Reutilization of Trace Buffers for Performance Enhancement of NoC based MPSoCs

Sidhartha Sankar Rout, Badri M, Sujay Deb
Indraprastha Institute of Information Technology Delhi, New Delhi, India,
e-mail: {sidharthas, badri17164, sdeb}@iiitd.ac.in

Abstract— The contemporary network-on-chips (NoCs) are so complex that capturing all network functional faults at pre-silicon verification stage is nearly impossible. So, on-chip design-for-debug (DfD) structures such as trace buffers are provided to assist capturing escaped faults during post-silicon debug. Most of the DfD modules are left idle after the debug process. Reuse of such structures can compensate for the area overhead introduced by them. In this work, the trace buffers are re-utilized as extended virtual channels for the router nodes of an NoC during in-field execution. Optimal distribution of trace buffers among the routers is performed based upon their load profiling. Experiments with several benchmarks on the proposed architecture show an average of 11.36% increase in network throughput and 13.97% decrease in average delay.

I. Introduction

Modern day system-on-chips (SoCs) integrate multiple cores in a single package. These cores need an efficient communication infrastructure to exchange information. Network-on-chip (NoC) being a shared medium, allows low latency as well as low contention parallel communication, and is considered as a suitable interconnection fabric for multi-processor system-on-chip (MPSoC) [1]. In an NoC based system, messages are transferred in the form of *packets*, and packets are further divided into *flits* as the smallest flow control units. While traversing from source to destination, packet routing decisions are taken at each intermediate router. Addition of *buffers* to the routers improves the network flow control, as they decouple the allocation of adjacent channels [2]. The network throughput can further be improved by dividing the buffer storage into multiple *virtual channels* (VCs) that can provide a deadlock free message routing on the network [3].

With the increasing complexity level of SoCs, the design of interconnection module is also becoming intricate. To achieve high speed, low power and reliable packet communication, several sophisticated NoC architectures are evolved with application specific topologies and additional features [4], [5]. Design complexity of interconnect unit prolongs the IP verification process. A time bound pre-silicon verification phase cannot ensure the complete functional correctness. Even if the processing cores behave correctly, the escaped network bugs can introduce multiple communication faults such as *dropped data fault*, *misroute*, *deadlock*, *livelock* etc. [6]. So, post-silicon debug is a necessary step to be performed on NoC IP for its validation [7], [8]. It is executed at system clock on initial prototype chips to capture the slipped functional errors [9]. Few critical signals are selected during design phase, which can be traced to increase the observability and controllability during debug phase [10]. Signal traces are stored in a design-for-debug (DfD) structure named *trace buffer* (TBUF), and are later exported to an external debugger for fault analysis.

System internal visibility and DfD overhead are competing in nature. The implementation cost of debug system largely depends upon trace buffer size [11] . As the size and complexity of NoC based MPSoCs are increasing, the trace buffer footprint is also growing to maximize the system obersvability during debug. The area overhead introduced by trace buffers is considered as a major design concern as the DfD hardware becomes non-functional once the SoC goes into production. To address this issue, reuse of architectural component as DfD component or vice versa have been proposed in several research fronts [7], [8], [12]–[16] . In this work, we have proposed to reuse the NoC trace buffer as extended VCs of network routers to improve the throughput and performance of NoC based MPSoCs. Total trace buffer can be distributed among all the routers to store corresponding packet traces during debug phase as shown in Fig. 1 (a). Whereas the same trace buffers can be reused as extended VCs of router input channels to improve network throughput during in-field execution as shown in Fig. 1 (b).

II. Background and Related Work

A. Virtual Channels for Network Performance

Addition of buffers to network router can improve network throughput and division of buffers to multiple VCs can further improve it [2], [3] . Research has shown that better outcome can be achieved in terms of area, power and performance by allocating the VC space specific to application [17] . Dynamic allocation of VC is explored in [18] to further enhance the efficiency of the system.

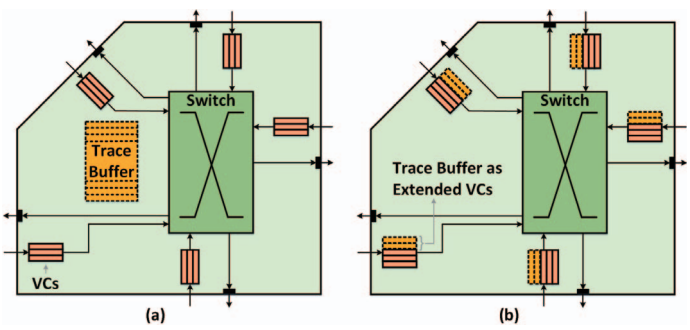

Fig. 1. (a) Trace buffer used for storing traces during debug, (b) Trace buffer used as extended VCs during in-field execution.

978-1-7281-4123-7/20/$31.00 © 2020 IEEE

B. Reuse of Architectural and Debug Components

Majority of debug hardware become unused and are treated vestigial once the chip is sent for production. So, many efforts have been made to reuse the architectural components for debug purpose or vice versa to overcome the area overhead introduced by DfD structure. The work in [12], instead of using a dedicated trace buffer, uses data cache to store both traces and data during debug. Authors in [13] propose to use L1 and L2 caches to store memory operation activity logs required for memory consistency and coherence validation. A dedicated portion of L2 cache of each node is used as trace buffer for NoC validation in [7], [8]. While [12] uses write-back circuitry to export trace content, [8] utilizes existing wired and wireless channels of NoC for same purpose. Similarly there are few efforts, which demonstrate reuse of DfD hardware for some architectural enhancements. Authors in [14] have re-employed the debug structure for online monitoring during runtime verification. Embedded Trace buffers are re-purposed for malware prevention in [15], and reused as victim cache to enhance cache performance in [16].

C. NoC Post-silicon Debug

There are several research efforts, which demonstrate multi-core debug platform using NoC transaction monitoring [19], [20]. But [7], [8] have established structured post-silicon debug frameworks for NoC itself. A generalized NoC debug platform inherited from [8] is shown in Fig. 2. The framework can monitor all the router nodes and record the state of packets passing through each of them as traces and stores them in the trace buffer. A global view of complete packet traces provides a picture of all communication activities on the NoC and is capable of indicating the presence of any network functional fault. The trigger unit, which can be configured from the host system, can invoke the periodic or event based trace collection. On-chip transfer of packet traces are performed though trace bus. The stored traces can be transported through a JTAG port to an external debug analyzer for bug detection and localization. Both [7] and [8] have used a dedicated portion of local L2 cache as trace buffer (use of architectural component as debug module), but they have not discussed about the consequences of fault in cache or in the cache controller. Performance degradation is also expected while reusing L2 cache as trace buffer. While, [8] uses both wired and wireless channels of the network for exporting trace to external debugger, [7] talks about on-chip debugger. Both wireless interface and on-chip debugger are costly solutions. In this paper we adopt the generalized debug platform shown in Fig. 2, and propose to reuse the trace buffer as extended VCs of NoC routers (use of debug module as architectural components), that can considerably improve the network performance and throughput. Our method also reuses the wired channels of NoC for trace transfer.

III. Trace Buffers Distribution

In this work, the trace buffers are distributed among all the nodes present in the network and are utilized as extended VCs during in-field normal execution. Distribution of trace buffers are performed using a *Fair Division* (FD) algorithm. A proportional fair division method says every agent receives at least its due share of a set of resources according to its own value function [21]. In this context, routing nodes are agents, trace buffers are resources and profile index (PI) of each node based on traffic condition are considered to be respective value function $f(v)$. Following two sub-sections present the details about the calculation of PI of each node, and accordingly fair distribution of available trace buffer resources among all the nodes.

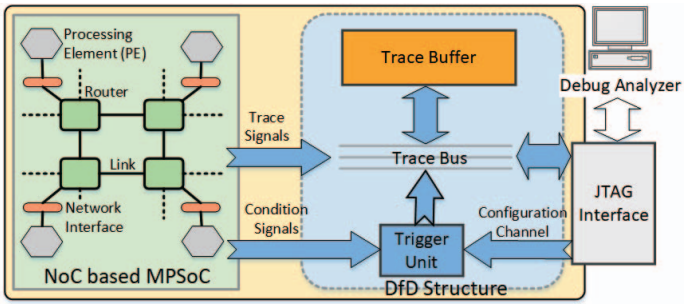

Fig. 2. Generalized NoC debug platform.

A. Profiling the Router Nodes

Each of the network routing nodes are profiled based on the amount of traffic passing through it for an application. Separate profiling of individual node is done for all the applications meant for the corresponding MPSoC. Finally, each node is assigned with a PI, which is the arithmetic mean of all of its profiling values. For a network with n nodes $\{N_1, N_2, ... N_n\}$ and m applications $\{A_1, A_2, ... A_m\}$, PI of each node can be realized as below in Eqn. 1:

$$PI_{N_i}|_{i\in\{1\ to\ N\}} = \left\{\sum_{j=1}^{m} p_{i,j}\right\}/j \tag{1}$$

Here $p_{i,j}$ represents profiling of i_{th} node for j_{th} application, and PI_{N_i} represents the final profile index value of i_{th} node. Value of $p_{i,j}$ becomes high if large number of packets traverse through the i_{th} node while j_{th} application is executed. PI value indicates an average occupancy level of the respective node by the payload packets. It can be concluded that the trace buffer requirement of each node is proportional to its PI value. So, value function $f(v)$ of each routing node is calculated based upon the corresponding normalized PI as shown in Eqn. 2.

$$f_i(v) = PI_{N_i}/\left\{\sum_{i=1}^{n} PI_{N_i}\right\} \tag{2}$$

These value functions are used in FD method for optimal distribution of the available trace buffers.

B. Fair Division of Trace Buffers

The objective of fair division is to divide and allocate resource share to each candidate, so that everyone would feel that it has got its due share according to its value function. Several FD mechanisms are proposed in literature, from which we have adopted *Dubins-Spanier Moving-Knife Procedure* [22] for the fair distribution of trace buffers. The *Moving-Knife Procedure* is originally proposed to solve the fair division of cake problem among n people, where $n > 2$. A knife is

slowly moved across a cake from its extreme left position, and whenever a person feels that the knife has moved $1/n^{th}$ of the total cake according to his measure, he calls "cut". Then he takes that piece of cake and exits. If two persons call at the same time, then the piece is given randomly to any one of them. The process is repeated for $n-1$ participants.

Algorithm 1 Modified Fair Division Algorithm for Trace Buffer Distribution

assumption: $S_{tb} = p * tb_{N(min)}$; ▷ where S_{tb} is the trace buffer size, $tb_{N(min)}$ is minimum trace buffer share per router, and p is a whole number

initial: $m = n; j = 1;$ ▷ total n number of nodes in the network $N = \{N_1, N_2 \ldots N_n\}$, j represents node number

while $m > 0$ **do** ▷ trace buffer distribution
  call function ***tb_fair_div***
  i = *Node number that asked for the trace buffer partition*
  if $tb_{N_i} \leqslant tb_{N(min)}$ **then**
    $tb_{N_i} = tb_{N(min)}$
  else
    if $(tb_{N_i} \,\%\, tb_{N(min)}) \geqslant tb_{N(min)}/2$ **then**
      $tb_{N_i} = \lceil tb_{N_i} \rceil$
    else
      $tb_{N_i} = \lfloor tb_{N_i} \rfloor$
    ▷ $\lceil tb_{N_i} \rceil$ and $\lfloor tb_{N_i} \rfloor$ returns the next and previous whole number divisible by $tb_{N(min)}$ respectively
  $S_{tb} = S_{tb} - tb_{N_i}$
  make $i \notin N$
  $m--$

if $(q = \sum_{j=1}^{n} tb_{N_i} \,/\, tb_{N(min)}) > p$ **then**
  For top q-p nodes assigned with maximum trace buffer share are reduced by $tb_{N(min)}$ *each*
else if $q < p$ **then**
  For top q-p nodes assigned with maximum trace buffer share are increased by $tb_{N(min)}$ *each*

function ***tb_fair_div*** definition
  Start the S_{tb} *division*
  Wait till any node asks for the trace buffer partition according to its value function f(v)
  If more than one node ask for the same partition then assign randomly to any one of them

In the context of trace buffer distribution, a modified FD algorithm is illustrated in Algorithm 1. This is shown for a NoC comprised of n nodes with individual node value function $f_i(v)$ and total trace buffer size of S_{tb}. The cake problem deals with the division of continuous resource, whereas trace buffer distribution is a discrete resource division problem. The smallest unit of trace buffer that can be assigned to a particular port of a router node should have the size equal to one VC. In case of a 2D mesh topology (considered for experiments in this work), each router having 5 ports results into minimum trace buffer share per router node ($tb_{N(min)}$) as 5 times a VC. Each time the trace buffer share of a node according to its value function is found not to be a whole number multiple of $tb_{N(min)}$, the final share is rounded up to either the next or previous whole number multiple of $tb_{N(min)}$. Final adjustment of trace buffer share to meet the size limit of available trace buffer is done on the top ranked nodes claiming the largest portions of it, as shown in Algorithm 1.

IV. Network Operation

This section discusses the network operation for our proposed solution in both debug as well as in-field execution modes. Fig. 3 (a) shows a 64 node network and (b) illustrates the supporting router structure. Each router node gets a portion of total trace buffer based on its value function as discussed in Section III. Following sub-sections highlight the use of trace buffer in different purposes during different operation modes. A mode signal coming from debug support unit (DSU) decides the mode of operation of the network.

A. During Debug Mode

During post-silicon debug the trace buffer is used for packet trace storage. Trace buffers corresponding to a particular router can equally be distributed among all the input ports during physical implementation (Fig. 3 (b)). Though distributed among ports, the trace buffer controller (TBUF Ctrl) considers all the trace buffers inside the router as a lumped trace storage space as can be seen in Fig. 1 (a). Whenever a packet flit reaches an input, the header decoder (HD) decodes the destination, routing computation (RC) finds the output port, virtual channel allocator (VC) assigns a particular VC, and the switch allocator (SA) grants the switch time slots to the input flits. The snapshot unit collects the packet state information (packet id, current node, input port, output port, and VC number) from HD, RC and VA and builds the respective packet trace by adding the corresponding timestamp value. The TBUF Ctrl generates the trace buffer address (*tbaddr*) and forwards the generated packet trace (*tdata_in*) to the corresponding *tbaddr* location. Generated *tbaddr* is usually a trace buffer location associated with the input port receiving the packet flit. If no trace buffer space is available at a particular input port, the TBUF Ctrl looks for empty trace buffer at other input ports of the same router and generates the corresponding *tbaddr* of a free trace buffer space. TBUF Ctrl asserts a *tb_full* signal, when it finds no free trace buffer space inside a router. This necessitates to export the trace buffer content (*tdata_out*) of the filled router to the external debug analyzer.

Trace transfer from the network routers is performed in two stages, namely local trace transfer (LTT), and global trace transfer (GTT). Whenever a particular router node runs out of trace buffer space, LTT phase starts for the corresponding router. For an example, node A in Fig. 3 (a) is a busy node, and its trace buffers fill up quickly. During the LTT phase, traffic switching in router A is paused and packet traces stored in its trace buffers are transferred to the trace bus through the existing network trace port (NTP). The network can be divided into multiple sub networks and each of them can have a NTP. While transferring the packet traces till the corresponding NTP, the intermediate routers (routers on the path from A to B) give higher priority to the traces over the normal payload while switching, and thus provide a faster trace transfer. Remaining

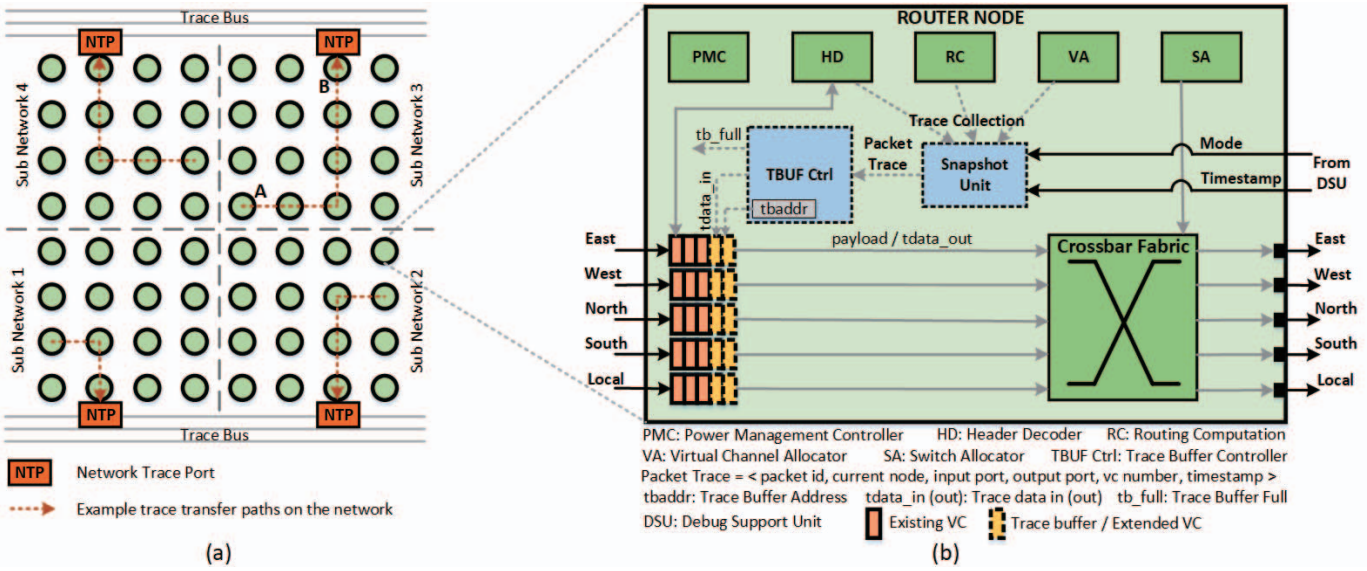

Fig. 3. (a) A 64 node network, (b) Router node architecture.

network exhibits the normal operation during LTT. After the completion of LTT phase, node A resumes with its normal operation. A GTT phase starts periodically after a pre-decided (during design phase) large time period. During this phase, the normal operation of the whole network is paused, and traces from all router nodes are collected trough the NTP ports. This is performed to collect complete network traces, so that trace analysis can be started in the debug analyzer. The network gets back to its normal operation after the GTT phase.

B. During In-field Execution Mode

During in-field execution mode, the trace buffers are no more used for debug purpose. In our proposed method, the router trace buffers are re-utilized as extended VCs (Fig. 3 (b)) to enhance the network performance. The incoming payload flits now find additional buffer space at each router input ports, leading to reduced chance of packet drop and deadlock conditions. During this mode, the TBUF Ctrl and snapshot units are power gated by the power management controller (PMC). The VA unit is designed according to the total VC (existing + extended), so that it can perform the desired arbitration during in-field execution mode. VA module keeps on snooping the available VC credit information of the down stream routers to assign the output channel to a particular incoming flit. The extended VC scenario in our method increases the possibility of VC credit having a positive value in most of the time, and thus increases throughput of the network.

V. Experimental Results

This section describes the simulation set up and discusses the results generated. Three VC distribution scenarios are evaluated such as: (i) baseline architecture (only existing VC and no extended VC), (ii) existing VC + equally distributed extended VC (whole trace buffer being equally distributed among the existing nodes), and (iii) existing VC + fair distribution based extended VC (whole trace buffer being distributed among the existing nodes according to their value functions). Network topology and simulation setup is listed in Table I. For simulation, a 8x8 2D mesh NoC is modeled on Noxim [23]. The router module is modified to incorporate the debug structures and VC extension. The proposed scheme is evaluated using 3 synthetic traffic patterns and 3 applications from SPLASH-2 benchmark suit [24]. Network level traces for Noxim is generated using Graphite simulator [25].

TABLE I
Network Topology and Simulation Setup

Component	Configuration
Topology	8x8 baseline 2D Mesh NoC, XY routing
Router	5 I/O ports, 4 existing VCs per port, extended VCs based upon value function, 2 flit buffers, 8 flit packets, 32 bit flits, 1 GHz operating clock
Debug Setup	8KB trace buffer size, single cycle snapshot, 32 bits packet trace size
Workload	Synthetic - *Random*, *Transpose*, *Butterfly*; SPLASH-2 - *Barnes*, *FFT*, *Radix*

A. Value Function Calculation and TBUF Distribution

Profiling of each NoC node is performed based on traffic condition for the calculation of value function $f(v)$. All the 6 workload patterns are executed on the 64 node NoC and profile index $p_{i,j}$ of each node for individual workload is calculated. Fig. 4 (a) shows the graphs of normalized $p_{i,j}$ for all the workload patterns. The $f(v)$ of each node is calculated from Eqn. 2 mentioned in Section III and is plotted in the same figure. Based on these $f(v)$ values, each node gets its due share of trace buffer. Fig. 4 (b) shows the distribution of trace buffer as a measure of extended VCs at each node of the NoC for both equal distribution and FD based distribution. The figure shows that for a 8KB of total trace buffer size, an equal distribution provides 15 additional VCs per router node while FD allocates as high as 30 additional VCs and as low as 10 additional VCs, which is proportional to the corresponding value function.

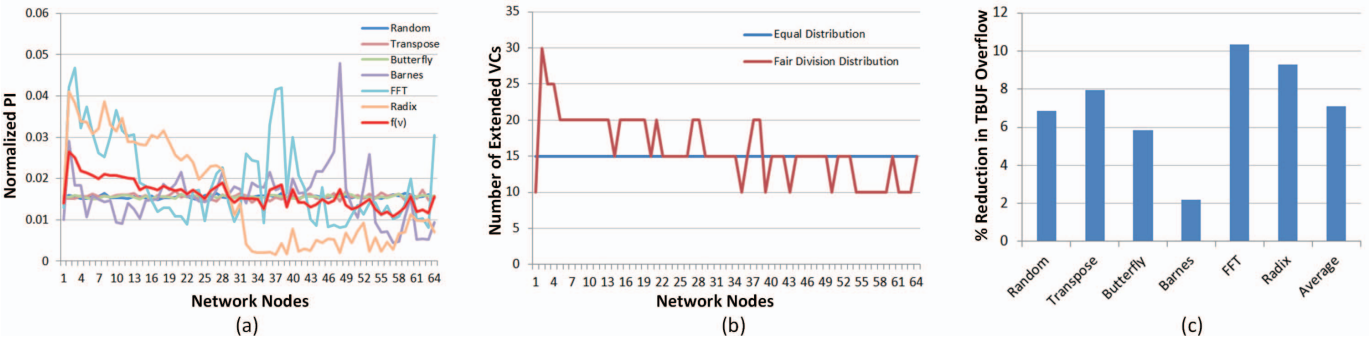

Fig. 4. (a) Normalized PI and value function of each node, (b) Trace buffer distribution in terms of extended VCs for both equal and Fair Division distribution, (c) Percentage reduction in local trace buffer overflow in case of Fair Division over Equal distribution.

Fig. 5. (a) % increase in network throughput (b) % decrease in network average delay for equal and Fair Division trace buffer distribution in comparison to the baseline architecture.

B. Trace Buffer Overflow

Whenever local trace buffers correspond to a router node fill up, the local trace transfer (LTT) phase starts as discussed in Section IV. This pauses the concerned router operation and degrades the intermediate routers operation while transferring the traces till the network trace port (NTP). Therefore, it can be concluded that more the number of local trace buffer overflow, longer the time period of debug process. Fig. 4 (c) shows an average 7% reduction in local trace buffer overflow in case of FD distribution over equal trace buffer distribution. This is because, fair distribution of trace buffers encourages parallel filling of buffer space in all routers, while in case of equal distribution, trace buffers in few routers fill up quickly, and in few others they are mostly unused. This shows that FD based trace buffer distribution would considerably speed up the debug process than the equal distribution case.

C. Network Performance

Network performance has been evaluated in terms of throughput and latency. The proposed scheme provides few additional VCs to each port of every router. As a result, network congestion reduces and thereby network throughput increases and global average packet delay decreases. Results shown in Fig. 5 (a) and (b) demonstrates that equal trace buffer distribution provides an average of 8.36% throughput improvement and 9.25% average delay reduction over baseline architecture. Whereas our proposed FD scheme provides an average of 11.36% increase in throughput and 13.97% decrease in average packet delay. The enhancement in performance parameters in case of FD scheme over equal distribution is achieved as the router nodes are allocated with additional VCs according to their requirement.

D. Fault Detection and Overhead

Re-utilization of trace buffers as extended VCs of router nodes introduces little degradation in the fault detection capability of the scheme. This is because the traces are stored in the distributed trace buffers inside the network routers and use the same network links for trace communication. During trace transfer till NTP, there is a chance that any of the intermediate router is faulty and the trace might never reach the NTP. In such scenarios, chances are there that the debug analyzer would never get a single trace correspond to a particular packet and that may lead to miss out a network fault. In our simulation we have modified the router code to introduce packet data drop and packet misroute faults to the network and executed the random traffic pattern on it. The experiments show our scheme is capable of detecting 94.53% of packet drop and 96.55% of packet misroute fault. Our proposed scheme introduces negligible area complexity to VC allocator as the number of VC increases in each router port. A baseline VC allocator synthesized in Synopsys DC compiler for 32nm technology is observed to occupy 15.3 um^2, whereas the VC allocator used in the proposed architecture occupies 32.7 um^2. VC allocator area overhead is very small in comparison to router area, which is observed to be 0.054 mm^2. At the same time the proposed method provides a huge area benefit in terms of trace buffer re-utilization as router VCs.

VI. Conclusion

This work proposes to reuse the debug trace buffer as extended VCs for the router ports to enhance the network performance. A modified *Fair Division* method is used to optimally distribute the total trace buffers among the routers according to their load condition. Simulation results show that the scheme increases an average of 11.36% network throughput and reduces 13.97% average packet delay. The area overhead introduced by modified VC allocator is observed to be negligible. The scheme is also found to be robust in terms of fault detection as 94.53% of packet drop fault and 96.55% of mistoure fault could be detected.

References

[1] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *computer*, vol. 35, no. 1, pp. 70–78, 2002.

[2] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[3] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, 1987.

[4] S. H. Gade and S. Deb, "Hywin: Hybrid wireless noc with sandboxed sub-networks for cpu/gpu architectures," *IEEE Transactions on computers*, vol. 66, no. 7, pp. 1145–1158, 2016.

[5] S. H. Gade, S. S. Rout, M. Sinha, H. K. Mondal, W. Singh, and S. Deb, "A utilization aware robust channel access mechanism for wireless nocs," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[6] A. Alaghi, N. Karimi, M. Sedghi, and Z. Navabi, "Online noc switch fault detection and diagnosis using a high level fault model," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*. IEEE, 2007, pp. 21–29.

[7] R. Abdel-Khalek and V. Bertacco, "Post-silicon platform for the functional diagnosis and debug of networks-on-chip," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, p. 112, 2014.

[8] S. S. Rout, K. Basu, and S. Deb, "Efficient post-silicon validation of network-on-chip using wireless links," in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. IEEE, 2019, pp. 371–376.

[9] P. Mishra, R. Morad, A. Ziv, and S. Ray, "Post-silicon validation in the soc era: A tutorial introduction," *IEEE Design & Test*, vol. 34, no. 3, pp. 68–92, 2017.

[10] Q. Xu and X. Liu, "On signal tracing in post-silicon validation," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*. IEEE Press, 2010, pp. 262–267.

[11] S. Mitra, S. A. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," in *Design Automation Conference*. IEEE, 2010, pp. 12–17.

[12] C.-H. Lai, Y.-C. Yang, and J. Huang, "A versatile data cache for trace buffer support," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 11, pp. 3145–3154, 2014.

[13] A. DeOrio, I. Wagner, and V. Bertacco, "Dacota: Post-silicon validation of the memory subsystem in multi-core designs," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. IEEE, 2009, pp. 405–416.

[14] N. Jindal, S. Chandran, P. R. Panda, S. Prasad, A. Mitra, K. Singhal, S. Gupta, and S. Tuli, "Dhoom: Reusing design-for-debug hardware for online monitoring," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, pp. 99:1–99:6.

[15] K. Basu, R. Elnaggar, K. Chakrabarty, and R. Karri, "Preempt: Preempting malware by examining embedded processor traces," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, pp. 166:1–166:6.

[16] N. Jindal, P. R. Panda, and S. R. Sarangi, "Reusing trace buffers as victim caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1699–1712, 2018.

[17] J. Hu and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*. IEEE Computer Society, 2004, pp. 354–361.

[18] S. S. Rout, H. K. Mondai, R. Juneja, S. H. Gade, and S. Deb, "Dynamic noc platform for varied application needs," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2018, pp. 232–237.

[19] S. Tang and Q. Xu, "A multi-core debug platform for noc-based systems," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[20] B. Vermeulen and K. Goossens, "A network-on-chip monitoring infrastructure for communication-centric debug of embedded multi-processor socs," in *2009 International Symposium on VLSI Design, Automation and Test*. IEEE, 2009, pp. 183–186.

[21] S. J. Brams and A. D. Taylor, *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press, 1996.

[22] L. E. Dubins and E. H. Spanier, "How to cut a cake fairly," *The American Mathematical Monthly*, vol. 68, no. 1P1, pp. 1–17, 1961.

[23] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Cycle-accurate network on chip simulation with noxim," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 27, no. 1, p. 4, 2016.

[24] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," *ACM SIGARCH computer architecture news*, vol. 23, no. 2, pp. 24–36, 1995.

[25] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, pp. 1–12.