Micro-Architectural support for High Availability of NoC-based MP-SoC

Ritika Singh
Indian Institute of Science
Bangalore, India
ritikasingh@iisc.ac.in

Shashank Vijaya Ranga
National Insitute of Technology Karnataka
India
shashankvijayaranga@gmail.com

Swarali Patil
BITS Pilani
India
swaralip20@gmail.com

Madhava Krishna
Morphing Machines Pvt. Ltd
Bangalore, India
madhav@morphingmachines.com

Mitsu Mehta
Indian Institute of Science
Bangalore, India
mitsumehta@gmail.com

Anoop M N
Indian Institute of Science
Bangalore, India
mnanoop2014@gmail.com

S K Nandy
Indian Institute of Science
Bangalore, India
nandy@iisc.ac.in

Chandan Haldar
Morphing Machines Pvt. Ltd
Bangalore, India
chandan@morphing.in

Ranjani Narayan
Morphing Machines Pvt. Ltd
Bangalore, India
ranjani.narayan@morphing.in

Francois Neumann
Safran Electronics and Defense
Paris
francois.neumann@safrangroup.com

Philippe Baufreton
Safran Electronics and Defense
Paris
phillippe.baufreton@safrangroup.com

***Abstract*—In this paper, we focus on increasing the availability of Multi-Processor System on Chip (MP-SoC) for executing user applications, even when some components of the system are faulty. A Network-on-Chip (NoC) provides high bandwidth communication substrate for the multitude of components/modules in such MP-SoCs. Health of such MP-SoC, and hence its availability, is largely dependent on the health of the NoC. We consider an NoC comprising a bi-directional toroidal mesh interconnection of routers. We use a distributed built-in-self-test to identify faulty communication links. We use information so obtained to determine healthy subsystems that can be made available for executing user applications. This feature is key for enhancing availability of MP-SoCs. We realize this feature as a micro-architectural enhancement in MP-SoC that incurs an insignificant hardware overhead of less than 2%. Latency incurred for analyzing availability of MP-SoC is also insignificant. We functionally validate our proposal by emulating the system on a FPGA device and demonstrate increase in availability of the MP-SoC.**

***Index Terms*—Network-on-Chip,MP-SoC,System Availability**

I. Introduction

There is a growing need for massive and accelerated computing in sectors spanning targeted medicines, brain simulation, particle simulation, next generation avionics, deep learning, etc. These domains of computing need uninterrupted services for long periods of time. Multi-Processor System-on-Chips (MP-SoCs) like, Epiphany [16], REDEFINE [12], Kalray [15], TILE64 [17], KiloCore [18] are some of the architectures in the frontier for accelerated embedded computing. Scalability is not only a desired quality, but a necessary attribute of a MP-SoC that is used in the above-mentioned domains. A Network-on-Chip (NoC) is often used as the interconnection network to connect various components of a MP-SoC, (viz., processing cores, memory modules, orchestration units etc.) and a control unit/host. NoC gracefully scales as the number of modules (to be interconnected) grows. NoC comprises of routers interconnected by a particular topology, in which a routing algorithm is implemented to transport information among modules.

Systems fault due to both hard and soft errors. While soft errors are transient in nature, hard errors are firm, and affect the availability of a system. Increasing availability of system despite hardware faults is the key to using these architecture in the said domains. Hard errors in system components/modules that are bound to a workflow can be overcome only if the workflow is realized on alternate components/modules that are functionally equivalent and fault-free. Systems with such faults can be made available for user applications if these faults can be circumvented, and healthy parts of systems are used for user applications. This might result in degraded performance; however user applications will continue to run with this caveat.

NoC is the backbone of MP-SoCs. Prior to program execution, NoC is used for loading instructions and data from control unit/host to compute and memory units. During program execution NoC is used to transport data

978-1-7281-0649-6/19/$31.00 ©2019 IEEE

among compute and memory units, and at the end of program execution NoC is used to pass back results to control unit/host. For uninterrupted usage of a system, it is therefore crucial to monitor the health of the NoC, isolate faults in the NoC and make healthy parts of the system available to user applications. Our objective in this work is exactly this. For purposes of this proposal, we take a generic MP-SoC as an example, in which routers are interconnected by bi-directional toroidal mesh topology. Each router is connected to processing cores, memory modules and other components of the MP-SoC. Our proposal is applicable to any MP-SoC that has an alternate topology of interconnection that follows certain discipline (of deterministic routing) to transport information among its components.

This paper is organized as follows: In section II we provide a summary of similar work from existing literature. In section III we provide a generic description of an MP-SoC architecture, mention how user applications are launched for execution on the MP-SoC and the importance of NoC in such an architecture. In section IV we talk about the methodology we follow to increase availability of MP-SoC. In this section we give details about a distributed algorithm to collect health of links (of NoC), and a way to use this information to recognize healthy sub-regions that can be used for application kernels, thus providing a method to increase availability of MP-SoC even when there are some faulty links in the NoC. In section V we provide implementation details and measure hardware needed to increase availability of MP-SoC. We draw conclusions of this proposal in section VI.

II. Previous Work

Over the past years, many authors have presented different test strategies for Network-on-chip. NoC has various components of the communication infrastructure that need to be tested individually.

Grecu et al in [1] propose a Built-In-Self-Test (BIST) for Network-on-Chip Interconnect Infrastructures, that focuses on testing the inter-switch links of the chip by progressive reuse of NoC infrastructure. The basic building blocks involved include a test data generator which launches the test vectors and a test error detector which samples and compares the received data. Grecu et al in another work [2] propose a similar method that also exploits the inherent parallelism of the data transport mechanism to reduce the overall test time, but addresses testing of both switches and inter-switch links. Raik et al in [3] talk about testing configurations for functional switch faults but like most of the previous work on testing, the focus is on testing the faults inside the switches. The testing method used in [4] by Mohammad Hosseinabady et al broadcasts the test vectors throughout the network. The algorithm proposed follows the scan-based methodology for testing the switches. In fact, majority of the previous work in this field involves scan-based techniques like the ones mentioned in [5], [6]. However, increasing availability of systems does not seem to be the focus of the above-mentioned endeavors. A strategy to online testing has been provided in [7] which involves a distributed test vector storage. Since the test vectors are not applied from a centralized location, it helps in reducing the test delivery latency. However, it is not clear how responses are made use of in decision making.

Peterson et al in [8] present a BIST strategy for testing the NoC interconnect network having mesh topology. In their method, the BIST modules in each Network Interface perform the testing of the datapath and the control logic. It is left to the operating system to analyze the information, identify portions of the fabric which are fault-free, and isolate the same. However the strategy for analyzing this information has not been clearly outlined. While our approach to exploit BIST for availability analysis is similar to their design, we propose an extendible BIST for NoC for very large MP-SoCs deployed in safety critical applications. Additionally, our scheme as elaborated in section IV, incurs smaller overhead in identifying healthy sub-regions even when there are faulty NoC links. In [9], a self monitoring and self-reconfiguring router is proposed.The self-monitoring unit runs BIST at periodic intervals and if a faulty component is detected, it identifies if the graph corresponding to the NoC is still connected. The self-reconfiguration unit updates the routing tables such that the NoC is deadlock free in its new topology. In our work, faults are detected through a testing phase either at power-on, or at behest of a user; Since we follow a dimension-ordered routing in the NoC, the need to identify a self-connecting graph does not arise. To that extent, our scheme adopts a simple strategy, whereby the need for distributed intelligence unit is obviated. Our objective is to suggest a scalable scheme for very large MP-SoCs, so as to increase availability of the system for critical applications, even under faulty conditions.

In [10] the utilization of available NoC link bandwidth is performed. Here, the aim is to determine availability at the link level, whereas our aim is to make system available at the NoC level. They accept partially faulty links by employing flit serialization and discard a link when number of faults are greater than some defined threshold in a link. However, we consider a link as unhealthy even when one of its bits is faulty.

In [11], a new fault tolerant routing algorithm (deterministic and distributed) for NoC is introduced which bypasses the high number of faulty links and can dynamically handle faults. In this algorithm, a faulty link is discovered and marked. Routing tables are updated to steer the traffic along the non-faulty links. However, we do not change the routing algorithm. We aim to discover a faulty link and mark a region which uses that link for communication as unavailable. We sacrifice some healthy resources in order to avoid overheads involved in changing routing algorithm.

III. Architecture of a generic MP-SoC

In this section we provide high level architectural details of a generic MP-SoC. This will help us set the stage for recognizing faults in the NoC of a MP-SoC, isolate faulty parts, and make available healthy subsystems of the MP-SoC for executing user applications.

Figure 1 is an illustrative example of a MP-SoC. The methods and approaches described in the following for increasing the availability are not limited to the MP-SoC described in figure 1, and can be easily applied to any MP-SoC that relies on a NoC as the communication infrastructure. The example MP-SoC has a NoC comprising $n \times m$ number of routers; in this illustration, $6 \times 7 = 42$ routers are connected by a bi-directional toroidal mesh topology. Each router, as shown in the figure is connected to four others routers by full duplex links. Each router in addition, has ingress and egress links, which connect to a Compute Resource (CR) through a Network Interface (NI). Note that each CR has a NI, but in order to maintain clarity of figure, we have shown only one CR with NI. For the objectives of this paper it is not important to detail the internals of a CR; without loss of information, we assume that each CR has certain compute and memory capacity, and computations within each CR are orchestrated by a local orchestrator. All the CRs together with the NoC form the execution engine. A control unit indicated in the figure is the interface by which the execution engine is connected to a host. Often, in order to maintain uniformity of design, the control unit is connected to a router using the ingress and egress links, much the same way as a CR. We term the column of routers, one of which is connected to the control unit, as gateway routers. Sinks in figure 1 indicate that neither CR nor control unit is connected to the NI of the routers. Example MP-SoC shown in this figure is inherently scalable to hundreds of resources. Note: It is not an architectural restriction to have a single control unit connected to the host. In order not to clutter the figure (and text), we have indicated a single control unit in the figure.

User application programs run on the host; parts of these applications (termed as kernels) that need accelerated execution are directed to execution engine. When the control unit receives requests for kernel execution from the host, it identifies the number of resources needed for successful execution of a kernel, allocates and book-keeps resources assigned. While allocating resources to a kernel, the control unit gathers together physically proximal CRs and creates a sub-region that is entirely devoted to the kernel (for the period of execution of kernel); refer to Figure 2 for example sub-regions indicated by hatched rectangles. The results of kernels are sent back to the host by the control unit. Architecturally such a MP-SoC ensures safe and secure execution of kernels by restricting intra-kernel communication (among the CRs allocated to a kernel) to within a sub-region. Our aim is to make

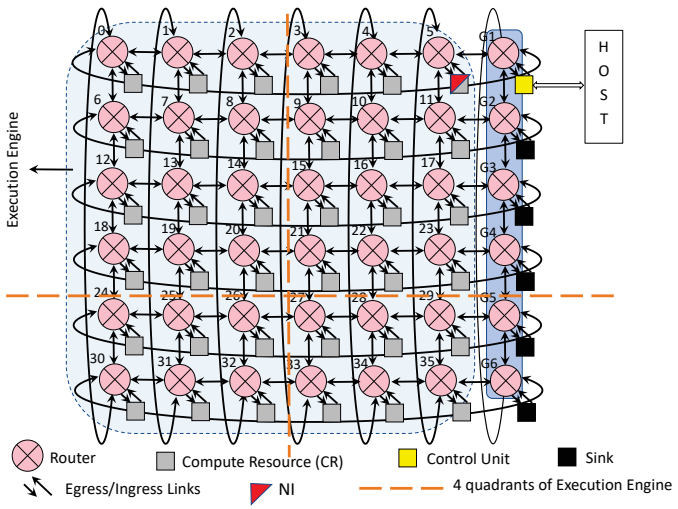

Fig. 1: An example MP-SoC
Note: Each CR is connected to a router through NI
We have shown only one CR with NI in order to maintain clarity of figure

available healthy parts of the system to user applications, and continue to provide safe-and-secure computing regions even if there are certain faults in the system.

Figure 2 zooms into a portion of figure 1. A packet sent from a source (control unit or CR) is injected into the NoC using the ingress link. A packet finds its way to the destination by hopping through intervening routers by following a shortest-path deterministic (X-Y routing) algorithm. And so, in the worst case, a packet hops through $(\lfloor m/2 \rfloor + \lfloor n/2 \rfloor)$ number of routers to reach its destination, i.e., in the worst case, a packet traverses routers in the periphery of a quadrant (indicated by dashed orange lines in figure 1) to reach its destination. Upon reaching destination, the packet is ejected by the router to the CR or control unit using the egress link.

Packets that are produced and consumed during execution of a kernel use inter-router and NI links within a sub-region (and do not use links outside of a sub-region). Continuous red lines in figure 2 indicate the intra-kernel communication of kernel 2. During the launch and end of execution of a kernel, inter-router links outside of a sub-region (indicated by continuous blue and purple lines in figure 2) are also used. However, NI links outside the sub-region are not used for this communication. Therefore as long as there exist paths from a sub-region to the control unit (and vice-versa), a few faulty NI links outside the sub-region do not prevent kernels from getting executed within available sub-regions.

IV. Methodology

We achieve our objective of increasing the availability of a MP-SoC by identifying "healthy" sub-regions (even when there exist some faulty links) in a systematic way by the following:

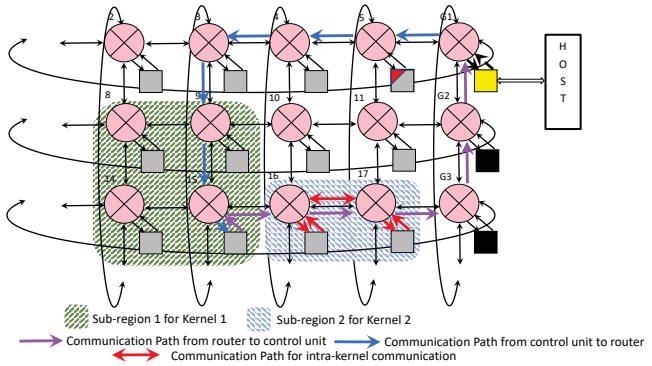

Fig. 2: Sub-Regions Allocated to Kernels
Sub-region 1 indicated by hatched green rectangle (comprising resources connected to routers 8, 9, 14 and 15) is allocated to Kernel 1
Sub-region 2 indicated by hatched blue rectangle (comprising resources connected to routers 16 and 17) are allocated to Kernel 2

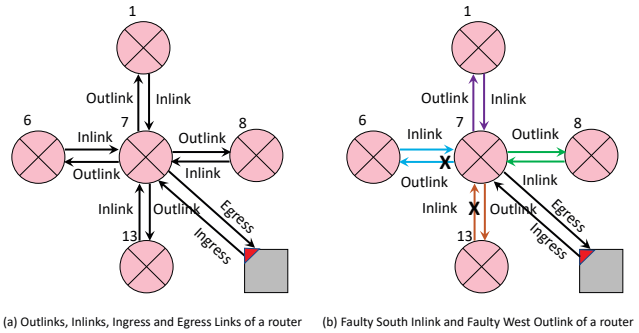

Fig. 3: Outlinks, Inlinks, Ingress and Egress Links of a Router

- Assessment and Consolidation of health of links
- Reachability Analysis
- Faulty link Analysis
- Determination of Availability of Healthy sub-regions

Algorithm used for recognizing healthy sub-regions needs to be deterministic. This is best done when there are no packets flowing in the NoC due to execution of kernels on the execution engine. Therefore, this analysis is performed at power-on. If request for this analysis comes from host (acting at behest of user) at some instant other than power on, then the analysis is performed only after currently executing kernels complete their execution.

A. Assessment and Consolidation of health of links

Primary objective of this phase is to assess health of links connecting routers and health of ingress/egress links. A link is considered faulty when information transferred over it does not reach the destination correctly. Each router assesses health of links that connect neighbouring routers and its NI by sending/receiving stimuli/responses over these links, and consolidating information regarding health of each link. We refer to stimulus and response packets as "ping" and "pong" packets respectively. Our intent is not only to check if information sent reaches destination, but also to determine if there are "stuck-at" faults (see [19]) in a link. At the end of this process, each router sends the consolidated information to the control unit. In order that we assess health of an inter-router link without ambiguity, we refer to the link as an "outlink" when it is used for packets going out from a router (to a neighbouring router), and the term "inlink" when it is used for packets coming in to a router (from a neighbouring router). Egress link is used for packets sent from router to NI, and ingress link is used for packets received from NI - refer to figure 3(a).

Assessment of health of inter-router links: Each router maintains status of outlinks and inlinks in all directions, and updates the status as it probes its neighbours. Status of links are updated to indicate healthy or faulty according to stimuli sent and responses received by routers. We have taken router 7 as an example in figure 4. Each router initializes health of all inlinks and outlinks in all 4 directions (east, south, west, and north) to a default value of "faulty" indicated by 0's in figure 4. Status of inlinks and outlinks of east, south, west and north are indicated by green, red, blue and purple colours respectively.

All routers start assessment by sending stimulus packets in their respective east direction. Stimulus packets comprise 3 ping packets - the first ping packet aids in detecting a defective link. The 2nd and 3rd ping packets help in detecting stuck-at faults - 2nd ping packet comprises all 0's, and the 3rd ping packet contains all 1's. Only when a destination router (router 8 which is to the east in this example) receives all 3 ping packets correctly (within a specified time interval), it sends back a response/pong packet to the source router. Upon receiving a pong packet (within a pre-determined time interval), the source router is sure that its outlink is not faulty and updates status of outlink in the east direction as healthy (indicating that the outlink that connects routers 7 and 8 is healthy and devoid of stuck-at faults). Following this, neighbour to the east (router 8), sends stimulus packets (comprising 3 ping packets) to router 7. Only upon receiving these packets correctly (within a specified time interval), the router (router 7) sets the inlink in the east direction as healthy, and sends a pong packet to the router in the east direction. All updates of status of east outlink and inlink of router 7 are captured in figure 4 in green colour. Corresponding updates of status of west outlinks and inlinks of router 8 are shown in blue colour in figure 4.

All routers in the MP-SoC initiate the assessment of health of links in the east direction simultaneously. Therefore, as router 7 starts probing its east neighbour, router 8 starts probing its east neighbour (router 9 - see figure 1),

TABLE I

Router 7	N	S	E	W
Outlink	1	0	1	0
Inlink	1	0	1	1

and updates the status of its east outlink/inlink according to responses it receives from its east neigbour. Since we do not discuss the actual status of east links of router 8 in this figure, the updated status of these links are represented by 'x' (x will be set to 1 or 0 depending on the health of east links of router 8).

And in parallel, router 6 (the west neighbour of router 7) starts probing its east neighbour (which is router 7). Figure 4 shows these probes and responses, and updates to the health of west links of router 7 indicated by blue colour in figure 4. Corresponding east outlinks and inlinks of router 6, and updates to health of these links of router 6 are shown in figure 4. It is important to note that no false positives are reported using this method of sending stimuli and receiving responses on outlinks and inlinks.

As an illustration, we have shown a faulty west outlink of router 7 - refer figure 3(b). Router 6 sends the 3 ping packets to its east neighbour, router 7. These pings reach router 7 correctly and router 7 marks its west inlink as healthy; however due to faulty west outlink of router 7, the pong packet that router 7 sends as a response, does not reach router 6. Therefore, router 6 does not update its east outlink as healthy. Router 7 sends 3 ping packets to its west neighbour (router 6), which again, due to fault west outlink of router 7, do not reach router 6. Therefore, router 6 does not send a response back to router 7, which results in maintaining status of west outlink of router 7 and east inlink and outlink of router 6 as faulty.

This procedure of sending stimuli and receiving responses is repeated by the source router to assess the health of all inter-router outlinks and inlinks in south, west and north directions. Since sending stimuli and receiving responses in all the directions is similar to the procedure described above, we have not described it in this paper. As an example, status of inlinks and outlinks of a router (router 7) with a faulty west outlink and faulty south inlink is shown in Table I.

Note : Irrespective of the health of the NI link, a router with healthy outlinks and inlinks can act as a pass-through router. For example, router 3 (refer to figure 2) with an unhealthy NI link can act as pass-through router for packets originating from the control unit destined to router 9. And, a router with some faulty outlinks or inlinks can be used as pass-through router. For example, router 4 (refer to figure 2) with a faulty west inlink can act as a pass-through router for packets originating from the control unit destined to router 15.[1]

[1]Note : Links close to control unit are crucial. Faults in one of these links marks a quadrant of execution fabric as unavailable.

Assessment of health of ingress/egress links: A router with a faulty ingress or a faulty egress link cannot be used for executing a kernel, since CR connected to it cannot be reached through some other links. Therefore the method of assessing health of ingress/egress links is simpler than assessing inter-router links. A router sends 3 pings (similar to the pings that a router sends on the inter-router links) on the egress link. If these 3 pings reach the NI as expected within a pre-determined time interval, then it is an indication that the egress link is healthy. Following this, NI sends 3 pongs on the ingress link. Reception of 1st pong by the router indicates healthy egress link, and correct reception of 2nd and 3nd pongs indicates healthy ingress link. At the end of this process, each router sends the status of outlinks and inlinks (in all 4 directions), and status of ingress/egress link to the control unit, and an information packet containing 0's in the bits corresponding to status information in the first packet. Each router sends this information to the control unit using the X-Y routing algorithm, at a pre-determined time. Note: These pre-determined times are computed in such a way that information sent by one router is not over-written by information sent by another router. Pseudo-code of this phase is shown in Algorithm 1.

Algorithm 1 Assessment of Health of Links

```
/* All routers do the following in parallel */
Initialize status of outlinks and inlinks in all four directions to unhealthy
for Direction d = east, south, west to north do
    Send Ping to router in direction d
    if Pong received from router in direction d then
        Update Outlink of direction d as healthy
    end if
    if Ping received from router in direction d then
        Send Pong to router in direction d
        Update Inlink of direction d as healthy
    end if
end for
Send updated status to the control unit
```

Consolidation of health of links: The control unit initializes all links to be unhealthy, and it updates status of a link only when it receives updated information from routers. Therefore even in case updated information from a router does not reach the control unit (due to some faulty links in the path from router to control unit), status of health of links is not falsely reported as being healthy. Status of every inter-router link is reported by 2 routers which share the link. If all the links are fault-free, the control unit will get two responses from each router sharing the link. This is necessary for the analysis because even when one router is not able to send its information to the control unit due to faulty links along the path, the adjacent router's information may reach control unit, and thus the link status can be inferred. Since no false positives

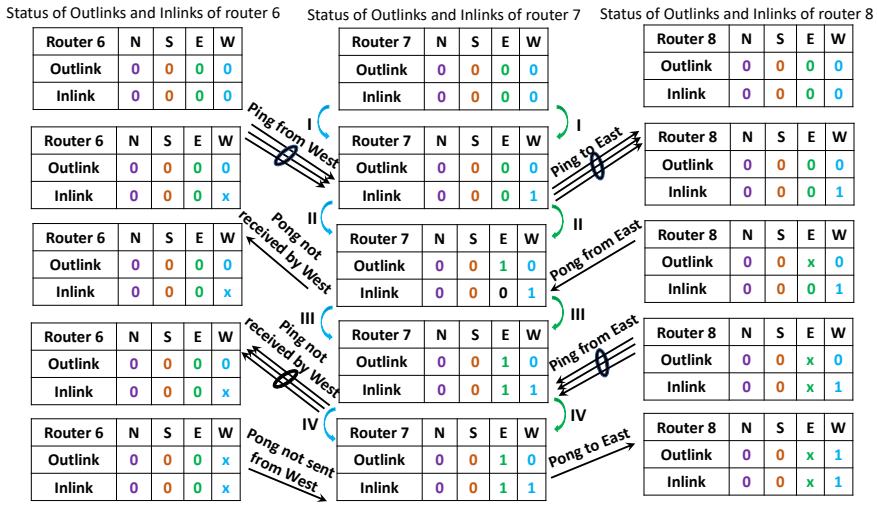

Fig. 4: Assessment of Health of Links
Updates of Health of Links of Router 7 indicated by Roman Numerals

are reported, a link is declared healthy (by the control unit) even if only one router (sharing this link with another router) reports it as healthy. The information packet helps the control unit to determine the bits which are affected by stuck-at-1 faults along the path (from router to control unit). Refer Algorithm 2 for pseudo-code of this phase.

Algorithm 2 Consolidation of Health of Links by Control Unit

```
Initialize status of all link be to unhealthy
for all Routers R do
  if Updated Information received from router R then
    Update status of corresponding links
  end if
end for
```

B. Reachability Analysis

As mentioned in section I, for launching a kernel (by the control unit) and at the end of execution of a kernel, there needs to exist healthy paths from the control unit to the resources identified for the kernel, and vice-versa (indicated by continuous blue and purple lines in figure 2). In this phase, the control unit examines health of path taken to reach a router, and health of path taken by a router to reach it. Only if both paths are healthy, the control unit marks a router as reachable.

Reachability of a router from control unit: In normal operation, packets originating from the control unit to a destination router are routed using the X-Y routing algorithm. All such packets use inlinks in the horizontal direction of some routers in the row containing the control unit, and inlinks in the vertical direction of some routers (see continuous blue line in figure 2) to reach the destination router. Therefore all inlinks of routers in the row starting from the next column containing the control unit till the column containing the destination router (inlinks of routers 5, 4 and 3 in figure 2), and all inlinks in vertical direction of routers starting from the next row that contains the control unit till the row that contains the destination router (inlinks of routers 9 and 15 in figure 2) need to be healthy. From the consolidated information of inlinks obtained in the previous phase, the control unit decides whether the destination router is reachable from it or not. See Algorithm 3 for pseudo-code of this analysis.

Reachability of control unit from a router: In normal operation, packets originating from a source router to the control unit are routed using the X-Y routing algorithm. All such packets use inlinks in the horizontal direction of some routers in the row containing the source router, and inlinks in the vertical direction of some gateway routers (see continuous purple line in figure 2) to reach the control unit. Therefore all inlinks of routers in the row starting from next column containing the source router till the column containing gateway routers (inlinks of routers 16, 17 and G3 in figure 2), and all inlinks in vertical direction of gateway routers starting from the next row that contains the source router till the row that contains the control unit (inlinks of routers G2 and G1 in figure 2) need to be healthy. From the consolidated information of inlinks obtained in the previous phase, the control unit decides whether it is reachable from the source router or not. Since reachability of control unit from a router is a dual of reachability of a router from a control unit, we have not provided psuedo-code for this analysis.

C. Faulty Link Analysis

Not all links of the NoC are used for communication from/to control unit to/from routers. However, all links within a sub-region are used while a kernel is getting executed. It is therefore necessary for the control unit

Algorithm 3 Reachability of a Destination Router R_d from Control Unit

```
Reachability of R_d from Control Unit = TRUE
/* Assume R_d is reachable from Control Unit */
Inlinks-of-Row = Healthy
Row = Row of Control Unit
Column = Column of Control Unit + 1
/* Check horizontal inlinks of all routers in the row containing Control Unit from column next to Control Unit till the column containing R_d */
while (Inlinks-of-Row = Healthy) and (Column <= Column of R_d) do
  if horizontal inlink of router(Row, Column) is unhealthy then
    set Inlinks-of-Row = Unhealthy
  else
    Increment Column
  end if
end while
Inlinks-of-Column = Healthy
Column = Column of R_d
Row = Row of Control Unit + 1
/* Check vertical inlinks of all routers in the column containing R_d from the next row containing Control Unit till the row containing R_d */
while (Inlinks-of-Column = Healthy) and (Row <= Row of R_d) do
  if vertical inlink of router(Row, Column) is unhealthy then
    set Inlinks-of-Column = Unhealthy
  else
    Increment Row
  end if
end while
if (Inlinks-of-Row = Unhealthy) OR (Inlinks-of-Column = Unhealthy) then
  Reachability of R_d from Control Unit = FALSE
  /* In any of the links used in a row or used in a column for Control Unit to reach R_d is not healthy, make R_d not reachable from Control Unit */
end if
```

to determine the health of all such links. Without loss of generality, the control unit uses status of links in east direction to determine health of horizontal links, and uses status of links in south direction to determine health of vertical links.

D. Determination of Availability of Healthy sub-regions

Results of analysis of previous steps are used by the control unit to determine healthy sub-regions of the execution engine. Before launching a kernel for execution on the execution engine, the control unit identifies a potential sub-region of the engine for the kernel. The control unit examines if all routers belonging to the sub-region are reachable. From the consolidated information of inlinks and outlinks obtained in the previous phase, the control unit decides if all links within the sub-region are healthy. The control unit also checks if ingress/egress links of routers in the sub-region are healthy. Only if all of the above conditions are satisfied, the control unit declares the sub-region as healthy and initiates launching of kernel. At the end of process, a fault coverage of 100 percent is guaranteed since there are no false positives reported. There can be a scenario where a large fabric is marked as unavailable due to one critical fault. However, a faulty router will never be marked as available.

V. Implementation and Results

In this section we present the results of hardware enhancements, and the time taken to assess the health of MP-SoC using out methodology.

A. Microarchitectural Enhancements

We used Chisel [20] to build a NoC comprising 12 routers. We implemented this example MP-SoC on a Virtex7 2000T FPGA device. In order to check the validity of our proposal, we artificially injected faults in the inter-router links. One such example was to inject a fault into a crucial link. We simulated a stuck-at fault in the south inlink of router G1. Our methodology identified the upper half of the execution engine as unavailable. This is one of the worst-case scenerio, since the south inlink of G1 is used by all routers in the upper half to communicate with the control unit. The lower half of the execution engine was available for executing user applications. We synthesized our enhanced design using Cadence RTL Compiler (for various sizes of execution engine) on a 40 nm technology, and observed overheads due to microarchitectural enhancements to be less than 2%.

B. Price for Increasing Availability

For an execution engine of size $n \times n$, as described in section IV, all routers assess the health of 4 inter-router links and ingress/egress links in parallel, as per Algorithm 1. This procedure takes constant time, irrespective of the size of the execution engine. Following this, each router, at a pre-determined time, sends the status of links to the control unit (refer Algorithm 1), which is a sequential process.The control unit then consolidates information so received in a sequential manner.

The time taken by routers to send their information to the control unit for an execution fabric of size $(n \times n)$ is calculated as follows:

For n - Even

No. of cycles taken by a packet to travel from each router to router G1: First row - 1 + 2 + ... + n/2 + n/2 + (n-2)/2 + ... + 1 + 0 (0 is for router G1)

Second row - 2 + 3 + ... + (n+2)/2 + (n+2)/2 + ... + 2 + 1

.

.
Middle row - (n+2)/2 + (n+4)/2 + ... (depends on n) + ... + (n+4)/2 + (n+2)/2 +n/2
.
.
Last row - 2 + 3 + ... + (n+2)/2 + (n+2)/2 + ... + 2 + 1
For n - Odd
No. of cycles taken by a packet to travel from each router to router G1:
First row - 1 + 2 + ... + n/2 + (n+2)/2 + n/2 + ... + 1 + 0 (0 is for router G1)
Second row - 2 + 3 + ... + (n+2)/2 + (n+4)/2 + (n+2)/2 + ... + 2 + 1
.
.
Middle row - (n+2)/2 + (n+4)/2 + ... (depends on n) + ... + (n+4)/2 + (n+2)/2 +n/2
.
.
Last row - 2 + 3 + ... + (n+2)/2 + (n+4)/2 + (n+2)/2 + ... + 2 + 1
Apart from this, packets from all routers take 2 additional cycles(one for going into the network interface and one for flit to packet conversion) to reach the control unit. Therefore, the sum of the above partial sums along with $n \times (n+1) \times 2$ (to reach control unit) gives the total number of cycles used in the consolidation phase. The time taken for these operations is hence of order $O(n \times n)$. Figure 5 shows the number of cycles incurred for consolidating health of links for varying sizes of the MP-SoC. As can be observed from the figure, the latency is proportional to size of NoC. A near linear graph shows how latency scales with fabric size. A MP-SoC with an execution engine of size 16x16 incurs a latency of 3056 cycles to consolidate health of links. A MP-SoC clocked at 1GHz, spends only 3 microseconds for consolidation of health, which is an insignificant overhead.

VI. Conclusions

NoC is a key component of MP-SoCs since it is inherently scalable to accommodate hundreds of processors. MP-SoCs are used in a multiple domains of applications, including those that require safe and secure computing. Multitude of routers are interconnected to form the NoC. In this proposal we analyse the availability of the excution fabric even in presence of some faults. It is shown that the location of faults is a more important factor than the number of faults in deciding the percentage of available fabric for kernel execution. In this proposal we showed how distributed built-in-self-test for NoC is realized as a simple micro-architectural enhancement that adds insignificant hardware overhead of less than 2% and insignificant latency for availability analysis. MP-SoCs are inherently parallel and we used this property to assess health of links (that interconnect routers) in a parallel manner. Our proposal not only identifies faulty links, but also identifies healthy portions of a MP-SoC that can be used for execution, thus making the system available to applications that need safe and secure computing environment.

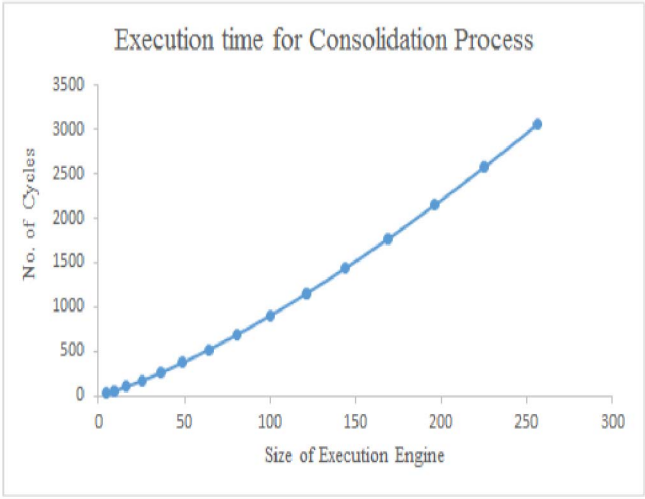

Fig. 5: Latency Incurred for Consolidation

VII. Future Work

This work can be extended to evaluate the availability of the system when there are other types of faults such as transient faults, or faults in the compute or memory resources connencted to the router. Performing this process in parallel to kernel execution can be explored and the corresponding effects on performance of kernel execution can be identified.

References

[1] C. Grecu, P. Pande, A. Ivanov, and R. Saleh, BIST for Network-on-Chip Interconnect Infrastructures, in VLSI Test Symposium, 2006. Proceedings.24th IEEE, pp. 6-pp, IEEE, 2006.

[2] C. Grecu, A. Ivanov, R. Saleh, and P. P. Pande, Testing Network-on-Chip Communication Fabrics, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 26, no. 12, p. 2201, 2007.

[3] J. Raik, R. Ubar, and V. Govind, Test Configurations for Diagnosing Faulty Links in NoC Switches, in Test Symposium, 2007. ETS'07. 12th IEEE European, pp. 29-34, IEEE, 2007.

[4] M. Hosseinabady, A. Banaiyan, M. N. Bojnordi, and Z. Navabi, A Concurrent Testing Method for NoC Switches, in Proceedings of the conference on Design, automation and test in Europe: Proceedings, pp. 1171-1176, European Design and Automation Association, 2006

[5] A. M. Amory, E. Briao,E. Cota, M. Lubaszewski, and F. G. Moraes, A Scalable Test Strategy for Network-on-Chip Routers, in Test Conference, 2005. Proceedings. ITC 2005. IEEE International, pp. 9-pp, IEEE, 2005.

[6] R. Nourmandi-Pour, A. Khadem-Zadeh, and A. M. Rahmani, An IEEE 1149.1-based BIST method for at-speed testing of inter-switch links in network on chip, Microelectronics Journal, vol. 41, no. 7, pp. 417-429, 2010.

[7] J. D. Lee and R. N. Mahapatra, In-field NoC-based SoC testing with Distributed Test Vector Storage, in Computer Design, 2008. ICCD 2008. IEEE International Conference on, pp. 206-211, IEEE, 2008.

[8] K. Petersen and J. Oberg, Toward a Scalable Test Methodology for 2D-mesh Network-on-Chips, in Proceedings of the conference on Design, automation and test in Europe, pp. 367-372, EDA Consortium, 2007.s

[9] Ren, Pengju and Kinsy, Michel A and Zhu, Mengjiao and Khadka, Shreeya and Isakov, Mihailo and Ramrakhyani, Aniruddh and Krishna, Tushar and Zheng, Nanning, FASHION: Fault-Aware Self-Healing Intelligent On-chip Network, arXiv:1702.02313, 2017

[10] Chen, C., Fu, Y. and Cotofana, S., 2017. Towards Maximum Utilization of Remained Bandwidth in Defected NoC Links. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36(2), pp.285-298.

[11] Iordanou, C., Soteriou, V. and Aisopos, K., 2014, October. Hermes: Architecting a top-performing fault-tolerant routing algorithm for Networks-on-Chips. In 2014 IEEE 32nd International Conference on Computer Design (ICCD) (pp. 424-431). IEEE.

[12] M.Alle, K.Varadarajan, A.Fell, N.Joseph, S.Das, P.Biswas, J.Chetia, A.Rao, S.Nandy, R.Narayan,et al., REDEFINE: Runtime reconfigurable polymorphic ASIC, ACM Transactions on Embedded Computing Systems(TECS), vol. 9, no. 2, p. 11, 2009

[13] M.Alle, K.Varadarajan, A.Fell, S.K. Nandy, and R. Narayan, Compiling Techniques for Coarse Grained Runtime Reconfigurable Architectures, proceedings of the 5th International Workshop on Applied Reconfigurable Computing, Karlsruhe, Germany, March 2009.

[14] A.Fell, P.Biswas, J.Chetia, S.K.Nandy, and R.Narayan, Generic routing rules and a scalable access enhancement for the Network-on-Chip RECONNECT, (proceedings of the SOCC 2009), SOC Conference, September 2009

[15] B.D.de Dinechin, R. Ayrignac, P.E. Beaucamps, P. Couvert, B. Ganne, P. G. de Massas, F. Jacquet, S. Jones, N. M. Chaisemartin, F. Riss, et al., A Clustered Manycore Processor Architecture for Embedded and Accelerated Applications., in HPEC, pp. 1-6, 2013.

[16] A. Olofsson,Epiphany-v: A 1024 processor 64-bit RISC System-on-Chip, arXiv preprint arXiv:1610.01832, 2016

[17] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, et al., Tile64-Processor: A 64-core SoC with Mesh Interconnect, in Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International, pp. 88-598, IEEE, 2008.

[18] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baas, A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array, in 2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits), pp. 1-2, IEEE, 2016.

[19] Junxiu Liu, Jim Harkin, Yuhua Li, Liam Maguire, Online fault detection for Networks-on-Chip interconnect, NASA/ESA Conference on Adaptive Hardware and Systems (AHS), July 2014

[20] Bachrach, Jonathan, et al. "Chisel: constructing hardware in a scala embedded language." In DAC Design Automation Conference 2012, pp. 1212-1221. IEEE, 2012.