Debug Data Collection for Functional Validation of Control-Flow in NoCs

Rawan Abdel-Khalek and Valeria Bertacco
Computer Science and Engineering Department, University of Michigan
(rawanak, valeria)@umich.edu

ABSTRACT

During emulation and post-silicon validation of NoC-based systems, lack of observability of the NoC's internal operations makes detection and debugging of functional bugs in the interconnect a difficult task. To verify the correctness of the control-flow portion of the NoC design, it's common to run tests that exercise the network functionality, while abstracting away the data content of traffic. In this context, we propose a methodology where network data packets are repurposed for the storage of debug information collected during execution. Upon error detection, the collected data can provide the visibility needed to help debug the error. In particular, the debug data analysis provides a reconstruction of network traffic, as well as several statistics, such as packet interactions, sequence of events within router, and packet latencies per router. In our experiments, we found that this approach allows us to reconstruct over 80% of the routes of observed packets.

1. INTRODUCTION

Networks-on-chip (NoCs) have become the prevalent communication paradigm for current and future chip-multiprocessor (CMP) and system-on-chip (SoC) architectures. To meet the continously increasing communication demand, NoC designs are growing in complexity. Router architectures often include a number of advanced features, such as virtual channels and intricate arbitration units. Moreover, a number of regular and irregular topologies can render the packet flow and the overall network subsystem extremely complex. In addition, network complexity is further increased when deploying elaborate routing protocols that utilize network state to guide routing decisions.

For these large CMP and SoC architectures, a great deal of effort is spent in the functional verification process. Moreover, with the increase in size and complexity of these systems, along with shrinking time-to-market windows, a lot of this effort is shifting towards the heavy use of emulation and post-silicon validation to ensure functional correctness. Both emulation and post-silicon validation provide orders of magnitude speedups, relative to software-based simulations of the design's RTL description. However, they suffer from limited observability of the design under test. Lack of visibility of internal operations makes the detection, diagnosis and debug of errors an extremely challenging process.

In our work, we aim to address the challenge of validating the control-flow in the complete NoC subsystem on these fast platforms. Verifying the functional correctness of the control flow portion of a NoC essentially means validating all functionality, and hence control decisions made in the network. The NoC functionality is entirely dependent on the traffic patterns observed and it is agnostic to the data content of the messages. Therefore, specialized test cases can be run with the goal of exercising as much of the network's functionality as possible. When running such tests, the data contents of packets are effectively irrelevant. In this context, we propose a methodology to greatly enhance the observability of the network traffic and its internal state to facilitate the detection and debug of control-flow functional errors. Our solution, called DiAMOND, proposes to replace packets' data contents with debug information collected during the network's execution. Along with this data collection mechanism, we instrument routers with small checkers that can detect various functional errors. Upon error detection, the collected debug data is analyzed by software-based algorithms running on the CMP/SoC cores or off-chip. The information that can be extracted from this data includes a detailed overview of the paths of packets, analysis of performance metrics at internal routers, as well as the sequence of events observed at a given router during a given interval. Armed with this enhanced visibility of network behavior, verification engineers can localize and debug functional (and in some cases performance) bugs.

2. RELATED WORK

Previous work on post-silicon validation of NoCs have proposed various approaches to increase NoC observability. In [1], authors instrument routers and network interfaces with monitors. These monitors filter network traffic to identify transactions of interest, as well as analyze performance and validate data flow errors. Similarly, approaches proposed by [2,3] add monitors to network routers that observe traffic and abstract it into events or transactions. Our solution differs by focusing on the validation of functional bugs in the control-flow portion of NoC designs. It can also detect and debug some types of performance bugs.

Other recent work proposes a post-silicon solution that relies on taking periodic snapshots of traffic to reconstruct packet paths and identify functional errors related to forward progress [4]. By periodically sampling traffic, this solution has low error detection probability for bugs that are transient in nature, such as misroutes or starvations. It also fails to detect other types of bugs, such as dropped packets. In contrast, our approach achieves better error detection capabilities, as well as provides greater observability of network traffic, as we can reconstruct longer and more uniform routes for each packets.

A number of works have targeted emulation of networks-on-chip [5–9], where authors proposed different ways to implement an emulation platform that allows the modeling and exploration of various NoC designs. Our work is complementary to these approaches. Independently of how the NoC is emulated, we provide a methodology to collect debug data from network routers to facilitate the debugging of functional errors in the control flow portion of the design.

3. THE DIAMOND SOLUTION

3.1 Overview

The aim of DiAMOND is to provide observability of the network's operation to facilitate the diagnosis and debug of functional errors in the NoC's control flow. Tests used to validate the NoC functionality aim at creating various network

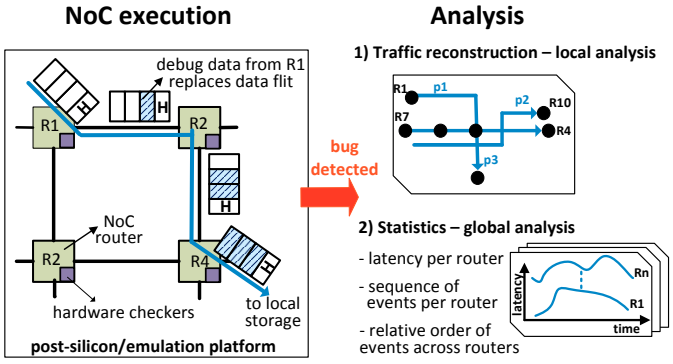

Figure 1: Overview of our solution. During NoC execution, debug data is collected at every hop and stored in the packet, overwriting data flits. Routers are instrumented with hardware checkers that monitor execution and flag functional errors. Upon error detection, the debug data is analyzed to reconstruct traffic and provide a number of relevant statistics.

traffic scenarios, while abstracting away the data content of messages. When running such tests, our solution relies on using the contents of packets to store debug data collected during execution. As packets traverse the network, their data content is substituted with debug information collected at every hop, as illustrated on the left side of Figure 1. Once a packet is delivered to its destination node, it is stored in the local cache or memory associated with that node. In parallel with debug data collection, small hardware checkers monitor the network's execution and detect functional bugs. Upon flagging an error, execution is halted and the debug data that has been collected in the caches is analyzed. This analysis process is first carried out locally, where each processor core examines the debug data of packets that were destined to it, and then globally, where debug data from all nodes are aggregated at a central location for a global overview of the network's behavior.

3.2 Debug Data Collection

Debug data is collected for every packet injected into the network and at every hop during its flight from source to destination. For every input buffer within the router, we add a register, called *log_buffer*, to store the collected debug information. In addition, we require each router to include a packet counter (*pckt_cntr*) that is incremented upon receiving a packet. The *log_buffer* is updated everytime a new packet is at the head of its corresponding input buffer. The information collected and stored in the *log_buffer* consists of:

1. The router ID
2. Arrival timestamp (*timestampA*) that indicates the value of the *pckt_cntr* when the header flit of the packet was received by the router.
3. Departure timestamp (*timestampD*) that indicates the value of *pckt_cntr* when the header flit was sent from the router. Logging timestampA and timestampD allows us to order packets passing through each router, as well as reason about packet interactions within a router.
4. A third timestamp (*pckt_latency*) that indicates the amount of time (in cycles) the packet's header flit remained in the router. This timestamp allow us to analyze packet latencies observed at interval routers.
5. The packet's input port and input virtual channel.
6. The output port and virtual channel the packet requests.

Once the *log_buffer* is complete, the debug data is written to one of the packet's body flits. The index of the flit to be written is maintained by a counter (*flit_write_index*) that is added to the packet's header flit. A typical flit width in NoCs is between 128 and 256 bits [10]. In our evaluation, we assume a flit width of 128 bits and a *log_buffer* size of 64 bits. Therefore, the debug data collected at every hop occupies only half a flit, with the remaining half written at the next hop.

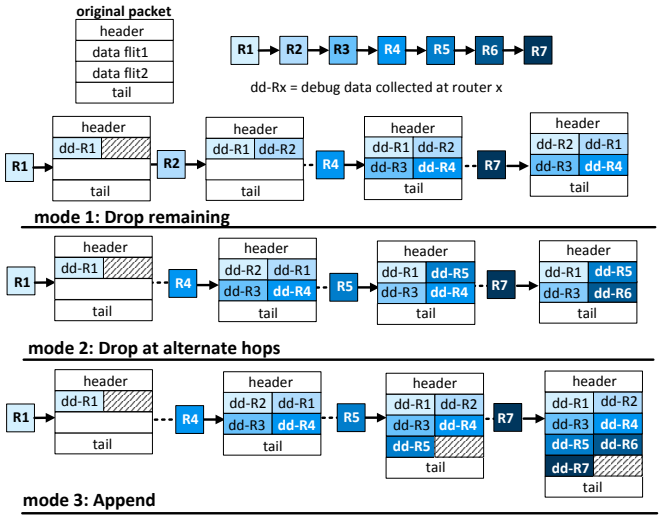

Figure 2: The three modes of operation of debug data collection.

In the case of packets that do not have enough flits to store the collected debug information, we provide three solutions for our approach, depending on the needs of the verification methodology in use: *drop remaining*, *drop at alternate hops*, and *append*. Figure 2 illustrates the behavior of each mode.

3.2.1 Drop Remaining

During drop remaining, when the number of hops in a packet's path exceeds the available flits in the packet, additional debug data is dropped. This mode of operation is simple to implement at the expense of low observability for packets with long routing paths.

3.2.2 Drop at Alternate Hops

When the space in the packet is exhausted, new debug data overwrites older debug data, creating an every-other-hop scheme. As opposed to drop remaining, this mode provides a longer and more uniform overview, even of long routing paths. Moreover, data belonging to the alternating missing hops can be partially reconstructed or extrapolated from the debug data that remains. For example, the output port requested along with the router ID can be used to determine the downstream router, for which we have no log.

3.2.3 Append

We also provide a mode of operation that allows routers to append new flits to the packet. While this mode provides

the complete path of a packet, it requires additional hardware to add the new flits. It also alters the network's original execution by creating longer packets.

3.3 Error Detection

We propose the use of a fine-grain detection approach that relies on adding small checkers to the NoC routers. These checkers monitor the network's runtime execution for signs of erroneous behavior, while targeting a wide range of functional bug manifestations detailed below.

Deadlock and starvation: A common technique to detect blocked packets uses counters, one for each input buffer. While a header flit is at the head of an input buffer, the corresponding counter is incremented in every cycle. If the counter exceeds a user-defined threshold, it flags an error [11].

Livelock: A common approach to detecting a livelock adds a hop counter to the header flit of every packet. The counter is incremented at every hop and a livelock is flagged if the counter exceeds a pre-defined threshold [12].

Dropped and duplicated packets: To detect dropped packets, we utilize the approach proposed by [13], where a packet counter is maintained per router. The counter is incremented upon receiving a tail flit and decremented upon sending one. If packets are not dropped, then the counter should reach a value of zero at some point within a checking window. Similarly, a packet counter reaching a negative value is used to identify packet duplication or creation.

Dropped and duplicated flits: In order to identify dropped and duplicated flits within packets, we require the addition of a *size* field to the header flit. A simple counter and comparator added to every input buffer checks whether the number of flits observed matches the *size* field.

Misrouting: Misrouting errors can be detected by simple checkers either at destination nodes or internal nodes. The exact checker implementation at internal routers is dependent on the routing protocol. For example, for deterministic routing or for minimal routing algorithms, a simple lookup table or assertion can detect such errors [13].

3.4 Debug Data Analysis

Once an error has been flagged, execution is halted and the collected debug data is processed in two steps:

3.4.1 Local Processing

During this phase, the contents of each local cache are individually analyzed. By examining the contents of every packet, its path through the network can be reconstructed. In addition, by examining the *pckt_latency* timestamps recorded, network performance can be analyzed. Periods that exhibited high packet latency can be identified along with the routers where this high latency was recorded. Finally, by comparing a packet's requested output port and output virtual channel within a router relative to the input port and input virtual channel of the downstream router, functional bugs in switch arbitration logic can be flagged.

3.4.2 Global Processing

Data from all local caches are aggregated at a central location and grouped per router. Using the *timestampA* and *timestampD* counters, each router's data is sorted by increasing time. The sorted information basically encapsulates the series of packets and events witnessed by each router during execution. This, in turn, gives insights regarding packet interactions within routers, allowing us to reason about the source of the error observed. Since each router's *timestampA* and *timestampD* represent the value of the router's packet counter, these timestamps do not have a notion of physical time. Therefore, the arrival and departure of packets from different routers can not be correlated. However, by leveraging techniques similar to those used in ordering events for distributed systems [14], we can still construct a partial order of events by using packets as points of reference.

4. EXPERIMENTAL EVALUATION

We modeled an 8x8 mesh network using the cycle-accurate Booksim simulator [12]. Our baseline router architecture consisted of an input-queued virtual channel router, with 5 input ports and 2 virtual channels per port. We ran both random directed traffic, as well as network flow traces from the PARSEC benchmark suite [15]. For the PARSEC network flow, traffic consisted of 1 flit control packets and 5 flit data packets.

PARSEC network flow	drop remaining	drop at alternate hops	append
blackscholes	83.2%	96.3%	100%
bodytrack	85.0%	97.1%	100%
dedup	84.4%	96.8%	100%
ferret	84.5%	96.9%	100%
freqmine	83.8%	96.6%	100%
streamcluster	84.3%	96.8%	100%
swaptions	84.2%	96.8%	100%
vips	81.4%	95.4%	100%
x264	83.0%	96.2%	100%
average	**83.76%**	**96.54%**	**100%**
uniform traffic packet size = 5 flits	87.1%	97.8%	100%

Table 1: Average path reconstruction

4.1 Path Reconstruction Results

We first examined the observability gained from our solution by evaluating the fraction of the path that can be observed for each packet, as shown in Table 1. For the PARSEC network flow, data packets consist of 3 body flits and can carry complete debug data from 6 routers along their path, providing a percentage of path reconstruction of 83.76% on average during the drop remaining mode. The drop at alternate hops mode provides higher path reconstruction results, as new debug data replaces older data by over-writing only the second half of each body flit. Moreover, routers pertaining to the alternate missing hops are extrapolated. Note that, for the PARSEC network flow, we are not able to collect any debug data for the 1 flit control packets during these two modes. Finally, for the append mode, we achieve 100% path reconstruction for both control and data packet, as expected. However, the append mode introduces a performance overhead, as longer packets increase network congestion and can slow down execution. Consequently, this mode may also expose different bugs than the original application. For the PARSEC network traces, the average network latency increases by 1.5-2 times relative to the baseline.

4.2 Case Study: Analysis Results

Packet interactions. Our solution allows us to examine packet interactions within routers, as well as construct a partial overview of global network behavior. Debug data from all nodes is sorted by increasing *timestampA* values, allowing us to order packet arrival and departure to and from each router. As an example, figure 3 shows the packets traversing router0 and router1 in the first 100 cycles of running uniform traffic at an injection rate of 0.19 flits/node/cycle. Since each router's *pckt_cntr* operates independently, the *timestampA* values are not synchronized across routers, preventing the establishment

of a complete global order of events across the network. However, by leveraging common packets as points of reference, we can establish a partial order. For example, in Figure 3, packet 132 is a common packet between routers 0 and 1. Based on that, events relating to packets 31, 87 and 62 in router0 (*i.e.* the events that occurred before sending packet 132) happened before events associated with packet 143 in router1.

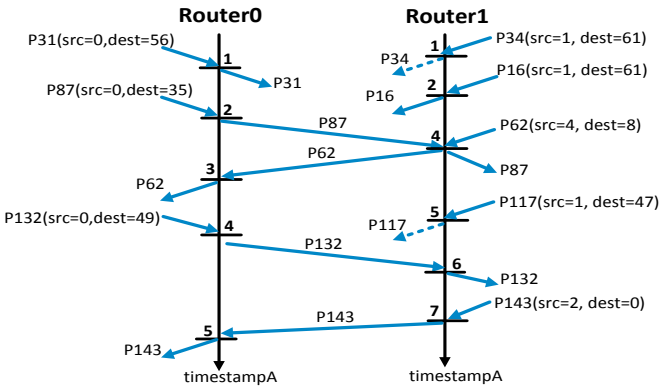

Figure 3: Example of reconstruction of packet interactions. TimestampA values are used to construct the sequence of events observed in each router.

Latency at internal routers. Typically, during the performance validation of NoCs, the debug information that can be collected relies on an end-to-end analysis of latencies and throughput. However, our solution has the benefit of providing performance statistics at internal routers. By examining the *pckt_latency* field recorded in the debug data, we can study the average and maximum packet latencies observed at every router and throughout the network's execution. We can also plot the average packet latencies observed within each router during a specific testbench execution. For example, Figure 4 shows this information for the *dedup* benchmark, where we can observe that router 49 exhibits the highest average packet latency compared to other routers. Using such results, we can identify potential performance bottlenecks in the network.

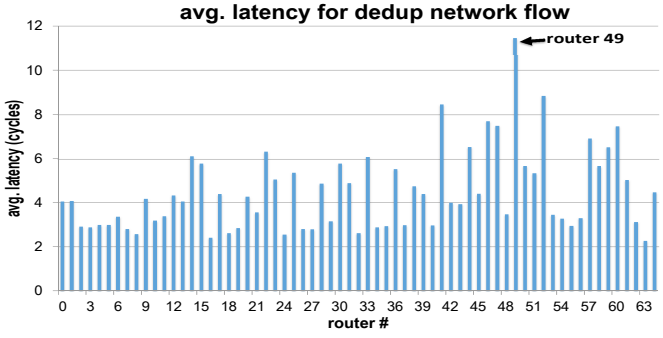

Figure 4: Average packet latency at each router for a sample benchmark.

Our scheme also allows examining packet latency values over time and per router. For example, Figure 5 shows the variation in packet latencies observed at router 49 throughout the execution of *dedup*. Based on that, execution periods of interest can be identified for further analysis, such as the period highlighted in Figure 5, where we record the first significant latency increase.

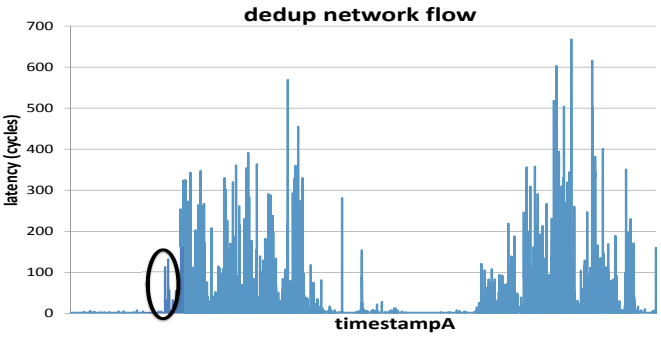

Figure 5: Packet latency at router 49 for dedup benchmark.

5. CONCLUSION

We presented DiAMOND, a debug solution for the post-silicon validation and emulation of networks-on-chip. Targeting the functional validation of the control-flow portion of NoCs, we log debug data during network execution and store them by replacing the data content of packets. Upon bug detection, the collected debug data provides increased observability of network traffic, allowing in most cases over 80% reconstruction of the paths of packets. DiAMOND also provides several functional, as well as peformance, statistics.

Acknowledgments. This work was supported in part by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, and NSF grant #1217764.

6. REFERENCES

[1] B. Vermeulen and K. Goossens, "A network-on-chip monitoring infrastructure for communication-centric debug of embedded multi-processor socs," in *VLSI-DAT*, 2009.

[2] C. Ciordas, K. Goossens, T. Basten, A. Radulescu, and A. Boon, "Transaction monitoring in networks on chip: the on-chip run-time perspective," in *IES*, 2006.

[3] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J. Meerbergen, "An event-based network-on-chip monitoring service," in *HLDVT*, 2004.

[4] R. Abdel-Khalek and V. Bertacco, "Functional post-silicon diagnosis and debug for networks-on-chip," in *ICCAD*, 2012.

[5] L. Yangfan, L. Peng, J. Yingtao, Y. Mei, W. Kejun, W. Weidong, and Y. Qingdong, "Building a multi-FPGA-based emulation framework to support networks-on-chip design and verification," *International Journal of Electronics*, vol. 97, 2010.

[6] L. Peng, X. Chunchang, W. Xiaohang, X. Binjie, L. Yangfan, W. Weidong, and Y. Qingdong, "A NoC emulation/verification framework," in *ITNG*, 2009.

[7] N. Genko, D. Atienza, G. De Micheli, J. Mendias, R. Hermida, and F. Catthoor, "A complete network-on-chip emulation framework," in *DATE*, 2005.

[8] N. Genko, D. Atienza, G. De Micheli, and L. Benini, "Feature - NoC emulation: a tool and design flow for MPSoC," *Circuits and Systems, IEEE*, vol. 7, 2007.

[9] M. Kouadri, M. Abdellah, B. Senouci, and F. Petrot, "Large scale on-chip networks:an accurate multi-FPGA emulation platform," in *DSD*, Sept 2008.

[10] S. Ma, N. Enright Jerger, and Z. Wang, "Whole packet forwarding: Efficient design of fully adaptive routing algorithms for networks-on-chip," in *HPCA*, 2012.

[11] K. Anjan and T. Pinkston, "DISHA: A deadlock recovery scheme for fully adaptive routing," in *IPPS*, 1995.

[12] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[13] A. Ghofrani, R. Parikh, S. Shamshiri, A. DeOrio, K.-T. Cheng, and V. Bertacco, "Comprehensive online defect diagnosis in on-chip networks," in *VTS*, April 2012.

[14] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, 1978.

[15] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," 2008.