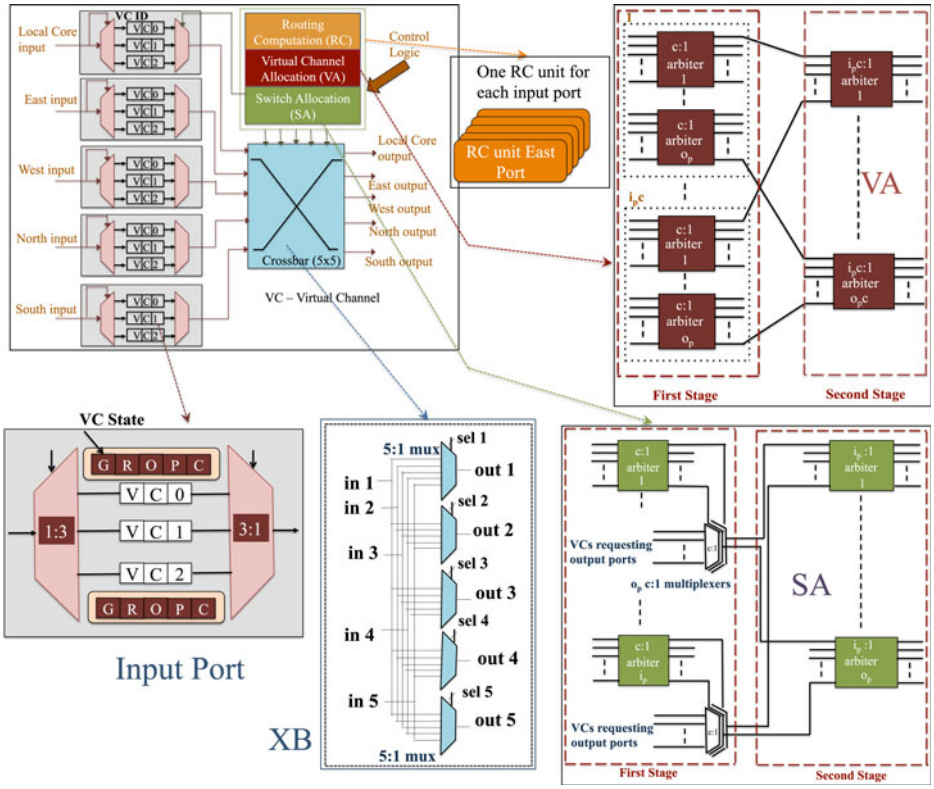


Fig. 3. Component details of a baseline router.

crossbar connections are configured every cycle and are determined by the winners of switch allocation stage.

3 Related Work

In this section, we provide a brief overview of the existing methodologies that tackle the issue of fault tolerance in the router pipeline.

In [35], the authors propose the BulletProof router, which provides fault tolerance from permanent faults via N-modular redundancy (NMR) technique. Spatial redundancy based techniques require the existence of multiple copies of hardware thus, significantly increasing the silicon footprint as well as the power consumed by the protected router.

In [36], the authors propose Vicis that tolerates permanent faults both at the network as well as at the router level. It employs a combination of input port swapping algorithm and an adaptive routing algorithm to redirect traffic around faulty links. An intra-router bypass bus is used to tolerate faults in the crossbar and error correcting codes (ECC) is used to tolerate faults in the datapath of the router.

In [37], the authors propose RoCo router, which can be disintegrated into independent row and column components and hence, a permanent fault in one component will not affect the other component and the router continues to operate in a degraded manner using the fault-free component.

In [38], the authors propose the use of default backup paths (DBPs) to bypass faulty components within a router. All the DBPs in the NoC form an unidirectional ring topology that allows the cores to communicate with each other even after all the routers in the NoC are faulty.

In [39], the authors propose the use of allocation comparator (AC) to detect transient faults. The AC unit performs three comparisons in parallel where it compares the routing computation, virtual channel allocation and switch allocation entries and detects the presence of a fault based on the concept of invariance checking.

In [40], the authors propose to protect the routing computation and arbitration units of an NoC router from transient faults by using the inherent information redundancy present in the NoC router. For faults in the routing computation unit, they use sigma and branch error detection method and for faults in the arbitration units they use a self-detecting and self-correcting round robin arbitration methodology.

In [41], [42], the authors model transient faults as either affecting the header of a packet or the payload of a packet. In order to recover from a transient fault, the authors have implemented a customized packet retransmission scheme where, the receiver after receiving a predefined number of packets sends an acknowledgement to the transmitter to confirm successful reception of the packets. To tolerate permanent faults, the authors have implemented a dynamic routing protocol to exploit the existence of alternate routes in the NoC. Both these implemented mechanisms achieve fault tolerance by operating at the network level of the system stack.

In [43], the authors model both permanent and transient faults as faulty links. To tolerate permanent faults, the authors have implemented a deflection routing algorithm that can learn. For transient faults, a combination of automatic repeat request and forward error correction mechanisms have been used. These mechanisms have been proposed for buffer less NoCs.

The motivation for the proposal of *Shield* is to design an integrated NoC router architecture that can tolerate both permanent and transient faults in the four stages of the

router pipeline using techniques that operate either at architectural or at circuit level in the system stack. Shield contains fault tolerant techniques for each pipeline stage and hence can tolerate at least one hard error in each pipeline stage. Put together, Shield can tolerate multiple hard errors in its pipeline. Shield uses a critical gate identification algorithm to identify the gates in its pipeline that are most sensitive to soft errors and hardens these critical gates to tolerate soft errors.

4 Guarding Against Hard Errors in Shield

In this section, we briefly summarize the techniques used to tolerate hard errors. First, we present the effect(s) of a hard fault in each individual stage and then we present the technique used to tolerate hard errors in each pipeline stage.

4.1 Effects of Hard Faults in the Pipeline

4.1.1 Routing Computation Stage

A fault in this stage will result in the calculation of an inaccurate or faulty output port. As a result, the packet will be forwarded to an incorrect downstream router. This could result in either increased latency of the packet if an adaptive routing algorithm is used to route packets or deadlock if a deterministic routing algorithm is used to route packets. There is also a potential chance that the downstream router might drop this unexpected packet resulting in a packet retransmission at a later time.

4.1.2 Virtual Channel Allocation Stage

A fault in this stage will result either in the allocation of an incorrect virtual channel at the downstream router or failure to allocate a virtual channel. If an incorrect virtual channel is allocated, when the packet is forwarded to the downstream router, it will be directed to an incorrect virtual channel. If that virtual channel is occupied by another packet, then the incoming packet overwrites the existing packet resulting in data corruption. There is also a potential chance that this new packet might be overwritten by another packet at a later time because of incorrect virtual channel allocation. On the other hand, if a virtual channel has not been allocated at all, then the packet will remain in the router buffer and will fail to progress in its pipeline. This results in serious ramifications in the performance of the network and might also potentially lead to deadlock.

4.1.3 Switch Allocation Stage

A fault in this stage will prevent the packet at an input port of the router access to the input port of the crossbar. In this situation, the packet cannot traverse through the crossbar and is blocked in the router. A blocked packet will be unable to free the virtual channel allocated to it eventually resulting in performance degradation and a potential deadlock situation.

4.1.4 Crossbar Stage

A fault in the crossbar will disrupt the connections between an output port of the router and the input ports of the router. As a result, a packet from any input port will not be able to access the affected output port. Thus, due to the fault, the router will either be operating in a degraded fashion or it needs to be completely bypassed by the ongoing packets.

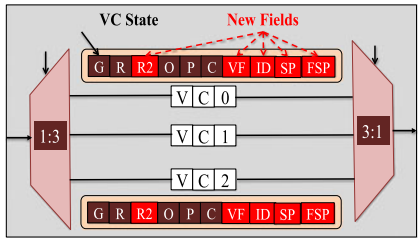


Fig. 4. Proposed input port to provide hard fault tolerance.

4.2 Techniques Used to Tolerate Hard Errors

4.2.1 Routing Computation Stage

The routing protocol used in the NoC determines the architecture of the routing computation unit. In this work, we use dimension order (XY) routing to route packets. XY routing does not require routing tables [40] and hence the silicon footprint of the circuit necessary for performing XY routing is minimal. Thus, we employ a spatial redundant approach where we duplicate the routing computation unit of every input port. This duplicate unit is maintained in power-gated mode as long as the original unit is functional and is powered on upon detection of a fault in the original routing computation unit. Once the duplicate unit is turned on, it continues to be on for the remainder of the time and so we implement the power-gating of the duplicate unit using sleepy transistor method discussed in [16]. Thus, fault tolerance for this stage is achieved via spatial redundancy.

4.2.2 Virtual Channel Allocation Stage

We used a two-stage separable virtual channel allocator for performing virtual channel allocation. As a result, we consider the scenario of fault in both the sub-stages independently. From Fig. 3, we can deduce that in the first sub-stage, each input virtual channel is associated with exact set of arbiters. When an arbiter associated with an input virtual channel is detected to be faulty, the complete set of arbiters associated with that specific input virtual channel is considered faulty. To perform virtual channel allocation, the packet in the affected input virtual channel borrows the arbiter set from another input virtual channel belonging to the same input port. This is possible because all the input virtual channels are associated with exact set of arbiters. By scanning through the 'G' state field, which indicates the state of the virtual channel, the affected virtual channel can identify another virtual channel belonging to the same input port to borrow arbiters from. To facilitate this arbiter sharing, the input port of the router needs to be modified as shown in Fig. 4. This arbiter sharing might incur a latency of one cycle when the affected virtual channel is unable to find a virtual channel whose arbiters are idle.

If an arbiter is faulty in the second sub-stage, it will result in that particular virtual channel at the downstream router not being allocated to any packet in the current router. This fault does not result in blocking the packet at the current

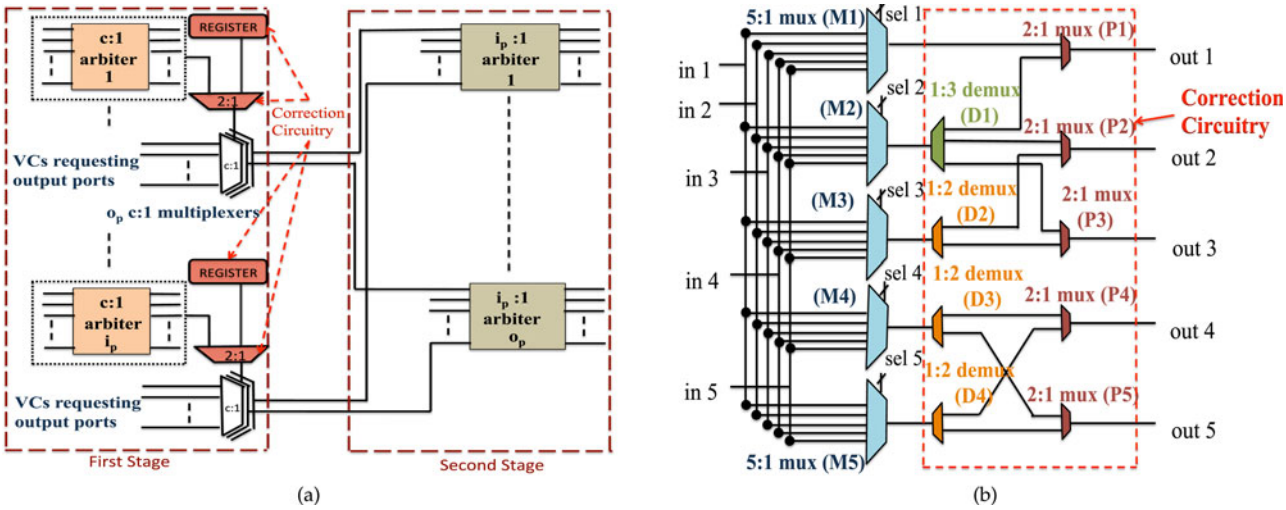


Fig. 5. Hard fault tolerance. (a) Proposed switch allocator. (b) Proposed crossbar.

router because another virtual channel belonging to the required input port at the downstream router can be allocated using the associated non-faulty arbiter. To do this, virtual channel allocation must be repeated, causing an additional latency of one cycle.

4.2.3 Switch Allocation Stage

We used a two-stage separable switch allocator for performing switch allocation. So, we consider the scenario of fault in both the sub-stages independently. From Fig. 3, we can deduce that each input port is associated with an arbiter that determines which virtual channel of the input port gets to compete in the second sub-stage of switch allocation. To overcome the situation of a faulty arbiter, we propose to create a bypass path for each arbiter in the first sub-stage (Fig. 5a). This bypass path is used to choose a virtual channel from an input port when the associated arbiter is faulty. The bypass path always chooses the same input virtual channel as the winner. This is made possible by adding a *2:1 multiplexer* at the output of each arbiter in the first sub-stage (Fig. 5a). One input to this 2:1 multiplexer is the output of the arbiter and the other input comes from a *register* that stores the identification of the default input virtual channel that is always chosen as winner when bypass path is activated. For example, if VC3 is decided to be the default input virtual channel that is always chosen by the bypass path, the identification of VC3 is stored in the register. When the default virtual channel is empty and there are flits in other virtual channels belonging to the same input port, flits from any other virtual channel of the same input port can be *transferred* into default virtual channel using implemented read/write logic. This transferring incurs a latency of one cycle.

If an arbiter is faulty in the second sub-stage, it makes the associated output port unreachable. This can be avoided by forwarding the packet via a secondary path. The existence of this secondary path to reach an output port is described in the next sub-section where we present the fault tolerance in the crossbar.

4.2.4 Crossbar Stage

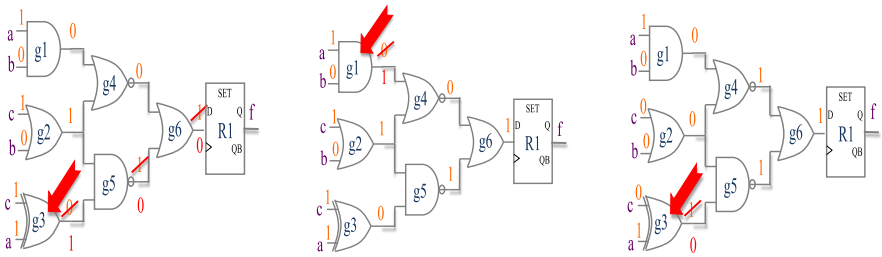
In the generic crossbar (Fig. 3), there exists only one path to reach a specific output port. If this path is faulty, the output port becomes inaccessible. To overcome this problem, we add extra circuitry comprised of *one 1:3 demultiplexer*, *three 1:2 demultiplexers* and *five 2:1 multiplexers*. Fig. 5b shows the modified crossbar. With the help of this additional circuitry, we can observe that there exist two paths to reach every output port. In the fault-free case, the primary path is used by the flits and in the event of a fault in the primary path; the secondary path can be used. The switch allocator generates the select signals to these additional multiplexers and de-multiplexers. This secondary path is also used to provide fault tolerance to the second stage of switch allocator.

We considered each stage of the pipeline independently and employed a technique customized to the stage's functionality that enables the stage to tolerate hard errors. For further details regarding hard error tolerance, please refer to [33]. In the following section, we present the proposed methodology for tackling soft errors.

5 Guarding Against Soft Errors in Shield

Transient faults in a logical circuit are mainly caused by particle strikes [25] on the circuit. Not all particle strikes on a circuit result in a transient fault. If the transient change in the charge, as a result of a particle strike is sufficient enough to cause a bit flip, and if the bit flip is captured by a sequential element, there is a chance for the particle strike to alter the computational result of the circuit. A transient fault is termed as a *soft error* when it is clocked by a sequential element of the circuit. Note that in this work, we focus on soft errors only in combinational logic circuits. We assume that, soft errors in sequential logic circuits such as buffers can be tolerated using techniques based on ECC such as in [44] and hence, we do not discuss fault tolerance regarding sequential circuits further in this paper.

The propagation of a transient fault to an output depends on the three different masking mechanisms inherent in a logic circuit namely, *logical masking*, *latch-window masking* and *electrical masking* [45], [46], [47], [48]. Logical masking arises when an active path from the affected gate to the output port or latch is absent.



(a) **Scenario I** (b) **Scenario II** (c) **Scenario III**

Fig. 6. Example of logical masking.

Latch-window masking occurs when the generated transient from the affected gate arrives at the latch or flip-flop at a time other than during the clock window for latching. Electrical masking occurs when the transient generated by the affected gate weakens as it traverses through the active path from the affected gate to the output port or latches. With technology scaling down, transistors are becoming smaller resulting in reduced electrical masking [49]. Further, due to higher clock rates (faster devices), the latch window masking is decreasing [49]. However, based on the definition, logical masking is completely dependent on the specific circuit. Therefore, among the three masking mechanisms, in this paper we are interested in logical masking and hence we describe it in further detail.

5.1 Logical Masking Analysis

Consider a simple combinational logic circuit such as shown in Fig. 6a. It takes in three inputs (*a*, *b*, *c*) and produces a single output (*f*). It is comprised of 6 gates (*g1*, *g2*, *g3*, *g4*, *g5*, *g6*). The output of *g6* is clocked into register R1 (D flip-flop) and the output of the circuit (*f*) takes the value stored in R1. We evaluate this simple circuit (Fig. 6a) in the following three different scenarios. We assume there is no electrical and latch-window masking in these scenarios.

Scenario I. Consider the input combination ($a = 1, b = 0, c = 1$) for this scenario (Fig. 6a). As a result, the output of the circuit (*f*) is *1*. Let us assume a charged particle hits the output of gate *g3* and flips its output value from *0* (in the fault-free case) to *1*. As a result, the output of gates *g5* and *g6* change to *0*. Hence, the value of *f* changes to *0*. Thus, in this scenario, a particle hit resulted in changing the output of the circuit.

Scenario II. Consider the input combination ($a = 1, b = 0, c = 1$) for this scenario (Fig. 6b). As a result, the output of the circuit (*f*) is *1*. Let us assume a charged particle hits the output of gate *g1* and flips its output value from *0* (in the fault-free case) to *1*. Due to the nature of *g4* (*nor gate*), the bit flip occurred at gate *g1* does not propagate to the output and hence, the output of the circuit (*f*) is unchanged. Thus, in this scenario, the transient fault (particle hit) has been *logically masked*.

Scenario III. Consider the input combination ($a = 1, b = 0, c = 0$) for this scenario (Fig. 6c). As a result, the output of the circuit (*f*) is *1*. Let us assume a charged particle hits the output of gate *g3* and flips its output value from *0* (in the fault-free case) to *1*. Due to the nature of *g5* (*nand gate*), the bit flip occurred at gate *g3* does not propagate to the output and hence, the output of the circuit (*f*) is unchanged. Thus, in this scenario as well, the transient fault (particle hit) has been *logically masked*.

From the three scenarios we just described, we can come to a conclusion that, though a particle hit at a specific gate results in changing the output of that particular gate, whether the output of the entire circuit changes or remains the same (effect of particle hit is logically masked) depends on the gate affected by the hit (Figs. 6a and 6b), as well as the input combination during the hit (Fig. 6c). Hence, the number of gates in a circuit and the total number of possible input vectors for the circuit play a crucial role in calculating the logical masking of a circuit. In the following sub-section, we present a mathematical model that we use to calculate the logical masking of an arbitrary combinational circuit.

5.1.1 Logical Masking Model

Consider an arbitrary combinational circuit with N_G gates, N inputs and T outputs. Therefore, the total number of inputs possible for the circuit is $N_V = 2^N$. Each input combination is a vector of length N and the circuit produces an output vector of length T for the particular input combination. Let M be the average logical masking of the entire circuit. For an arbitrary gate i and arbitrary input vector j, we define $M(i, j)$ as:

- $M(i, j) = 1$, if a transient fault occurs at gate i when the input vector is j and the output vector of the circuit is identical to the output vector of the circuit in the absence of a transient fault.
- $M(i, j) = 0$, if a transient fault occurs at gate i when the input vector is j and the output vector of the circuit is different from the output vector of the circuit in the absence of a transient fault.

Let,

- $p_{G(i)}$ be the probability that the gate i is affected by a transient fault.
- $p_{V(j)}$ be the probability that the input vector is j when the transient fault occurred.

The average logical masking M of the circuit can be calculated as

$$M = \sum_i \sum_j p_{G(i)} p_{V(j)} M(i, j). \quad (1)$$

We assume all gates have equal probability of being affected by a transient fault and hence,

- $p_{G(i)} = 1/N_G$.

We also assume that, all the input vectors are equally probable and hence,

- $p_{V(j)} = 1/N_V$.

The above assumption for equal probability for all input vectors is justified as follows: in the context of an NoC, we assume that all the output ports of an NoC router have equal probability of encountering a packet. Thus, all the input ports of an NoC router have equal probability of receiving a packet. Therefore, all the input vectors for the routing computation stage have equal probability. Based on the description of virtual channel and switch allocation stages (Section 2.2), it can be deduced that arbiters form the fundamental building blocks for both these stages. We use round-robin arbiters in both the stages. Round-robin arbiters perform a fair arbitration [9]. This indicates that all the virtual channels of any input port have equal probability of storing a packet. Thus, the input vectors for both virtual channel allocation and switch allocation stages have equal probability. Since, all the output ports are equally probable, all the input vectors for the crossbar also have equal probability.

Thus, the average logical masking M of the circuit can be rewritten as

$$M = \frac{\sum_i \sum_j M(i,j)}{N_G N_V}. \quad (2)$$

From Equation (2), we can deduce that the value of average of logical masking ranges between 0 and 1 i.e., $0 < M < 1$. Considering a pessimistic approach wherein we assume that electrical masking and latch-window masking do not occur, whether a bit flip at the output of a gate affects the output of the circuit (manifest as soft error) or not solely depends on if the bit flip is logically masked or not. In this particular scenario, the average logical masking value of a circuit can be viewed as a measure of the circuit's ability to tolerate soft errors i.e., the probability (P) of a single transient fault being masked by the circuit. Thus, calculating $1 - P$ gives the probability of a single transient fault in the circuit manifesting as a soft error.

5.1.2 Selective Hardening of Critical Gates

Hardening the gates of a logic circuit is a well-known methodology to make the circuit tolerate transient faults [47], [48], [50]. The process of hardening a gate involves either having a duplicate copy of the gate (architectural modification) or resizing the gate (circuit modification) by increasing the width of the gate's transistors [51], [52]. Employing either of these approaches results in increasing the gate's driving strength and shortens the amplitude of the transient pulses generated by particle strikes [51]. We employ the gate resizing technique for hardening a gate. A gate can be considered immune to transient faults, if it is sized such that, the generation of a pulse with amplitude larger than $V_{dd}/2$ is not possible by the deposited charge as a result of particle strike [50], where V_{dd} is the operating voltage.

Hardening all the gates of a circuit incurs too much cost in terms of the area and power consumption of the circuit. *Selective hardening* [53], [54], [55], which involves hardening only a few selected gates of a circuit, is known to improve the circuit tolerance to soft errors. The goal of selective hardening is to achieve a right balance between the required soft error tolerance and the overhead incurred as a result of hardening [54]. One of the key aspects in achieving the best tolerance to soft errors via selective hardening is to identify which gates to harden. In order to identify the gates of a circuit to be hardened, we use the traditional hardening algorithm presented in [52] as the base and modify it according to our requirements to identify the critical gates of a circuit and to rank the gates according to their criticality. The *criticality* of a gate is determined by its sensitivity to soft errors. The higher the sensitivity of a gate, the greater is its criticality. In the following sub-section we present the details of our critical gate identification algorithm.

5.1.3 Critical Gate Identification Algorithm

In this sub-section, we present the algorithm we use to identify the critical gates of a circuit. The mathematical model presented in Section 5.1.1 is the crux of this algorithm. It takes the circuit description as an input and outputs the list of gates sorted according to their sensitivity towards soft errors. The working of the algorithm can be described as follows in three steps.

Step 1. Based on the circuit description provided as input, the algorithm calculates the number of inputs taken by the circuit and the number of gates in the circuit. Based on the number of inputs, the number of input vectors is calculated.

Step 2. An input vector is applied to the circuit and the correct output of the circuit is computed. Then, the output of a randomly chosen gate is flipped (simulating a charged particle hit) and the output of the circuit is computed again. This computed value is compared to the original output value to determine if the change in the output of the chosen gate has affected the output of the circuit. If it has affected, then the chosen gate's criticality is updated. This process of flipping the output of a gate value and comparing it with the original value is repeated for all the gates in the circuit. Note that at any point of time only one gate's output is flipped.

Step 3. Step 2 is repeated for all possible input vectors to determine the aggregate criticality value of all the gates.

Table 1 shows the pseudo code for the critical gate identification algorithm. Since the algorithm sorts all the gates of a circuit according to their criticality values in a descending order, the higher a gate is in the list, higher is its sensitivity towards soft errors. The number of gates hardened depends upon the area overhead that is accepted. Based on the analysis described in [51], we increase the gate size by *3X* for every gate that is chosen to be hardened. The impact of increasing the size of some gates is presented in Section 7 where we discuss the overhead incurred by the techniques to achieve fault tolerance. In the following section, we perform reliability analysis to estimate the improvement in reliability achieved via employing the discussed techniques.

Note that Equation (2) is applicable when all the input vectors are equally probable. In the event where some input vectors are more probable than the other, a correlation exists between the input and output vectors. Then, the logical masking M of the circuit is calculated as

$$M = \frac{\sum_i \sum_j M(i,j) * p_{V_{(j)}}}{N_G}. \quad (3)$$

TABLE 1
Critical Gate Identification Algorithm

```
function Critical_Gate_Identification (circuit)
begin
   list_input_vector ⇐ input vectors of the circuit
   list_gates ⇐ gates of the circuit

   /* Set the criticality of all gates to 0 */
   for G in 1 to num_gates_circuit
   begin
      list_gates[G].criticality ⇐ 0
   end

   /* Set the value of the counter to 0 */
   counter ⇐ 0

   /* For each input vector */
   for vec in 1 to num_input_vectors
   begin
      Apply input vector list_input_vector[vec] to circuit
      Compute the output of the circuit
      Output1 ⇐ circuit(list_input_vector[vec])
      for G in 1 to num_gates_circuit
      begin
         Flip output of gate list_gates[G]
         Compute the output of the circuit with flipped value
         of list_gates[G]
         Output2 ⇐ circuit(list_input_vector[vec])
         if (Output1 != Output2)
         begin
            /* Increase the gate's criticality */
            list_gates[G].criticality++
         end
         else
         begin
            /* Increment counter */
            counter ⇐ counter+1
         end
         Correct the output of the gate list_gates[G]
      end
   end
   /* Calculate masking of the circuit */
   M = [counter/(num_input_vectors*num_gates_circuit)]*100

   /* Sort list_gates according to criticality of gates */
   SORT (list_gates, Descending)
end
```

However, the critical gate identification algorithm can still be used to identify the most vulnerable gates in the event of correlated input and output vectors. Since, some of the input vectors have higher probability, the logic gates of the circuit that exist in the critical path of these high probability input vectors will have increased probability to be deemed vulnerable than the remaining gates in the circuit.

6 Reliability Improvement Analysis

In this section, we conduct an analysis to quantify the improvement in reliability achieved by Shield in comparison to the baseline unprotected router and other fault tolerant router architectures. Fig. 7 shows the comprehensive view of the Shield router architecture with the redundant routing computation units, proposed input port, proposed switch allocator, proposed crossbar and with selective hardening performed on routing computation, virtual channel allocation and switch allocation circuits (shown in the figure with light rectangular shading). Due to the fundamental difference in the nature and behavior of a transient and a hard fault, we estimate the reliability improvement with respect to soft and hard errors separately.

6.1 Reliability Improvement for Soft Errors

As mentioned in Section 5.1, masking of a circuit (logical masking) plays an important role in determining its vulnerability to soft errors. The higher the masking, the better is the immunity of a circuit to soft errors. Thus, we estimate the reliability improvement of Shield with respect to soft errors by calculating the improvement in logical masking achieved via hardening critical gates in comparison to the logical masking of the baseline unhardened circuitry. Consider a five-input, five-output port NoC router with each input port comprised of four virtual channels. Assume that it is part of an 8×8 NoC that employs dimension order (XY) routing to route packets from source to destination. Fig. 8 shows the path of a flit through the various circuits of the router pipeline. It traverses through a comparator responsible for routing computation stage, four input and 20 input arbiter responsible for virtual channel allocation, couple of five input arbiters responsible for switch allocation and 5 to 1 multiplexer responsible for the crossbar stage.

Using the mathematical model presented in Section 5.1.1, we calculate the masking of all these circuits exclusively. For explanation purposes, we show here how the masking of a 5:1 multiplexer is calculated. Fig. 9 shows a 5:1 multiplexer realized using four 2:1 multiplexers. It has five inputs (data) and a 3-bit select signal (control). We are interested in the select (control) signal because, it decides which input to forward to the output. As can be seen in Fig. 9, the circuit is comprised of four gates (*g1, g2, g3 and g4*). The 3-bit select signal gives us a total combination of eight input vectors. However, since it is a 5:1 multiplexer, we only need five input vectors out of the total eight possible combinations. When we execute the critical gate identification algorithm on this circuit, the algorithm evaluates the circuit a total of 20 ($[number\ of\ input\ vectors * number\ of\ gates] = 5 * 4 = 20$) times. Among these 20 executions of the circuit, the algorithm revealed that the output value computed by flipping the output of a random gate matched the original fault-free output value seven times. This gives the 5:1 multiplexer a logical masking value of $7/20 = 35$ percent. The algorithm also gave as output the list of gates sorted according to their criticality values. The order of the gates in the list is *g4*, *g3*, *g2* and *g1*. Note that, *g1* and *g2* have the same logical masking value. We ranked *g2* higher than *g1* for the sake of explanation. The discussion holds even if *g1* is ranked higher than *g2*. Using the same approach, we execute the critical gate identification algorithm and calculate the masking and the sorted list of gates according to their criticality values for the comparator, four input arbiter, 20 input arbiter and five input arbiter circuits respectively.

In the interest of achieving the right balance between the area overhead incurred as a result of sizing the gates and the soft error tolerance achieved, we chose to take the *two* most critical gates of the mentioned circuits and harden them. Also, since a 5:1 multiplexer has an inherently high logical masking value (35 percent) and a soft error in the crossbar

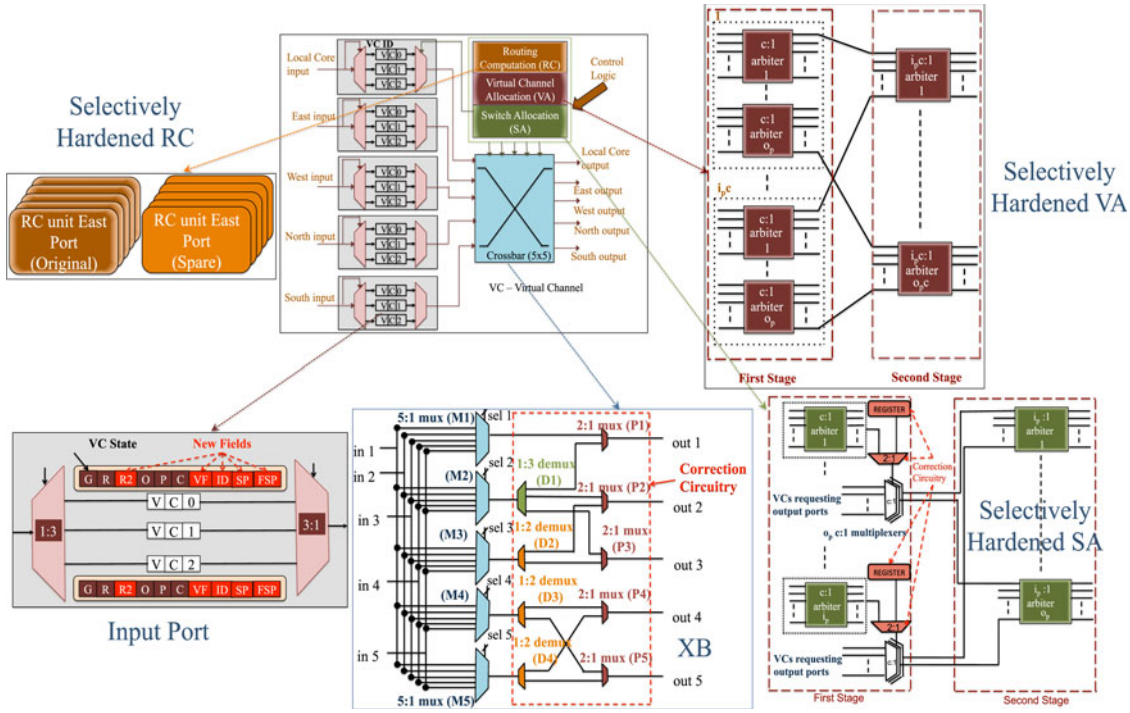


Fig. 7. Component details of a Shield router.

stage is not as severe as a soft error in the remaining three pipeline stages, we do not perform hardening of any gates in the crossbar. In other words, a soft error in crossbar will not result in entire packet being misdirected and hence, we do not perform hardening in crossbar. Therefore, the size of the top two ranked gates of the comparator (comp), four input arbiter (arb4), 20 input arbiter (arb20) and the couple of five input arbiters (arb5) are increased by three times. After the top two ranked gates of all the mentioned components are hardened, we execute the critical gate identification algorithm again on all the components to estimate their updated logical masking values. The logical masking of the entire pipeline ($LM_{pipeline}$) can be calculated as

$$LM_{pipeline} = LM_{comp} * LM_{arb4} * LM_{arb20} * LM_{arb5} * LM_{arb5} * LM_{5:1mux}. \quad (4)$$

As mentioned in Section 5.1.1, we assume that all output ports of a router have equal probability of encountering a packet. This assumption led to the observation that all the input vectors of each pipeline stage have equal probability. Using this assumption, we calculated the average masking of each component in the pipeline stages. Therefore, the average masking calculated for each component is independent of all the other components. Since the probabilities are independent, logical masking of the entire pipeline is calculated as the product of logical masking (probability) of every component in the pipeline.

We calculate the logical masking of the baseline pipeline ($LM_{baseline}$) as well as the hardened pipeline ($LM_{hardened}$) based on Equation (4). We introduce a metric called *Soft Error Tolerance Improvement Factor (SEIF)*, which is defined as the ratio of $LM_{hardened}$ with $LM_{baseline}$:

$$SEIF = \frac{LM_{hardened}}{LM_{baseline}}. \quad (5)$$

Based on the definition of logical masking, we can deduce that the higher the logical masking value, the better is the tolerance to soft errors. So, from the definition of SEIF, a value of SEIF greater than 1 indicates improvement in soft error tolerance compared to the baseline pipeline. With the use of selective hardening, the value of SEIF is 3, indicating that due to hardening certain critical gates, the soft error tolerance of the hardened pipeline has increased by a factor of 3. This improvement achieved via selective hardening incurred an area overhead of approximately 3 percent in comparison to the baseline router.

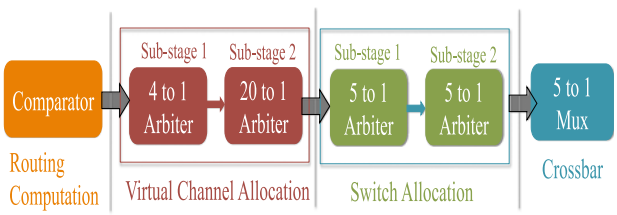


Fig. 8. Path of a flit through the router pipeline.

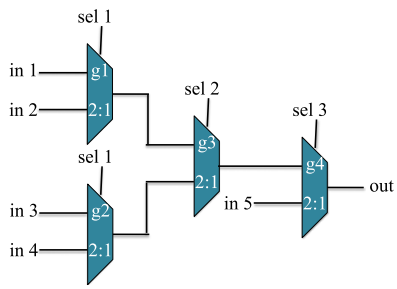


Fig. 9. 5:1 multiplexer realized as a combination of 4 2:1 multiplexers.

TABLE 2
Comparison of Shield Architecture with Other Existing Soft Error Tolerant Architectures

Architecture	Hard Faults	Soft Errors	Operation level in System stack	Reliability	Area	Power
Shield	Yes	Yes	Circuit-level	2 soft errors per RC, VA and SA stages	34 percent	31 percent
Allocation Comparator [39]	No	Yes	Architecture-level	2 soft errors per RC, VA and SA stages	1.2 percent	1.7 percent
Information Redundancy [40]	No	Yes	Architecture-level	2 soft errors per RC, VA and SA stages	9 percent	3 percent

Table 2 presents the comparison between Shield and existing soft error techniques presented in [39], [40]. It can be deduced from the table that the higher area and power overhead of the Shield architecture is due to its ability to tolerate hard errors in addition to the soft errors, whereas the other two architectures can tolerate only soft errors.

6.2 Reliability Improvement for Hard Errors

To quantify the reliability improvement of Shield in comparison to the baseline unprotected router with respect to hard faults, we use Mean Time to Failure and Silicon Protection Factor (SPF) [35] metrics.

To estimate the MTTF of Shield, we calculate the FIT of the baseline circuitry and the correction circuitry using the architectural level reliability-modeling framework proposed in [56]. The reliability framework proposes to calculate the FIT of a circuit based on an approach called as Failure in time of Reference Circuit (FORC). This approach allows the designers to estimate the failure rate of a circuit without delving into low level details regarding the circuit implementation and the specific technology involved. This framework provides mathematical equations to calculate the failure rate of a circuit due to TDDB. The reason for specifically choosing TDDB fault mechanism comes from the fact that, it has no recovery effect on digital circuits unlike other fault mechanisms such as electromigration and negative bias temperature instability [56] and also as technology continues to scale down, it will evolve into one of the major causes for hard faults [57].

Using the provided mathematical equations, we estimate the FIT of the RC, VA, SA and XB stages of a five-input, five-output port router with each input port comprised of four VCs. We estimate the MTTF of the baseline router taking into account the FIT values of the four stages of the pipeline. Then, we also estimate the FIT of the correction circuitry involved in providing protection from hard faults to the pipeline stages. We estimate the MTTF of a Shield router taking into account the FIT values of the four stages of the pipeline as well as the FIT values of the correction circuitry. Comparing the MTTF values of a Shield router and a baseline unprotected router reveals that, Shield's MTTF value is six times the MTTF value of the baseline unprotected router. This indicates that the Shield's reliability value has increased by 500 percent in comparison to the baseline router. The details of this analysis have been presented in [33].

To compare Shield with existing fault tolerant router architectures such as BulletProof [35], Vicis [36] and RoCo [37] from a hard fault viewpoint, we use SPF as a metric. SPF is defined as the ratio of mean number of faults to cause failure and the area overhead incurred by the additional circuitry. This definition suggests that, higher SPF value indicates better reliability. Figs. 10a, 10b and 10c compare the area overhead, mean number of faults to cause failure and the Silicon Protection Factor value of BulletProof, Vicis, RoCo and Shield architectures. Based on these figures, we can deduce that, between the fault tolerant router architectures, Shield has the second lowest area overhead compared to the baseline router and has the highest mean number of faults to cause failure which resulted in a higher SPF value. Thus, we conclude that, the higher SPF value of Shield indicates that it provides better reliability at lower cost. The analysis regarding the mean number of faults to cause failure of Shield has been presented in detail in [33].

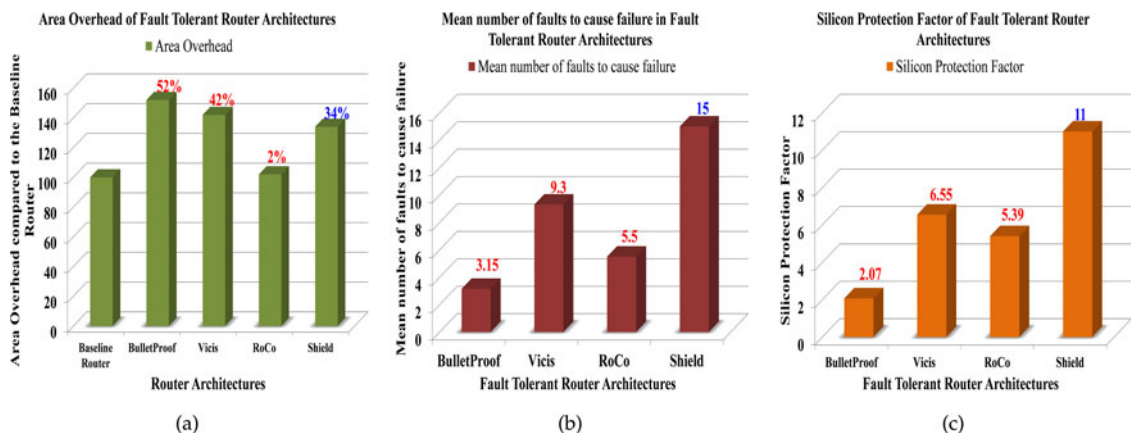


Fig. 10. Comparison of fault tolerant router architectures. (a) Area overhead. (b) Mean number of faults to cause failure. (c) Silicon protection factor.

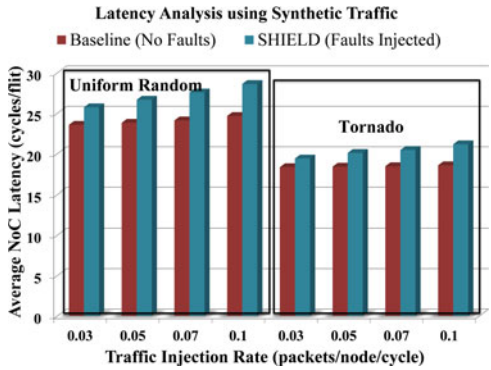


Fig. 11. 4x4 NoC with Shield using synthetic traffic.

7 Performance Analysis of Shield: Area, Power and Critical Path

To evaluate the impact of the correction circuitry on area, power and critical path we developed the baseline and the Shield (Fig. 7) versions of a five-input port, five-output port router with each input port consisting four VCs, where each channel can hold a total of four 32-bit flits, in Verilog and synthesized using Cadence Encounter RTL compiler at 45 nm technology. We also took into account the resized gates and their overhead. Synthesis results reveal that Shield incurs an area overhead of 34 percent (Fig. 10a) and total power overhead of 31 percent with respect to the baseline router.

To find the accurate effect on critical path due to the correction circuitry, we repeated the synthesis process of both the baseline and the modified pipeline stages at varying clock period values to identify the respective clock periods that result in zero slack time. Synthesis results reveal that the critical paths of VA, SA and XB stages have increased by 20, 10 and 25 percent due to the correction circuitry. However, the increase in the critical path of the RC stage is negligible because of employing spatial redundancy. These increased values also take into account the increase in critical path due to selective hardening used to improve soft error tolerance. Since the clock period should be long enough to accommodate all computations, the clock period of Shield is increased by 25 percent in comparison to the baseline unprotected router.

8 Latency Analysis of Shield

To study the impact on latency, we simulate two configurations of NoC comprised of Shield routers using GEM5 [58] simulator. The basic four-stage pipeline within a router is simulated using GARNET [59]. We modified the generic router's pipeline to implement Shield's pipeline.

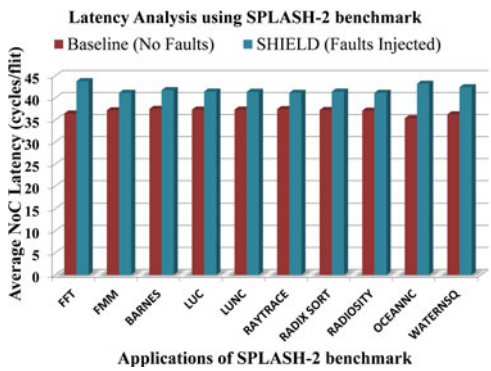


Fig. 12. 8 × 8 NoC with Shield routers using SPLASH-2.

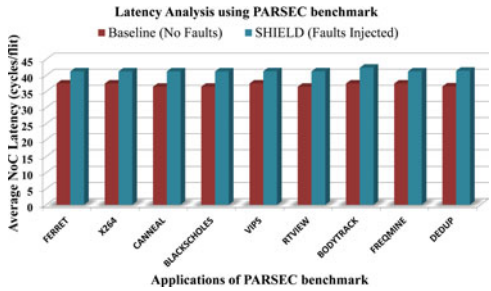


Fig. 13. 8 × 8 NoC with Shield routers using PARSEC.

The first configuration is a 4 × 4 mesh-based NoC on which we inject uniform random and tornado synthetic traffic patterns at varying injection rates (0.03, 0.05, 0.07 and 0.1 packets/node/cycle) and executing the simulation for 10 million cycles. Each simulation is run 10 times and the average value is calculated. The ideal way to simulate faults is to inject faults according to the actual FIT values deduced in [33]. However, the FIT values are very small and the applications need to run for a very long time to be able to inject faults using the FIT values. To accelerate the simulations, we injected a fault into the pipeline stage of a router after 1 million cycles of its operation. For example, in a router, if both routing computation unit and virtual channel allocator have operated for at least 1 million cycles, we inject one fault in the routing computation unit and one fault in the virtual channel allocator. Thus, the router is functioning in the presence of two faults.

Fig. 11 shows the latency of a NoC comprised of Shield routers in the presence of faults in comparison to a NoC comprised of baseline routers with no faults while executing uniform random and tornado traffic patterns. Note that Shield does not incur any additional latency in the absence of hard faults. Therefore, we compare latency of Shield (in the presence of faults) with the baseline router (in the absence of faults). The average increase in latency for uniform random traffic and tornado traffic patterns has been observed to be 13 and 10 percent respectively.

The second configuration is a 8 × 8 mesh-based NoC on which we executed SPLASH-2 [60] and PARSEC [61] benchmarks. Here, to accelerate the simulations, we injected a fault into the pipeline stage of a router after 10 million cycles of its operation. Figs. 12 and 13 show the latency of a NoC comprised of Shield routers in the presence of faults in comparison to a NoC comprised of baseline routers with no faults while executing SPLASH-2 and PARSEC benchmarks respectively. Note that Shield does not incur any additional latency in the absence of hard faults. Therefore, we compare latency of Shield (in the presence of faults) with the baseline router (in the absence of faults). The average increase in latency for SPLASH-2 and PARSEC is 13 and 10 percent respectively.

9 Conclusion

In this work, we presented Shield, an NoC router architecture capable of tolerating both hard and soft errors in the

routing pipeline. Using architectural modifications, we showed that each pipeline stage of Shield could tolerate at least one hard fault. Assuming each pipeline stage is affected by only one hard fault, Shield can tolerate at least four hard faults in the pipeline. Using circuit level modifications we improve Shield's vulnerability to soft errors. We introduced a new metric called Soft Error Improvement Factor and using this metric, we showed that the soft error tolerance of Shield is three times compared to the baseline unprotected router. From the perspective of hard faults, using MTTF metric, we showed that Shield is six times more reliable than a baseline unprotected router and with the help of Silicon Protection Factor, we proved that Shield is at least 1.5 times more reliable than existing fault tolerant router architectures. We conducted hardware synthesis and observed that Shield incurs an area and power overhead of 34 and 31 percent respectively in comparison to the baseline unprotected router. Overall evaluation of results show that, Shield achieves a right balance between reliability improvement achieved and the overhead incurred.

ACKNOWLEDGMENTS

This work was supported by NSF awards 1547034, 1547035, 1547036, 1600820.

REFERENCES

[1] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Jul./Aug. 1999.
[2] S. Borkar, "Thousand core chips: a technology perspective," in *Proc. IEEE Design Autom. Conf.*, 2007 pp. 746–749.
[3] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. IEEE Design Autom. Conf.*, 2001, pp. 684–689.
[4] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *IEEE Comput.*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
[5] S. Kumar, A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2002, pp. 105–112.
[6] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.
[7] International Technology Roadmap for Semiconductors, 2011.
[8] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," in *Proc. IEEE NorChip Conf.*, vol. 31, 2000, pp. 166–173.
[9] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA, USA: Morgan Kaufmann, 2003.
[10] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-Tile 1.28TFLOPS network-on-chip in 65nm CMOS," in *Proc. IEEE Int. Solid State Circuits Conf.*, 2007, pp. 98–589.
[11] D. Park, R. Das, C. Nicopoulos, J. Kim, V. Narayanan, R. Iyer, and C. R. Das, "Design of a dynamic priority-based fast path architecture for on-chip networks," in *Proc. IEEE Symp. High Perform. Interconnects*, 2007, pp. 15–20.
[12] A. Kumar, L. S. Peh, P. Kundu, and N. K. Jha, "Express Virtual Channels: Towards the Ideal Interconnection Fabric," in *Proc. Int. Symp. Comput. Archit.*, 2007, pp. 150–161.
[13] D. Park, S. Eachempati, R. Das, A. K. Mishra, Y. Xie, V. Narayanan, and C. R. Das, "MIRA: A multi-layer on chip interconnect router architecture," in *Proc. Int. Symp. Comput. Archit.*, 2008, pp. 251–261.
[14] A. K. Kodi, A. Sarathy, and A. Louri, "Adaptive channel buffers in on-chip interconnection networks—A power and performance analysis," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1169–1181, Sep. 2008.
[15] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Aergia: Exploiting packet latency slack in on-chip networks," in *Proc. Int. Symp. Comput. Archit.*, 2010, pp. 106–116.
[16] L. Chen and T. M. Pinkston, "NoRD: Node-router decoupling for effective power-gating on on-chip routers," in *Proc. IEEE/ACM Int. Symp. Microarchit.*, 2012, pp. 270–281.
[17] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov./Dec. 2005.
[18] M. Pirretti, G. M. Link, R. R. Brooks, V. Narayanan, M. Kandemir, and M. J. Irwin, "Fault tolerant algorithms for network-on-chip interconnect," in *Proc. IEEE Annu. Symp. VLSI*, 2004, pp. 46–51.
[19] S. Oussalah and F. Nebel, "On the oxide thickness dependence of the time-dependent-dielectric breakdown," in *Proc. IEEE Electron Devices Meeting*, 1999, pp. 42–45.
[20] C. E. Blat, E. H. Nicollian, and E. H. Poindexter, "Mechanism of negative-bias-temperature instability," *J. Appl. Phys.*, vol. 69, no. 3, pp. 1712–1720, 1991.
[21] G. V. Groeseneken, "Hot carrier degradation and ESD in submicrometer CMOS technologies: How do they interact?" *IEEE Trans. Device Mater. Rel.*, vol. 1, no. 1, pp. 23–32, Mar. 2001.
[22] R. Barsky and I. A. Wagner, "Electromigration-dependent parametric yield estimation," in *Proc. IEEE Int. Conf. Electron., Circuits Syst.*, 2004, pp. 121–124.
[23] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Trans. Electron Devices*, vol. ED-26, no. 1, pp. 2–9, Jan. 1979.
[24] G. A. Sai-Halasz, M. R. Wordeman, and R. H. Dennard, "Alpha-particle-induced soft error rate in VLSI circuits," *IEEE Trans. Electron Devices*, vol. ED-29, no. 4, pp. 725–731, Apr. 1982.
[25] J. F. Ziegler, "Terrestrial cosmic ray intensities," *IBM J. Res. Develop.*, vol. 42, no. 1, pp. 117–140, 1998.
[26] K. J. Kuhn, "Reducing variation in advanced logic technologies: Approaches to process and design for manufacturability of nanoscale CMOS," in *Proc. IEEE Electron Devices Meeting*, 2007, pp. 471–474.
[27] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 818–831, Jun. 2005.
[28] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blauww, "A highly resilient routing algorithm for fault-tolerant NOCs," in *Proc. Conf. Design, Autom. Test Eur.*, 2009, pp. 21–26.
[29] A. DeOrio, L. S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *Proc. Int. Conf. Parallel Archit. Compilation Tech.*, 2011, pp. 298–309.
[30] S. Lin, J. Shi and H. Chen, "Designing cost-effective network-on-chip by dual-channel access mechanism," *J. Syst. Eng. Electron.*, vol. 22, no. 4, pp. 557–564, 2011.
[31] D. DiTomaso, A. K. Kodi, and A. Louri, "QORE: A fault-tolerant network-on-chip architecture with power-efficient quad function channel (QFC) buffers," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2014, pp. 320–331.
[32] P. Poluri and A. Louri, "Tackling permanent faults in the network-on-chip router pipeline," in *Proc. Int. Symp. Comput. Archit. High Perform. Comput.*, 2013, pp. 49–56.
[33] P. Poluri and A. Louri, "An improved router design for reliable on-chip networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2014, pp. 283–292.
[34] L. S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2001, pp. 255–256.
[35] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky, "Bulletproof: A defect-tolerant CMP switch architecture," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2006, pp. 5–16.
[36] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester, "Vicis: A reliable network for unreliable silicon," in *Proc. IEEE Design Autom. Conf.*, 2009, pp. 812–817.
[37] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. S. Yousif, and C. R. Das, "A gracefully degrading and energy-efficient modular router architecture for on-chip networks," in *Proc. Int. Symp. Comput. Archit.*, 2006, pp. 4–15.
[38] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston, "A lightweight fault-tolerant mechanism for network-on-chip," in *Proc. IEEE/ACM Int. Symp. Netw.-on-Chip*, 2008, pp. 13–22.
[39] D. Park, C. Nicopoulos, J. Kim, V. Narayanan, and C.R. Das, "Exploring fault-tolerant network-on-chip architectures," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, 2006, pp. 93–104.

[40] Q. Yu, M. Zhang, and P. Ampadu, "Exploiting inherent information redundancy to manage transient errors in NoC routing arbitration," in *Proc. IEEE/ACM Int. Symp. Netw.-on-Chip*, 2011, pp. 105–112.
[41] M. Ali, M. Welzl and S. Hessler, "A fault tolerant mechanism for handling permanent and transient failures in a network on chip," in *Proc. Int. Conf. Inf. Technol.*, 2007, pp. 1027–1032.
[42] M. Ali, M. Welzl, S. Hessler, and S. Hellebrand, "An efficient fault tolerant mechanism to deal with permanent and transient failures in a network on chip," *Int. J. High Perform. Syst. Archit.*, vol. 1, no. 2, pp. 113–123, 2007.
[43] C. Feng, Z. Lu, A. Jantsch, M. Zhang, and Z. Xing, "Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 6, pp. 1053–1066, Jun. 2013.
[44] M. Srinivasan, T. Theocharides, V. Narayanan, M. J. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks on chips," *IEEE Design Test Comput.*, vol. 22, no. 5, pp. 434–442, Sep./Oct. 2005.
[45] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson, "On latching probability of particle induced transients in combinational networks," in *Proc. Int. Symp. Fault-Tolerant Comput.*, 1994, pp. 340–349.
[46] L. W. Massengill, A. E. Baranski, D. O. Van Nort, J. Meng, and B. L. Bhuva, "Analysis of single-event effects in combinational logic-simulation of the AM2901 bitslice processor," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2609–2615, Dec. 2000.
[47] K. Mohanram and N. A. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," in *Proc. Int. Test Conf.*, 2003, pp. 893–901.
[48] Q. Zhou and K. Mohanram, "Cost-effective radiation hardening technique for combinational logic," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2004, pp. 100–106.
[49] P. Shivakumar, M. Kistler, S. W. Kecker, D. Burger, and L. Alvisis, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, 2002, pp. 389–398.
[50] Q. Zhou and K. Mohanram, "Gate sizing to radiation harden combinational logic," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 25, no. 1, pp. 155–166, Jan. 2006.
[51] V. Srinivasan, A. L. Sternberg, A. R. Duncan, W. H. Robinson, B. L. Bhuva, and L. W. Massengill, "Single-event mitigation in combinational logic using targeted data path hardening," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 6, pp. 2516–2523, Dec. 2005.
[52] D. J. Palframan, N. S. Kim, and M. H. Lipasti, "Precision-Aware Soft Error Protection for GPUs," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 49–59.
[53] R. R. Rao, D. Blaauw, and D. Sylvester, "Soft error reduction in combinational logic using gate resizing and FlipFlop selection," in *Proc. Int. Conf. Comput.-Aided Design*, 2006, pp. 502–509.
[54] I. Polian and J. P. Hayes, "Selective hardening: Toward cost-effective error tolerance," *IEEE Design Test Comput.*, vol. 28, no. 3, pp. 54–63, May/Jun. 2011.
[55] S. N. Pagliarini, G. G. dos Santos, L. A. de B Naviner, and J. F. Naviner, "Exploring the feasibility of selective hardening for combinational logic," *Microelectron. Rel.*, vol. 52, no. 9, pp. 1843–1847, 2012.
[56] J. Shin, V. Zyuban, Z. Hu, J. A. Rivers, and P. Bose, "A framework for architecture-level lifetime reliability modeling," in *Proc. IEEE/IFIP Conf. Dependable Syst. Netw.*, 2007, pp. 534–543.
[57] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The impact of technology scaling on lifetime reliability," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, 2004, pp. 177–186.
[58] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Archit. News*, vol. 39, pp. 1–7, 2011.
[59] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proc. Perform. Anal. Syst. Softw.*, 2009, pp. 33–42.
[60] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. Int. Symp. Comput. Archit.*, 1995, pp. 24–36.
[61] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implementations," in *Proc. Int. Conf. Parallel Archit. Compilation Tech.*, 2008, pp. 72–81.

Pavan Poluri received the bachelor's and master's degrees in the field of computer science in 2007 and 2009 respectively. He is currently working toward the PhD degree at the High Performance Computing Architectures and Technologies Lab, The University of Arizona. His research interests include computer architecture, network-on-chip, reliability, and modeling and simulation. He is currently a student member of the IEEE.

Ahmed Louri received the PhD degree in computer engineering from the University of Southern California in 1988. He is currently a full professor of electrical and computer engineering at the University of Arizona, and the director of the High Performance Computing Architectures and Technologies Laboratory (www.ece.arizona.edu/~hpcat). His research interests include computer architecture, networks-on-chip, parallel processing, power-aware parallel architectures, and optical interconnection networks. He is a fellow of the IEEE and a member of the OSA.