Interconnection Fabric Design for Tracing Signals in Post-Silicon Validation

Xiao Liu and Qiang Xu
CUhk REliable computing laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {xliu,qxu}@cse.cuhk.edu.hk

ABSTRACT

Post-silicon validation has become an essential step in the design flow of today's complex integrated circuits. One effective technique that provides real-time visibility to the circuit under debug (CUD) is to monitor and trace internal signals during its normal operation. Typically, a large number of signals are tapped and a subset of them are selected to be observed in each debug process. These trace signals need to be transferred to on-chip buffers and/or off-chip trace ports for analysis. Existing solutions use multiplexer trees or specific access networks to conduct the above duty, which, however, either provide less visibility to the CUD or result in high hardware cost. In this paper, we propose a novel trace signal interconnection fabric design to tackle the above problem. Experimental results on benchmark circuits show the efficacy of the proposed solution.

Categories and Subject Descriptors

B.7.3 [**Integrated Circuits**]: Reliability and Testing

General Terms

Verification, Design.

Keywords

Post-Silicon Validation, Trace-Based Debug

1. INTRODUCTION

With the ever-increasing design complexity and the ever-shrinking market window for today's integrated circuit (IC) products, it is increasingly difficult to guarantee the correctness of the design solely through pre-silicon verification. Post-silicon validation has thus become an essential step in the design flow of complex ICs, which has a significant impact on the profitability of these products because of its associated time-to-market delay and high re-spin cost [2].

Since the circuit under debug (CUD) is a piece of silicon that has already been fabricated, there is only limited visibility of its internal signals, which makes post-silicon validation an extremely difficult task. One widely-used technique to mitigate this problem is to reuse the CUD's existing test structure (e.g., JTAG and scan chains) to run/stop its operation and observe whether the values in the circuit's storage elements are the expected values [9]. This debug methodology facilitates to identify those easy-to-find bugs that leave "evidences" when the circuit halts at very low hardware cost. Many tricky bugs, however, only manifest themselves after a long period of operations, and it is quite difficult to identify them with the above post-mortem debug strategy. Moreover, the behavior of many bugs is not repeatable, making diagnosis with this run/stop debug methodology even more difficult.

In order to reason the root cause of the design's erroneous behavior effectively, it is invaluable to have real-time visibility to the CUD during its normal operation, which can be obtained by monitoring and tracing the CUD's internal signals [7, 18]. As shown in [1], for million-gate industrial designs, it is common to tap thousands of signals in the circuit and select a subset of them (say, 32 signals) to trace concurrently in each debug process[1] [10, 12]. These trace signals are then transferred to on-chip trace buffers and/or off-chip trace ports for later analysis.

Interconnecting the large number of tapped signals to the trace buffers/ports is not an easy task and involves non-trivial design-for-debug (DfD) overhead. Existing solutions typically use pipelined multiplexer (MUX) trees to conduct the above duty (e.g., [1, 11, 20]). As any signals going through the same multiplexer cannot be observed concurrently, this ad-hoc technique limits the visibility to the CUD. At the same time, since bugs often occur in unexpected scenarios, designers are typically not knowledgeable about exactly which signals should be traced together at the design stage and it is a rather cumbersome process for them to manually build the MUX network that satisfies debug needs.

In this paper, we propose a novel interconnection fabric design to tackle the above problem, which contains two main parts: (i). a *MUX network* that connects those mutually-exclusive tapped signals, which can be designated by designers and/or extracted automatically based on structural analysis; (ii). a *non-blocking concentration network* that is able to transfer any m signals out of n inputs ($m \leq n$) to the trace buffers/ports. With the proposed method, designers are able to flexibly select any trace signal combinations (so long as they are not mutually-exclusive and do not exceed the provided trace bandwidth) in each debug process, which significantly enhances the CUD's debuggability at low DfD hardware cost.

The remainder of this paper is organized as follows. Section 2 reviews related prior work and motivates this paper. The proposed interconnection fabric for tracing signals in post-silicon validation is detailed in Section 3. Section 4 then presents our experimental results on benchmark circuits. Finally, Section 5 concludes this work.

[1] Due to the limited trace bandwidth, it is impossible to trace all the tapped signals at the same time.

DAC'09, July 26–31, 2009, San Francisco, California, USA.

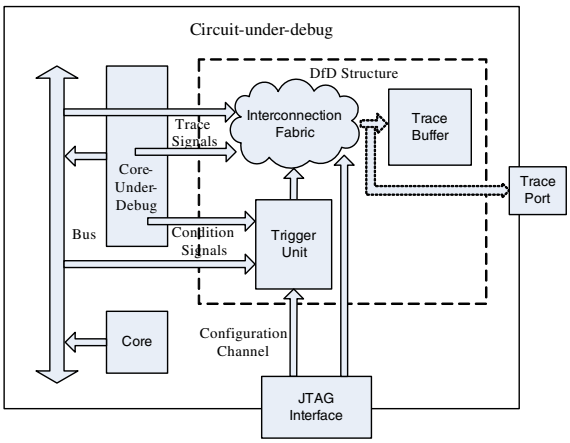


Figure 1: Trace-Based Debug Architecture.

2. PRELIMINARIES AND MOTIVATION

2.1 Trace-Based Debug Methodology

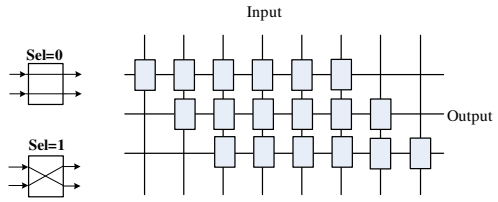
Introducing DfD circuitries to trace the CUD's internal signals provides real-time observabilities into the circuit. This debug methodology facilitates designers to conduct root cause analysis for the abnormal behavior of the CUD and has gained wide acceptance in the industry (e.g., [1, 3, 4, 13, 21]). As designers are not knowledgeable about which part of the design may contain bugs, a large number of signals need to be tapped in the system, typically in the thousand range for million-gate designs [1]. Due to the associated DfD area cost and debug bandwidth requirement, however, it is impossible to *concurrently* trace all the tapped signals. Instead, only a small number of internal signals (say, 32) can be real-time observed together, and it is up to the designers to determine which signals to trace at a specific debug phase, according to the system's erroneous behavior. These signals are then transferred to on-chip trace buffers and/or off-chip trace ports for diagnosis, through an *interconnection fabric* (see Fig. 1). Typically, a trigger unit controls the start and stop of the tracing, in which the triggering mechanism can be configured through JTAG interface through the debug configuration channel [19]. This so-called *trace qualification* process is very useful to reduce the large volume of debug data that are to be analyzed.

Interconnecting a large number of tapped signals to the trace buffers and/or trace ports involves non-trivial design-for-debug (DfD) overhead. In the following, we overview the possible design options for this interconnection fabric in prior work.

2.2 Related Work and Motivation

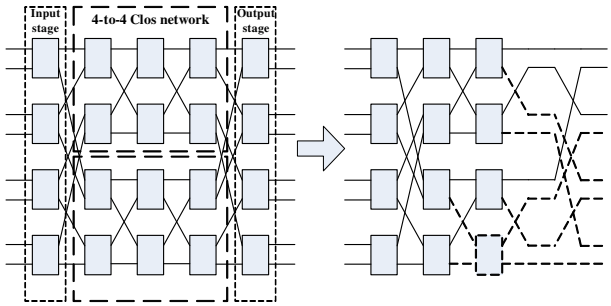
Industrial designs typically use MUX trees to select a subset of the tapped signals to trace in each debug process, in which the control signals to the multiplexers can be configured through the JTAG interface (e.g., [1, 20]). To satisfying the timing constraint for the tracing logic, the MUX trees can be pipelined. In addition, when the tapped signals are coming from different clock domains, first-in first-out (FIFO) buffers and/or flip-flop chains can be used to ensure data safety [1].

To conduct root cause analysis for design bugs effectively, it is desired to have the flexibility to concurrently observe certain related tapped signals (e.g., state elements and those control signals that activate state transitions for finite state machine) under the trace bandwidth limit. MUX-based interconnection fabric, however, limits this flexibility and reduces the visibility to the CUD, as any signals going through the same multiplexer cannot be traced concurrently. It is hence up to the designers to manually build the MUX network based on their design knowledge, which is a rather cumbersome process for them. More importantly, since bugs often occur in unexpected scenarios, designers are typically not knowledgeable about exactly which signals should be traced together at the design stage to satisfy their debug needs.



(a) 2×2 crossbar switch (b) 8-to-3 sparse crossbar concentrator

Figure 2: Sparse Crossbar Concentrator.



(a) 8-to-8 Clos network (b) 8-to-3 Narasimha concentrator network

Figure 3: Narasimha Network.

To overcome the above limitation, designers can resort to *non-blocking concentration networks*, which have been extensively studied in the context of communication theory [16]. A n-to-m concentrator is able to transfer *any* m out of n ($m \leq n$) input signals to the output side, which naturally satisfies our interconnection fabric design needs that tries to select a subset of trace signals from a large number of tapped signals. A well-known concentrator, namely *sparse crossbar concentrator*, is a direct-connected network constructed using crossbar switches. [14] proved that the number of switches required to build such *single-stage* concentrators is at least $(n-m+1) \times m$ and and an example 8-to-3 concentrator is shown in Fig. 2.

Narasimha [15] proposed a multi-stage concentrator that is able to dramatically reduce the required number of crossbar switches. This *Narasimha concentrator* is developed on top of the well-known 3-stage *Clos network* [5]. Clos network is able to provide all possible permutations of the n ($n = 2^k$) inputs at the output side of the network, and it is constructed in a recursive manner. An example 8-to-8 Clos network is depicted in Fig. 3 (a). The 8 input signals are firstly assigned to four 2×2 crossbar switches, and their outputs are connected *evenly* to the upper and lower sub-networks, which are 4-to-4 Clos networks. The output stage is constructed in a reverse manner, and the whole process ends when the sub-network are all 2×2 crossbar switches. More details about the Clos network can be found in [8]. As it is not necessary to provide the "permutation" capability in a concentrator, Narasimha concentrator eliminates the output stage of the Clos network. It also removes those crossbar switches that do not drive outputs. An example 8-to-3 Narasimha concentrator is shown in 3 (b).

Recently, Quinton and Wilton [17] proposed to use the Narasimha concentrator to connect a programmable logic block to the other cores in SoC designs. While the original Narasimha concentrator requires the number of inputs to be 2^k (similar to the Clos network), it can be easily revised to take arbitrary number of inputs, as shown

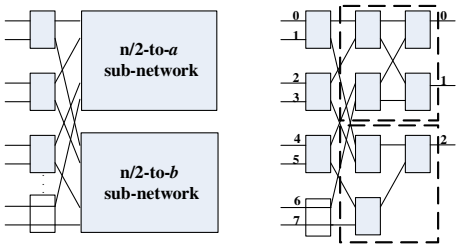


Figure 4: Revised Narasimha Concentrator in [15].

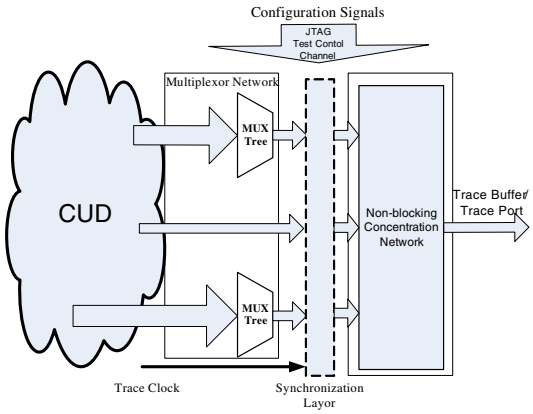


Figure 5: Proposed Interconnection Fabric.

in Fig. 4. [17] also did minor modifications to the Narasimha concentrator design to further reduce its hardware cost. To be specific, [17] proposed to replace the last crossbar unit with two direct wires at the input stage, as shown in Fig. 4. This replacement, however, cannot always provide the desired functionality. A counter example for a 8-to-3 network is shown in Fig. 4. If the last crossbar at input stage is reduced and we would like to trace inputs {0, 1, 7} at the same time, apparently, input 7 should be transferred to output 2 with the lower sub-network. Then, one of the other two inputs cannot find a path to the output. Consequently, the combination of inputs {0, 1, 7} cannot be traced together, which violates the functionality for non-blocking concentrator design.

Apparently, using concentrators to construct trace signal interconnection fabric provides better visibility to the CUD when compared to MUX trees, but at the cost of more DfD area. In practice, some tapped signals are not highly correlated with each other, and hence it is not necessary to observe them concurrently. This observation motivates us to combine MUX trees and concentrators to have a flexible yet low-cost tracing network. In addition, even for the revised Narasimha concentrator, it still contains lots of redundant elements. Take a special case for 8-to-8 network as an example, the simplified concentrator requires 9 crossbar switches to construct it, but these switches actually can all be eliminated. This motivates us to propose a new concentrator design with much lower hardware cost.

3. PROPOSED INTERCONNECTION FABRIC DESIGN

The problem investigated in this paper can be formulated as:

Given N_{tp} tapped signals in the CUD, those highly-correlated signals should be able to be traced concurrently, while others not. We are to design an interconnection fabric to transfer a subset of N_{bf} (typically, $N_{bf} \ll N_{tp}$) tapped signals to the trace buffers/ports at runtime, which satisfies the above requirement at the minimum hardware cost.

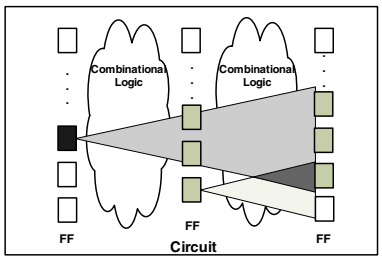


Figure 6: Forward Propagation for Correlation Extraction.

To tackle the above problem, we propose to design the interconnection fabric as shown in Fig. 5, which contains two main parts: (i). a *multiplexer network* that connects those mutually-exclusive tapped signals, which can be designated by designers and/or extracted automatically based on structural analysis. This stage outputs N_{pc} potentially-correlated signals. (ii). a *non-blocking concentration network* that is able to transfer any N_{bf} signals out of N_{pc} inputs to the trace buffers and/or trace ports. One of the main objectives of our design is to minimize N_{pc}. The reason behind this is that, accessing the same signals with MUX tree always results in smaller DfD cost when compared to using nonblocking concentrator, since the latter one requires more resources to achieve the "any combination" objective.

3.1 Multiplexer Network for Mutually-Exclusive Signals

Firstly, we need to determine which tapped signals are highly-correlated and hence may need to be traced together in post-silicon validation. As discussed earlier, it is a rather cumbersome process for the designers to manually conduct this duty. We introduce a simple yet effective method to facilitate this process through circuit structural analysis. For a tapped signal, its *related logic elements* are those that are either in the logic cone that drives this signal starting from inputs or in the logic cone that this signal drives until outputs. For two tapped signals, if there is no overlap between their respective related logic elements, they are not correlated at all. The above constraint is quite stringent and it does not reflect how high the correlation is among tapped signals. We therefore first levelize the sequential elements of the circuit. Obviously, the closer a logic element is to the tapped signal, the more related it is with the signal. Then, instead of checking the overlapping of all the related logic elements of two tapped signals, we only check whether there is any overlap within the neighboring N_l levels, in which N_l is a user-defined value. If there is, we call these tapped signals *highly-correlated* and should be able to be observed together. Otherwise, they are *mutually-exclusive*.

According to the above, we build an *uncorrelation graph* among tapped signals. In this graph, each vertex represents a tapped signal and we add an edge between two vertices if they are not highly correlated. The graph is initialized as a complete graph. The edges are then gradually removed from the graph by conducting forward propagation analysis for the circuit from the first sequential logic level, as shown in Fig. 6. Note, there is no need to conduct backward analysis for a tapped signal, since the correlations are already obtained by those tapped signals in previous logic level through forward propagation, if any.

Our MUX network is composed of a number of MUX trees (see Fig. 5), which are used to connect mutually-exclusive signals and only one of the signals is required to be observed from each MUX tree at a time. In our uncorrelation graph, each MUX tree corresponds to a *clique*. To minimize the number of outputs N_{pc} from the MUX network, it is equivalent to use the minimum number of

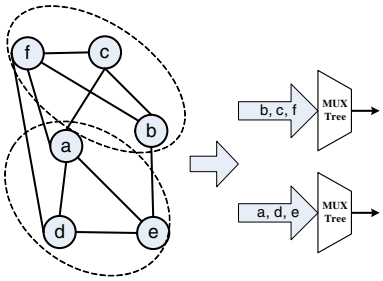


Figure 7: An Example of Uncorrelation Graph.

cliques to cover all the vertices in the uncorrelation graph. This "minimum clique cover problem" is a well-known NP-hard problem in graph theory, and we can resort to a classical greedy heuristic to solve it [6]. As shown in Fig. 7, at least two cliques are required to cover all 6 nodes, and two MUX trees are introduced accordingly to merge these signals and $N_{pc} = 2$.

The above discussions are applicable when all tapped signals are from the same clock domain. If, however, we have tapped signals from different clock domains to transfer to the same trace buffer and/or trace port, we need to build the uncorrelation graph separately for signals from each domain and we need to add a synchronization layer before these signals going into the non-blocking concentrator (see Fig. 5). At the same time, in order not to miss any traced data, it is important to trace signals using the fastest clock among all domains.

3.2 Non-Blocking Concentration Network for Concurrently-Accessible Signals

Starting from the revised Narasimha concentrator that is able to take arbitrary number of inputs as shown in [17] (without using their simplification method as it violates our desired concentration capability, see Section 2.2), we propose several simplification rules to reduce the DfD cost of the concentration network, detailed in the following.

- **Rule 1: Replacement of crossbars that provide redundant path**

As mentioned before, the structure presented in existing work provides redundant paths where signals will never go through. We first introduce a theorem for the output port assignment of the revised Narasimha network, which is missed in [17]. This theorem guarantees that any m out of n input signals are accessible by this concentrator.

Theorem 1 For any n-to-m Narasimha-based concentrator, if the m output nodes are evenly distributed into the top half and the bottom half of the concentrator, it is able to provide n to m accessibility.

Proof: Any two input signals of the crossbar can be connected to the top half and the bottom half of the concentrator, respectively. Therefore, any m out of n input signals can evenly flow into two sub-fabrics with no larger than one difference (i.e., $\{k+1, k\}$ or $\{k, k\}, k = \lfloor n/2 \rfloor$). Recursively, every sub-fabric also has *non-blocking* feature. As a result, if the output nodes are assigned accordingly, the accessibility is guaranteed. □

To propagate the simplification effect, we firstly assign the input and output signal effect on each switch. As depicted in Fig. 8, every switch has a (Input Effect, Output Effect) initialized as (0, 0). Then Input/Output Effect of switches in input/output stage is updated as the number of assigned input/output signal. After that, the Input/Output Effect are propagated forwardly/backwardly. The forward propagation is as follows. Consider one switch at middle stage, if its Input Effect is 2, two back-end connected switches will add 1 for their own Input Effect, since both paths are required to transfer signals. Similarly, when its Input Effect is 1, then only one connected switch will add 1 for its Input Effect (here we choose the upper one). The backward propagation is conducted in the same way but with the opposite direction.

After both propagations, the redundant units in the original concentrator can be identified and simplified as shown in Fig. 8. If both Input Effect and Output Effect are 2 for a switch, it remains to be a crossbar unit. If they are 2 and 1 respectively, the switch is simplified to be a MUX. Otherwise, it is replaced by a wire. With the above simplification rule, for a 9-to-4 concentration network, its cost will be reduced from 16 crossbars to 8 crossbars and 5 MUXes.

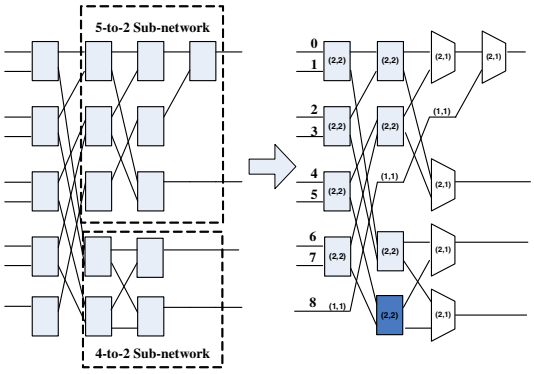


Figure 8: 9-to-4 Network Example of Rule 1.

- **Rule 2: Reduction of crossbar at input stage**

As discussed earlier, [17] proposed to replace the last crossbar unit at the input stage with two direct wires, but it cannot always hold the desired property for concentration network. The following theorem presents the condition for such replacement.

Theorem 2 For any n-to-m network, a crossbar switch at the input stage switch can be replaced by two direct wires, provided both n and m are even.

Proof: We start from the case that n is even. By Theorem 1, at the input stage, m out of n signals can be evenly distributed into two sub-networks (i.e., $n/2$-to-a and $n/2$-to-b, where $|a-b| \leq 1$). We consider the to-be-reduced unit as the one whose two inputs are connected directly to two sub-networks, given m is even. If neither signals from two sub-networks are traced, the replacement does not affect the original transfer capability. If one of them is traced, other units at this stage is able to evenly distribute $m-1$ signals. If both are traced, since they are separated into two sub-networks, the functionality is reserved.

However, if m is odd (say, $a = (m-1)/2$ and $b = (m+1)/2$), suppose the signal connected to the sub-block with a outputs is traced, the other $m-1$ signals should be connected to the sub-networks with $a-1$ and b outputs respectively. Since $b-(a-1) = 2$, this requirement cannot be satisfied.

Similarly, when n is odd, if we trace the last signal (e.g., Input 8 in Fig. 8) and an input that also connects to the top half sub-network (Input 6), the replacement is not applicable. □

We apply this rule to further simplify the fabric at the input stages. For the 9-to-4 network depicted in Fig. 8, only one switch can be reduced in a 4-to-2 sub-network. To note, this rule does not work for the special case that the input stage contains only one switch.

- **Rule 3: Replacement of crossbars that provide unnecessary permutation**

Even after applying previous simplification rules, the structure still unnecessarily cost large amount of crossbar units for n-to-m concentrator, when m is large. Let us use a 8-to-7 network shown in Fig. 9 to demonstrate our simplification rule. Starting from the last stage, it is obvious that the switches here can only change the order of two signals to provide unnecessary "permutation". Consequently, these crossbars can be replaced with two wires as depicted in Fig. 9 (b). Next, from these direct wires, the frontend crossbar switch may be directly connected to output ports with two wires too. Based on the same principle, it can be further replaced and we continue to process the frontend crossbar switches until the input side is reached. The result for this example is shown in Fig. 9 (c) and we are able to remove four crossbar switches.

In particular, for the special case with n-to-n structure, after applying this rule, no crossbar units is required.

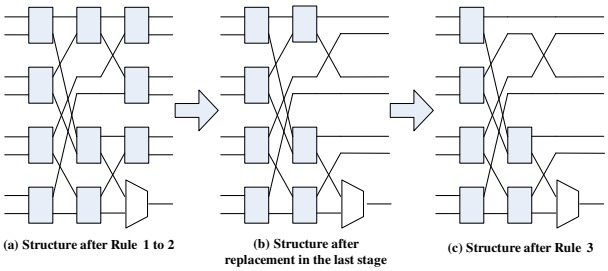


Figure 9: 8-to-7 Network Example of Rule 3.

4. EXPERIMENTAL RESULTS

We conduct experiments on three large ISCAS'89 benchmark circuits to evaluate the DfD cost of the proposed interconnection fabric and compare against existing structures. The correlation constraints are extracted by using the methods presented in Section 4.1 with $N_l = 3$. We randomly select 50, 100, 150, and 200 tapped signals and automatically generate interconnection fabric to connect those signals to trace buffers with various widths. The fabrics are inserted into the benchmark circuits and the DfD cost is obtained by using a commercial synthesis tool.

Table 1 shows the DfD cost introduced by the proposed approach and three existing methods. Column 1 presents the number of tapped signals; Column 2 is the proportion of the edges among all possible connections in the uncorrelation graph; Column 3 is the number of signals output from our multiplexer network; Column 4 is the trace buffer width; Column 5-8 are the DfD area cost in terms of 2-input NAND equivalent gates; and Column 9 shows the DfD area reduction resulted from the proposed method, when compared with that in [17] (again, without applying their simplification method). In this table, *MUX* refers to the MUX tree that does not consider signal correlations. It indicates the lower bound for the DfD area cost to connect N_{tp} signals to N_{bf} outputs. *Sparse* corresponds to the sparse crossbar concentrator design in [12].

As expected, MUX tree and sparse crossbar concentrator costs the least and the most DfD area, respectively. With the growth of the tapped signal count, generally speaking, more DfD area is required for the interconnection fabrics (e.g., s38417 case shown in Fig. 10). When the number of tapped signal is fixed, the cost of MUX tree slightly decreases with the increase of buffer width, because less signals are required to be concentrated. By contrast, we observe proportional increase of the sparse concentrator DfD area with various buffer widths in almost all cases. Similarly, the DfD cost using [17] grows steadily. For the proposed interconnection fabric, however, its area cost depends on not only the signal count

Table 1: Experimental Results for DfD Area Cost

Input #	Edge %	N_{pc}	Buffer width	DfD Cost (2-input NAND gates) MUX	Sparse	[17]	Prop.	Δ
s35932								
50	94.1%	13	8	384	7224	2571	651	74.7%
			16	339	11760	3177	339	89.3%
			32	339	12768	3177	339	89.3%
100	95.2%	13	8	828	15624	5280	1101	79.2%
			16	789	28560	6192	789	87.3%
			32	789	46484	7380	789	89.3%
150	95.2%	28	8	1275	24024	8157	2055	74.8%
			16	1215	45476	9909	2067	79.1%
			32	1110	80084	11223	1110	90.1%
200	95.0%	32	8	1728	32424	10740	2589	75.9%
			16	1662	62276	12660	2685	78.8%
			32	1524	113692	14484	1524	89.5%
s38584								
50	32%	40	8	1533	8399	3746	3210	14.3%
			16	1461	12935	4334	3474	19.8%
			32	1352	13943	4352	3330	23.5%
100	44%	68	8	4444	19234	8871	6456	27.2%
			16	4372	32170	9783	6888	29.6%
			32	4236	49980	10971	7080	35.5%
150	39%	104	8	4301	27106	11215	8213	26.8%
			16	4245	48429	12967	9149	29.4%
			32	4096	83014	14281	9677	32.2%
200	40%	143	8	6466	37197	15514	10840	30.1%
			16	6402	66907	17433	12091	30.6%
			32	6263	118335	19257	13249	31.2%
s38417								
50	69.8%	27	8	528	7374	2721	1233	54.7%
			16	462	11910	3309	1266	61.7%
			32	363	12918	3327	363	89.1%
100	64.6%	60	8	1280	16047	5239	3036	42.1%
			16	1208	28983	6119	3384	44.7%
			32	1026	46794	7307	3492	52.2%
150	67.6%	74	8	1923	24649	8782	4049	53.9%
			16	1851	46039	10534	4607	56.3%
			32	1707	80647	11848	4883	58.8%
200	64.6%	109	8	2314	33025	11341	5622	50.4%
			16	2242	62784	13261	6495	51.0%
			32	2098	114215	15085	7233	52.1%

area overhead reduction $\Delta = (1 - \frac{\text{Area of proposed fabric}}{\text{Area of [17]}}) \times 100\%$

and buffer width but also the correlation constraints. For the case that signals have low correlations (e.g., circuit s35932), the number of signals feeding into the non-blocking concentrator is usually quite small, as they have been processed in the MUX network stage. In this case, the cost of the proposed interconnection fabric can be significantly reduced, without sacrificing the debug flexibility. Taking s35932 as an example, as there are very few correlations among singals, the DfD cost of our interconnection fabric is the same as or slightly higher than that of MUX tree. In average, the DfD cost is reduced up to 83% when compared with [17]. For the case that traced signals are highly related (e.g., s38584), the DfD cost for the proposed design reduces for roughly 27% by applying the simplification rules on non-blocking concentrators. One thing to be noted is that that DfD area cost reported in this table seems to be relatively high when compared to the original circuit size, we attribute this to the large percentage of tapped signals (in practical designs, the percentage is smaller [1]).

To investigate the impact of the number of propagation levels N_l during correlation extraction, we conduct a case study for s35932 when tapping 200 signals and transferring to 16-bit trace buffer. When we set N_l to be 6, N_{pc} increases to 68 and the DfD area cost grows to 4197 2-input NAND gates. If $N_l = 12$, the DfD area cost further grows to 7809 gates. From the above, we can see that building correlation constraints among signals has a significant impact on the cost of the interconnection fabric. In practice, designers should carefully select N_l based on their design knowledge.

We also study the effectiveness of each simplification rule on non-blocking concentrator. In this experiment, we trace 100 sig-

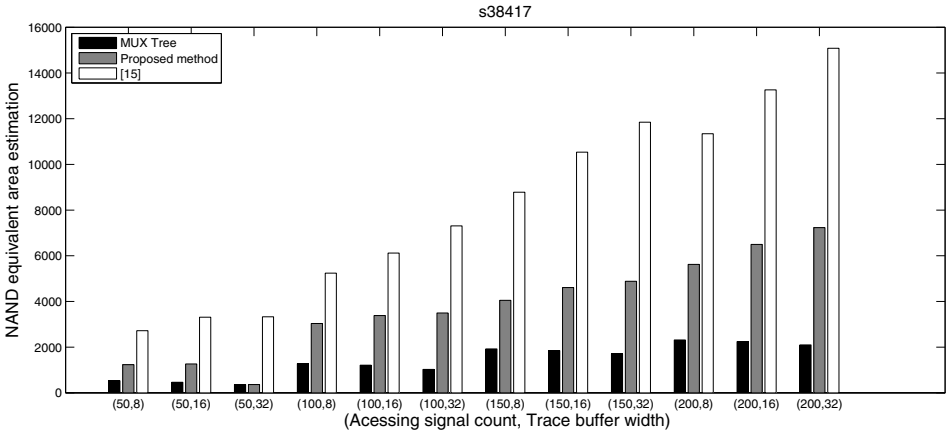


Figure 10: Experimental results for DfD area cost of s38417.

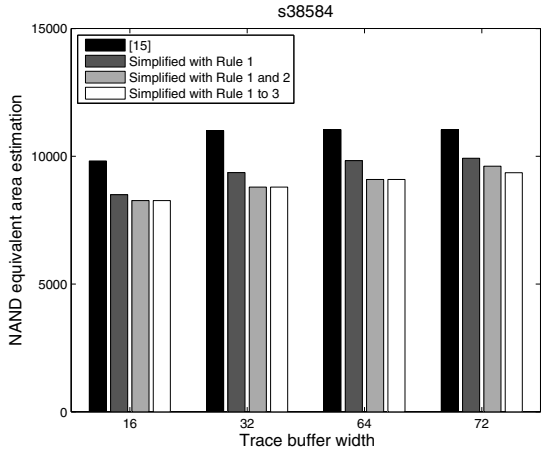


Figure 11: Experimental results for DfD area cost of s38584 for simplification rule evaluation.

nals in s38584 and build the interconnection fabric using the concentrator only. As depicted in Fig. 11, Rule 1 is the most effective one when the fabric contains 32 outputs, because the original fabric contains most redundant paths at this moment. Rule 2 saves most area for 64-bit buffer. It is more effective than 16-bit and 32-bit cases because more crossbar switches at input stages that have not been replaced by MUXes can be reduced by this rule. It is more effective than 72-bit because the rule does not work for the subnetwork containing an odd number of outputs. The impact of Rule 3 is observable only when the output number is 72, since only under such circumstances the switch at the last stage of the fabric is connected with two outputs.

Finally, to address the timing issue, we can pipeline the proposed interconnection fabric by inserting flip-flops into it. Consider the case that tracing 200 signals with 8-bit buffer in s38417. The original proposed fabric introduces a few critical paths. After pipelining, these critical paths can be eliminated, with additional 3199 gates.

5. CONCLUSION

In this paper, we propose a novel trace signal interconnection fabric design that contains a multiplexer network that connects those mutually-exclusive tapped signals and a novel simplified non-blocking concentrator. Experimental results on benchmark circuits show that the proposed solution is able to significantly reduce DfD area cost while satisfying designers' debug flexibility requirement.

6. ACKNOWLEDGEMENTS

This work was supported in part by the General Research Fund CUHK417406, CUHK417807, and CUHK418708 from Hong Kong SAR Research Grants Council, and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

7. REFERENCES

[1] M. Abramovici. In-System Silicon Validation and Debug. *IEEE Design & Test of Computers*, 25(3):216–223, May-June 2008.
[2] M. Abramovici, et al. A Reconfigurable Design-for-Debug Infrastructure for SoCs. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 7–12, 2006.
[3] Altera Inc. Design Debugging Using the SignalTap II Embedded Logic Analyzer.
[4] ARM. Embedded Trace Macrocell Architecture Specification.
[5] C. Clos. A Study of Non-Blocking Switching Networks. *Bell System Technical Journal*, 32:406–424, 1953.
[6] J. Gramm, J. Guo, F. Huffner, and R. Niedermeier. Data Reduction, Exact, and Heuristic Algorithms for Clique Cover. In *Workshop on Algorithm Engineering and Experiments*, pp. 8–94, 2006.
[7] A. B. Hopkins and K. D. McDonald-Maier. Debug Support for Complex Systems On-chip: A Review. In *IEE Proc., Computers and Digital Techniques*, pp. 197–207, July 2006.
[8] F. Hwang. *The Mathematical Theory of Nonblocking Switching Networks* . World Scientific Publishing Company, 1999.
[9] D. Josephson and B. Gottlieb. Debug Methodology for the McKinley Processor. In *Proc. IEEE International Test Conference (ITC)*, pp. 451–460, 2001.
[10] H. F. Ko and N. Nicolici. Automated Trace Signals Identification and State Restoration for Improving Observability in Post-Silicon Validation. In *Proc. IEEE/ACM Design, Automation, and Test in Europe*, pp. 1298–1303, 2008.
[11] H. F. Ko, A. B. Kinsman, and N. Nicolici. Distributed Embedded Logic Analysis for Post-Silicon Validation of SOCs. In *Proc. IEEE International Test Conference (ITC)*, paper 16.3, 2008.
[12] X. Liu and Q. Xu. Trace Signal Selection for Visibility Enhancement in Post-Silicon Validation. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 1338–1343, 2009.
[13] MIPS Technologies Inc. EJTAG Trace Control Block Specification.
[14] S. Nakamura and G. M. Masson. Lower Bounds on Crosspoints in Concentrators. *IEEE Tran. on Computers*, C-31(12):1173–1179, Dec. 1982.
[15] M. J. Narasimha. A Recursive Concentrator Structure with Applications to Self-Routing Switching Networks. *IEEE Tran. on Communications*, 42(234):896–898, Feb.-Apr. 1994.
[16] N. Pippenger. On the Complexity of Strictly Nonblocking Concentration Networks. *IEEE Tran. on Communications*, 22(11):1890–1892, Nov. 1974.
[17] B. R. Quinton and S. J. E. Wilton. Concentrator Access Networks for Programmable Logic Cores on SoCs. In *Proc. International Symposium on Circuits and Systems (ISCAS)*, pp. 45–48, 2005.
[18] S. Tang and Q. Xu. A Multi-Core Debug Platform for NoC-Based Systems. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 45–48, 2007.
[19] B. Vermeulen and S. K. Goel. Design for Debug: Catching Design Errors in Digital Chips. *IEEE Design & Test of Computers*, 19(3):37–45, May 2002.
[20] B. Vermeulen, S. Oostdijk, and F. Bouwman. Test and Debug Strategy of the PNX8525 Nexperia™ Digital Video Platform System Chip. In *Proc. IEEE International Test Conference (ITC)*, pp. 121–130, 2001.
[21] Xilinx Inc. Chipscope Pro Software and Cores User Guide.