A Clustering-Based Scheme for Concurrent Trace in Debugging NoC-Based Multicore Systems

Jianliang Gao*, Jianxin Wang*, Yinhe Han†, Lei Zhang† and Xiaowei Li†
* *School of Information Science and Engineering, Central South University*
† *Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences*

Abstract

Concurrent trace is an emerging challenge when debugging multicore systems. In concurrent trace, trace buffer becomes a bottleneck since all trace sources try to access it simultaneously. In addition, the on-chip interconnection fabric is extremely high hardware cost for the distributed trace signals. In this paper, we propose a clustering-based scheme which implements concurrent trace for debugging Network-on-Chip (NoC) based multicore systems. In the proposed scheme, a unified communication framework eliminates the requirement for interconnection fabric which is only used during debugging. With clustering scheme, multiple concurrent trace sources can access distributed trace buffer via NoC under bandwidth constraint. We evaluate the proposed scheme using Booksim and the results show the effectiveness of the proposed scheme.

1. Introduction

In the design flow of integrated circuits (ICs), pre-silicon verification techniques, such as simulation or formal methods, aim to eliminating bug in a circuit before it is manufactured [1, 2]. With the increasing design complexity and the inadequate accuracy in modeling ICs, pre-silicon verification is insufficient to guarantee bug-free first silicon [3]. Due to the escalating cost of volume production, it is imperative to identify the escaped bug as soon as the first silicon is available [4]. Compared with pre-silicon verification, the challenge of post-silicon debug is the observability of the internal signals [5].

Scan-based and trace-based technologies are common design-for-debug (DfD) to improve the visibility of internal signals for post-silicon debug [3]. Scan-based technology needs to stop functional execution which might cause data invalidation. Especially, it is difficult or even impossible to reproduce the bugs for the non-determination of parallel environment [6]. Trace-based technology can achieve real-time observability by transferring trace data into on-chip trace buffer and the saved data are subsequently analyzed off-line [7]. In today's post-silicon debug, trace-based technologies are widely used to identify the potential bugs [8].

For the parallelism of multicore system, trace signals from different cores need to be traced simultaneously. Trace-based technologies face the new challenge of concurrent trace when coming to multicore systems. In concurrent trace, the states of multiple trace sources which belong to different cores need to be transferred and stored simultaneously [9]. When multiple trace sources access the same trace buffer, trace data might be lost for the bandwidth constraint. Moreover, interconnection fabric for concurrent trace is high hardware cost since trace signals are distributed more dispersively [10]. Due to the challenge of concurrent trace, it is essential to develop a holistic solution that takes these problems into account in debugging multicore systems.

In this paper, we propose a clustering-based scheme for concurrent trace to address the above problems. The proposed scheme clusters multicore system according to the bandwidth constraint and the distribution of trace sources. With distributed trace buffer in each cluster, multiple concurrent sources can be traced simultaneously. The major contributions of this work are:

- We propose a unified communication framework which implements real-time trace and functional execution simultaneously in the same interconnection fabric. In this way, the interconnection fabric which is only used during debugging is eliminated.
- A clustering scheme is proposed which implements concurrent trace for multicore systems. By clustering multicore system and distributing trace buffer, multiple concurrent sources can be traced simultaneously.
- An algorithm for sharing trace buffer between clusters is proposed, which can further improve the utilization of trace buffer.

The work was supported in part by National Natural Science Foundation of China (NSFC) under grant No.(61106036,60806014, 61076037, 60906018, 60921002, 60831160526,61173006,61103203), National Basic Research Program of China (973) under grant No. 2011CB302503, and in part by China Postdoctoral Science Foundation under grant No.2011M500982 and Foundation of Guangxi Key Laboratory of Wireless Wideband Communication & Signal Processing under grant No. 21103.

978-3-9810801-8-6/DATE12/©2012 EDAA

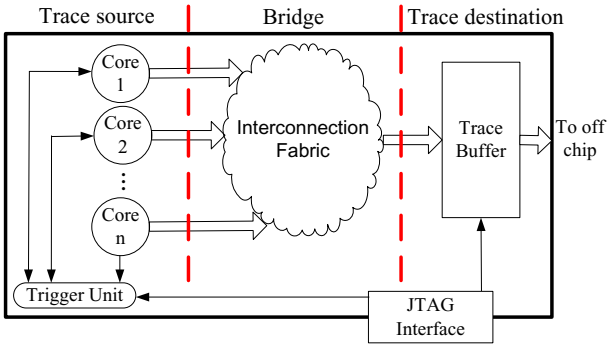


Figure 1: Trace-based post-silicon debug

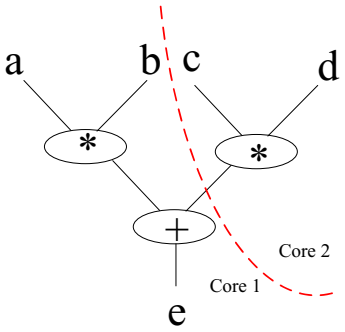


Figure 2: An example requirement for concurrent trace

2. Preliminaries and Motivation

2.1. Real-time Trace

Real-time trace provides a convenient approach to observe the internal signals while the system is running. Compared with scan-based technology, it need not stop functional execution. Moreover, bugs are usually not predictable in multicore systems, whereas scan-based technology cannot deal with the un-reproducible bugs. Consequently real-time trace is of increasing importance for debugging. Fig. 1 shows the framework of trace-based debug in multicore systems. It comprises three components: trace source, trace destination and the bridge (interconnection fabric) between them. When the trigger unit is triggered via JTAG interface, the states of trace sources are transferred into trace buffer by interconnection fabric. In manycore system, trace sources are distributed in different cores and they might be trace simultaneously, which increases the requirement for interconnection fabric and trace buffer. Furthermore, these additional DfD are usually used only during post-silicon debug, which is a crucial hardware cost for on-chip resource.

Prior works have discussed each of the three parts separately. Extracting the correlation of trace signals has been discussed in [3], which aims to select the trace signals which are of the maximum observability. Liu and Xu have proposed a novel interconnection fabric, which reduced the cost of interconnection fabric and can select among many trace signals [10]. Trace buffer mainly has been discussed about its utilization. To improve the utilization of trace buffer, several compression algorithms have been proposed [7, 8]. The cost of on-chip trace resource is an important issue since it takes up on-chip resource and only used in debugging process. More importantly, real-time trace must ensure the data transferring without loss, which becomes more challenging when coming to multicore systems.

2.2. Debugging Multicore System

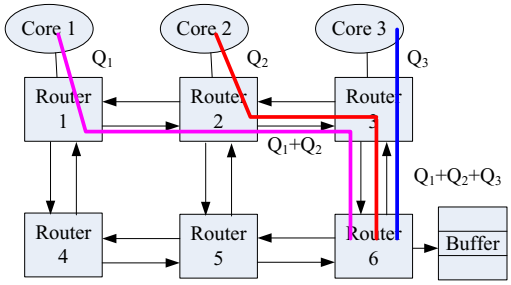
Multicore architecture is becoming a promising framework in today's IC design [11], whereas debug becomes more difficult for the increased parallelism. To trace the propagation of bugs, the appointed signals should be traced in consecutive clock cycles. However, it is difficult to trigger bugs in parallel environment and a great amount of consecutive trace data intensifies the requirement of on-chip interconnection and memory.

The difficulties of tracing multicore system are detailed as follows. Firstly, more signals should be observed at the same time for the parallel property of multicore systems, i.e. more trace data should be transferred and written into trace buffer simultaneously. For the bandwidth constraint, trace data might be lost before being stored. The second challenge is that transmission latency decreases the utilization of trace buffer. We take the example as shown in Fig. 2 to demonstrate it. Assuming the process of $a * b + c * d$ is need to observed, and $a * b$, $c * d$ are executed at Core 1 and Core 2 respectively (their results are added at Core 1). Thus, internal signals of Core 1 and Core 2 both need to be traced simultaneously (concurrent trace). If the latency of transferring the result $c * d$ from Core 2 to Core 1 becomes longer, trace cycles consequently become more to observe the whole calculate process. It means the decrease of trace buffer utilization. The utilization (U) can be expressed as:

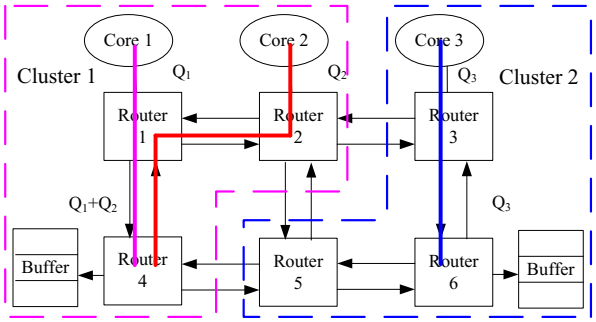
$$U = t_n/(t_n + t_d) \tag{1}$$

where t_n, t_d are the time of normal execution and the additional latency caused by network-on-chip (NoC) transferring respectively. When t_d increases, the utilization decreases. Therefore, the latency is an important metric in multicore system and it should be low for functional data. In Section 4, the results of latency will be shown in detail.

In multicore system, NoC is usually used as communication infrastructure. Reusing NoC in post-silicon debug can reduce the DfD cost. Tang and Xu have proposed a framework to transfer debug data via NoC [12]. However, they have not considered the concurrence of multiple trace sources. In [9], the mechanism is proposed to reuse NoC, but it needs to stop the execution which cannot meet the requirement of real-time trace.



(a) Concurrent trace with centralized trace buffer



(b) An example clustering structure for concurrent trace

Figure 3: Concurrent trace when debugging multicore system

2.3. Motivation

Existing proposals focus on trace signals, interconnection fabric, and trace buffer respectively. In multicore debugging, several challenges are introduced: i) the parallel environment requires to trace multiple sources simultaneously (concurrent trace), and the existing mechanisms of centralized trace buffer cannot meet the requirement of bandwidth constraint; ii) the additional interconnection fabric is high cost for concurrent trace. What is more, the high-cost DfD is only used during debugging; iii) the utilization of trace buffer decreases as the latency increases. Therefore, one relevant question is: can we develop a holistic solution that takes these factors into consideration?

The answer is positive. For example, in Fig. 3a, three cores generate trace data streams concurrently (to simplify the analysis, only three cores are included in this example). Assuming their bandwidth requirements are Q_1, Q_2, and Q_3 respectively, the link between Router 2 and Router 3 needs to reserve at least $(Q_1 + Q_2)$ bandwidth for trace data. Similarly, the link between Router 3 and Router 6 should reserve $(Q_1+Q_2+Q_3)$ to avoid data loss. If the requirement of concurrent signals exceeds the maximum link bandwidth, trace data might be lost and the concurrent trace fails.

We can utilize NoC to transfer trace data with quality-of-service support instead of adding a use-once interconnection fabric. Furthermore, the framework can be clustered and trace buffer is distributed into different clusters. Thus, the scheme can implement concurrent trace and improve the utilization of distributed trace buffer. Fig. 3b shows a possible clustering result for concurrent trace. The three cores are cataloged as different clusters and the requirement of bandwidth decreases.

3. Proposed Scheme for Concurrent Trace

In multicore system, multiple sources are often needed to be observed for parallel programs. Centralized trace buffer cannot meet the requirement of concurrent trace. In this section, a clustering-based scheme is proposed. Firstly, the unified communication framework is presented to support the transmission of both debug data and functional data. Then, a clustering algorithm is described to implement concurrent trace of multiple sources. To improve trace buffer utilization, we also present a buffer sharing algorithm.

3.1. Problem Formulation

Concurrent trace is a challenging problem in multicore system debugging. To formulate the problem, we define the following terms:

Definition 1: signal cell $S\{s_i\}$ is the set of the signals s_i which belong to the same core and need to be traced simultaneously.

Definition 2: concurrent group $G\{S_i\}$ is the set of signal cells S_i that belong to different cores and need to be traced simutaneously.

Definition 3: buffer utilization is the ratio of valid information to all data saved in the trace buffer.

Then, the concurrent trace problem can be formulated as: **Given** Concurrent group $G\{S_j\}$, NoC topology $G(V, E)$, bandwidth threshold Th. **Find** a cluster structure and the placement of distributed trace buffer. **Such that:** (i) trace all concurrent group without data loss under the constraint of bandwidth threshold; (ii) maximize(buffer utilization).

To tackle this problem, we design a clustering-based scheme which catalogs NoC-based multicore system into clusters and trace buffer is distributed in the clusters.

3.2. Unified Communication Framework

In the clustering-based scheme, NoC is used to transfer debug data and functional data concurrently as is called unified communication framework. The principle is to ensure that trace data is transferred without loss and functional data is transferred with the least increased latency. In actual debug, the data of consequent clock cycles are traced into buffer when the predefined conditions are triggered.

Therefore, trace data should be transferred with guaranteed-throughput service to avoid data loss. In NoC-based system, debug data are also packaged with flits which is the same as functional data. The flit injection rate (C_{ir}) is denoted as

$$C_{ir} = x_i * f_c / W_f \tag{2}$$

where x_i is the number of trace signals, f_c, W_f are the frequency of the core and the flit width respectively. Then, the minimum number of flits that should be transferred at every clock cycle can be expressed as:

$$Q_f = C_{ir}/f_r = x_i * f_c/(W_f * f_r) \tag{3}$$

where f_r denotes the frequency of the router. If the bandwidth cannot meet the requirement of Q_f, trace data might be lost. For example, if $Q_f = 0.2$, router interface collects the states of trace signals every five clock cycles to form a flit. Therefore, the network should be of the capability of transferring a flit within the interval of five clock cycles.

In debug mode, partial bandwidth are reserved to transfer trace data with guaranteed throughput. The requirement of bandwidth increases as trace sources when more than one data stream in a link. At the same time, the latency of functional data is also an important metric for the reserved bandwidth of trace data. When requesting switch resource, trace data is of higher priority over functional data. And functional data is transferred with best-effort service, which is used commonly in NoC. In this way, the two different kinds of data streams can be transferred simultaneously according to their special requirements.

Algorithm 1: Clustering NoC-based multicore system

Input: NoC topology $G(V, E)$, concurrent group S, and bandwidth threshold Th
Output: The clustering result

1 **Initiate** each router as a cluster, i.e. C=$\{c_1, c_2, ..., c_n\}$, where $\forall c_i = \{router_i\}$;
2 Set a control variable stop=false;
3 **while** not(stop) **do**
4 Candidate=NULL;
5 **foreach** two clusters $c_i, c_j \in C$ **do**
6 Unite the two clusters, i.e. $c'_i = c_i + c_j$;
7 Update $C' = C - \{c_i, c_j\} + \{c'_i\}$;
8 OverTh = CheckBWThreshold(C', Th, G, S);
9 **if** not(OverTh)
10 Candidate=union(Candidate,$\{[c_i, c_j]\}$)
11 **if** Candidate is empty
12 stop=true
else
13 **foreach** Candidate **do**
14 EdgeNum=CalculateEdgeNumber(c_i, c_j)
15 SignalNum=CalculateSignalNumber(c_i, c_j)
16 $[c_i, c_j]$= $\underset{[c_i,c_j]\in Candidate}{\arg\max}\ (EdgeNum)$ and $\underset{[c_i,c_j]\in Candidate}{\arg\min}\ (SignalNum)$
17 Update C, i.e. $C = C - \{c_i\} - \{c_j\} + \{c_i + c_j\}$;
18 **return** C

Algorithm 2: Sharing trace buffer between clusters

Input: the results of clusters, concurrent trace signals
Output: destination of share trace buffer

1 **Initial** the set of unused trace buffer $B = \{b_1, b_2, ..., b_m\}$
2 Order set B according to buffer sizes
3 Initial the set of used trace buffer $D = \{d_1, d_2, ..., d_n\}$
4 Calculate buffer utilization u_i = traced signal/buffer size
5 **foreach** $b_i \epsilon B$ **do**
6 **foreach** $d_j \epsilon D$ **do**
7 CalculateHops(b_i, d_j)
8 CalculateUtilization(b_i, d_j)
9 Select minimum hops and maximum utilization d_j
10 Share b_i to cluster d_j
11 Update unused buffer B=B-$\{b_i\}$;
12 Update cluster $D = D + \{b_i\}$;
13 return D;

3.3. Clustering NoC-based System

In the unified communication framework, trace data and functional data are transferred on the same interconnection fabric. To achieve guaranteed throughput for trace data, we cluster the NoC-based system for concurrent trace. The goal of clustering NoC is to find a structure which can support concurrent trace in multicore system. In the clustering-based scheme, the routers of the NoC (including the cores attached to the routers) are separated into clusters and trace buffer is distributed to each cluster subsequently. With this cluster structure, all trace data of the cluster is transferred to trace buffer of this cluster.

Algorithm 1 details the process of clustering a NoC-based system. In the process, the set C denotes the current state of clustering NoC. In the initial stage, every router (along with the attached cores) is took as one cluster. In the following, a greedy strategy is applied to combine two clusters as a new cluster in each iteration. The principle of combination clusters is that the smaller and tighter connected clusters are selected preferentially under the requirement of bandwidth threshold (Th). Line 8 ($CheckBWThreshold$) checks whether the combined cluster C‘ exceeds bandwidth threshold Th by applying all concurrent groups onto the NoC-based system. If it does not exceed the threshold (Th), this new cluster can be taken as a candidate (Line 10). If the set Candidate is still empty after all two clusters in C have been processed, the iteration is finished and return the current C as the final result. Otherwise, the algorithm evaluates the number of the edges between the candidate clusters and the number of trace signals. Then, the cluster of maximum edge number and minimum signal number is selected as new cluster into the final cluster set in this iteration (Line 16).

Threshold Th denotes the maximum bandwidth on every link between routers and it is the worst case. In each cluster, there are more than one router can be used to place buffer. The router that is of the smallest hop number to other routers is selected to attach trace buffer of this cluster.

Table 1: Simulation parameters

Parameter	Value
Flits_in_packet	8
Simulate_count	100
Traffic model	Uniform
Arbiter scheme	iSLIP
Number of virtual channel	4
Buffer size of VC	4
Routing protocol	Dimension order
NoC topology	8×8 mesh
Link Width	128 bits

3.4. Sharing Trace Buffer between Clusters

In some actual debug process, only partial concurrent group needs to be traced. For example, in Fig. 3b, if core 3 has no signals to be observed in a special debug case, trace buffer of Cluster 2 can be shared by the other cluster such as Cluster 1. When knowing trace buffer is free, the problem is to determine which cluster can share it.

Algorithm 2 shows the detail of the proposed sharing strategy. The input parameters include concurrent groups, NoC topology and memory distribution. Firstly, the larger buffer has higher priority since it can balance the utilization of trace buffer. Therefore, the unused trace buffer is ordered in Line 2. All used clusters try to share free trace buffer. To each free buffer, two metrics are calculated in Line 7–8, i.e. the hops to one cluster and the buffer utilization. Finally, the cluster which is of the minimum hops and the maximum utilization is selected to share the current free trace buffer.

Trace data are routed to trace buffer of their cluster or the trace buffer that they can share. This mechanism does not design special routing protocols and is scalable for real applications.

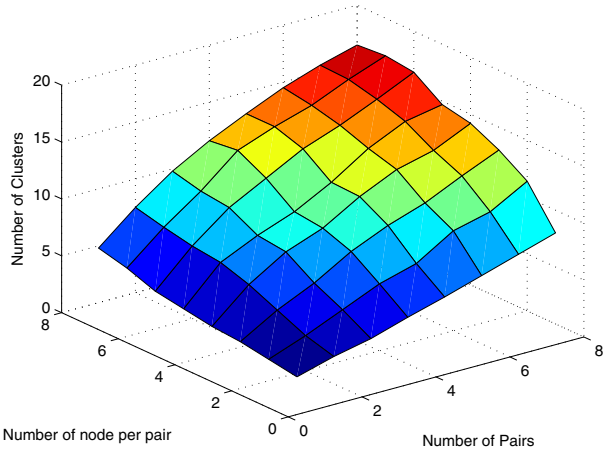


Figure 4: Clusters with various concurrent signals

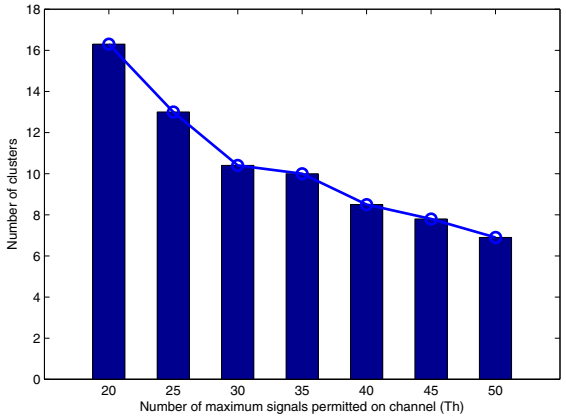


Figure 5: Clusters with various bandwidth threshold (Th)

4. Experimental Results

We implement a simulation based on Booksim [13]. In the simulation, trace data and functional data are concurrently injected into NoC. The parameters of the simulation are set as shown in Table 1. In the following, we will analyze the simulation results of clustering and latency in detail.

4.1. Results of Clustering

In a 8x8 mesh network, we simulate the proposed clustering scheme. As discussed in Section 3, the number of clusters is related to concurrent group and bandwidth threshold (denoted S and Th in Algorithm 1 respectively). In this section, we analyze the clustering results under various concurrent trace signals. Fig. 4 shows the number of clusters under various concurrent group (presented as the number of traced pairs and the average number of nodes in a pair). In the simulation, the threshold Th is set to 20 and the number of trace signals is random from 10 to 20. If no cluster is adopted, two concurrent trace cells (30 trace signals on average) might exceed the threshold of centralized buffer. With the proposed scheme, even if eight groups of trace signals (eight trace cells per group) exist in multicore systems, all the concurrent groups can be traced simultaneously by 17 clusters on average.

To evaluate the impact of bandwidth threshold Th, we fix the trace cells and conduct several experiments. The number of trace cells and the average number of routers in per trace group are both set as eight. Fig. 5 shows the average number of clusters with Th varies from 20 to 50. As can be seen that the number of clusters decreases with the increase of Th. The results can be used to determine the threshold according to memory utilization in actual silicon debug.

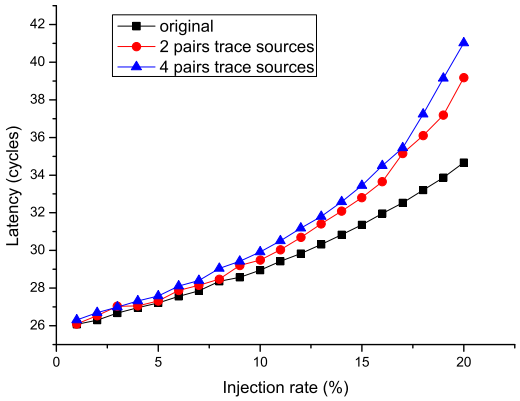


Figure 6: Latency with different trace signal pairs

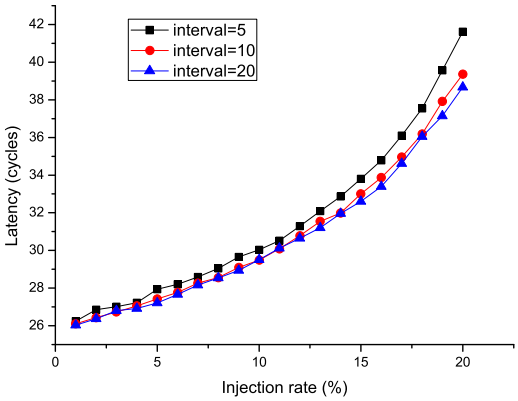


Figure 7: Latency with different injection intervals

4.2. Results of Latency

The proposed scheme eliminates the high-cost interconnection fabric which is used only during debugging. A correlative problem, as discussed in Section 2.2 (Formula 1), is the increased latency of functional data which reflects the impact on the utilization of trace buffer. In this section, we evaluation the latency under various conditions. Fig. 6 shows the latency (cycles) with different injection rate (load). As can be seen that latency increases as the debug pairs increase. Compared with the original latency without trace, the increase is quite small. For example, the latency only increases 1.33% when injection rate is 0.05.

To evaluate the unified communication framework, we obtain the latency of functional data under different injection intervals of trace data. As stated in Section 3 (Formula 3), the injection interval means the speed of injecting trace data into NoC. Fig. 7 shows the results of functional data latency with injection intervals of trace data being 5, 10 and 20 respectively. As can be seen that the increase of latency is very small, especially when the injection rate is low. It demonstrates that the impact to functional data is negligible when trace data is under the bandwidth threshold. Therefore, the proposed framework eliminates the use-once buffer while keeping functional data latency is considerable low.

5. Conclusions

In this paper, we proposed a novel clustering-based scheme for concurrent trace in debugging NoC-based multicore systems. In the proposed scheme, the multicore systems are cataloged into clusters and each cluster is attached with private trace buffer. Furthermore, we design an algorithm to share buffer between clusters, which improves the utilization of trace buffer. In this way, concurrent trace is implemented efficiently and the requirement of interconnection which is used only during debugging is eliminated. Experimental results demonstrate the effectiveness of our solution.

References

[1] M. Abramovici, "In-system silicon validation and debug," IEEE Design & Test of Computers, vol. 25, pp. 216-223, 2008.
[2] J. Keshava, N. Hakim and C. Prudvi, "Post-silicon validation challenges: how EDA and academia can help," in Proc. ACM/IEEE Design Automation Conference, 2010, pp. 3-7.
[3] H. F. Ko and N. Nicolici, "Automated trace signals identification and state restoration for improving observability in post-silicon validation," in Proc. Design, Automation and Test in Europe Conference, 2008, pp. 1298-1303.
[4] E. Singerman, Y. Abarbanel and S. Baartmans, "Transaction based pre-to-post silicon validation," in Proc. ACM/IEEE Design Automation Conference, 2011, pp. 564-568.
[5] H. Yu-Chin, "Maximizing full-chip simulation signal visibility for efficient debug," in Proc. International Symposium on VLSI Design, Automation and Test, 2007, pp. 1-5.
[6] B. Vermeulen and K. Goossens, "Interactive Debug of SoCs with Multiple Clocks," IEEE Design & Test of Computers,, Vol. 28, No. 3, pp. 44-51, 2011.
[7] J. Yang and N. Touba, "Expanding trace buffer observation window for in-system silicon debug through selective capture," in Proc. IEEE VLSI Test Symposium, 2008, pp. 345-351.
[8] E. Anis and N. Nicolici, "Low cost debug architecture using lossy compression for silicon debug," IEEE Transactions on Computers, Vol. 60, No. 7, pp 937-950, 2011.
[9] H. Yi, S. Park and S. Kundu, "On-chip support for NoC-based SoC debugging," IEEE Transaction on Circuits and Systems-I: Regular Papers, Vol. 57, No.7, pp. 1605-1614, 2010.
[10] X. Liu and Q. Xu, "Interconnection fabric design for tracing signals in post-silicon validation," in Proc. ACM/IEEE Design Automation Conference, 2009, pp. 352-357.
[11] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesmh interconnect for a teraflops processor", IEEE Micro, pp. 51-61, 2007.
[12] S. Tang and X. Qiang, "A multi-core debug platform for NoC-based systems," in Proc. ACM/IEEE Design Automation Conference, 2007, pp. 1-6.
[13] http://nocs.stanford.edu/booksim.html