On Efficient Silicon Debug with Flexible Trace Interconnection Fabric

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

Trace-based debug solutions facilitate to eliminate bugs escaped from pre-silicon verification and have gained wide acceptance in the industry. Generally speaking, a number of "key" signals in the circuit are tapped, but not all of them can be observed at the same time due to the limited trace bandwidth. Therefore, a trace interconnection fabric is utilized to output either a subset of signals with multiplexor (MUX) network or compressed signatures with XOR network to the trace memory/port in each debug run. However, both kinds of trace interconnection fabrics have limitations. On one hand, with MUX-based fabric, the visibility of the circuit is limited and it requires many debug runs to locate errors. On the other hand, with XOR-based fabric, typically clean "golden vectors" (i.e, without unknown bits) are required so that signatures are not corrupted. In this paper, we propose a flexible trace interconnection fabric design that is able to overcome the above limitations, at the cost of little extra design-for-debug hardware. Experimental results on benchmark circuits demonstrate the effectiveness of the proposed technique.

1 Introduction

Today's complex integrated circuit (IC) designs increasingly rely on post-silicon validation to eliminate bugs that escape from pre-silicon verification [1–3]. One effective post-silicon debug technique is to monitor and trace the behaviors of the circuit under debug (CUD) during its normal operation [4]. As shown in [5], for million-gate industrial designs with trace-based debug support, it is common to tap thousands of "key" signals in the circuit. Since it is impossible to trace all the tapped signals at the same time due to the limited trace bandwidth, a trace interconnection fabric is used to link the large number of tapped signals to the trace buffers and/or trace ports [6].

Existing trace-based solutions typically use multiplexor (MUX) network as trace interconnection fabric, in which we trace a subset of the tapped signals in each debug run. By doing so, we are able to observe these selected signals continuously and use them to reason silicon bugs, e.g., to reconstruct instruction flow [7] or to monitor bus communication protocol [8]. At the same time, however, such MUX-based fabric limits the visibility of the design since it only provides part-view of the CUD in each debug run, while the states of other tapped signals are not visible. Consequently, many debug runs are usually required so that the bug's erroneous effects can manifest themselves, leading to high bug localization effort.

Yang and Touba introduced another kind of trace interconnection fabric in [9]. Instead of observing a subset of the tapped signals in each debug run, it provides an overview of all tapped signals in the CUD by grouping tapped signals with a few XOR networks so that each XOR network compacts the relevant tapped signals into a parity signal, and tracing the compacted signatures comprising of these parities. By comparing the traced signatures with the correct ones calculated from the "golden vectors", we are able to locate the bug's erroneous effects into a group of tapped signals in one debug run, if any. Further diagnosis needs to be conducted to identify the actual erroneous signal(s) among the group of candidate signals, which is not addressed in [9].

One major limitation of XOR-based trace interconnection fabric is that, as a lossy compaction method, it requires *fully-specified* "golden vector" in order not to corrupt the compacted signatures. However, there are many sources of unknown bits (i.e., X-bits) in today's design (e.g., bus contention, multi-cycle paths, and multi-domain interactions [10]), rendering such trace architecture less effective. While various X-tolerant compactors have been proposed for test response compaction in the literature (e.g., [11, 12]), they are not readily applicable for silicon debug. This is because, X-bits in test responses are known a priori and hence it is relatively "easy" to design corresponding design-for-testability

978-1-4673-1595-1/12/$31.00

(DfT) hardware and the associated algorithms to mask or cancel X-bits in test responses. On the contrary, bugs are unpredictable at design time and hence we are not knowledgeable about the vectors to-be-used in silicon debug before tape-out. Therefore, it is essential to have a flexible trace architecture that is able to effectively deal with unknown X-bit distributions in silicon debug.

In this work, we propose a novel signal tracing approach to overcome the limitations of existing solutions. We first present a hybrid trace interconnection fabric that is able to tolerate unknown bits in "golden vectors", at the cost of little extra design-for-debug (DfD) overhead. Then, we introduce a systematic signal tracing procedure to automatically locate erroneous signals with just a few debug runs. Experimental results on benchmark circuits demonstrate that the proposed flexible trace interconnection fabric facilitates more efficient silicon debug than both MUX-based fabric and XOR-based fabric.

The remainder of this paper is organized as follows. Section 2 reviews related works and presents the motivation of our work. In Section 3, we present the proposed flexible interconnection fabric design. The systematic error localization methodology is then detailed in Section 4. Experimental results on benchmark circuits are shown in Section 5. Finally, Section 6 concludes this work.

2 Preliminaries and Motivation

With the ever-increasing design complexity and the ever-shrinking market window for today's IC products, it is increasingly difficult to guarantee the correctness of the design solely through pre-silicon verification, requiring post-silicon validation to catch bugs left in the design. During silicon debug (see Fig. 1), designers try to feed certain test input vectors that can activate the bug and observe their error effects to identify them. Since the circuit under debug is a piece of silicon that has already been fabricated, the main challenge is that there is only limited visibility of its internal signals. Consequently, one of the key issues in silicon debug is how to efficiently capture the error evidences, so that designers can quickly localize the error within a small region of the CUD, and then apply various diagnosis methods (e.g., [13]) to root-cause the bug and fix it.

One widely-used silicon debug technique is to reuse the CUD's existing test structure (e.g., scan chains) to run/stop its operation and observe whether the values in the circuit's storage elements are the expected values [3, 14]. Even though effective for identifying those easy-to-find bugs that leave "evidences" when the circuit halts, this low-cost technique provides little help for tracking those tricky bugs that takes a long period of operation to manifest themselves. Moreover, the behavior of many bugs is not repeatable, making diagnosis with this run/stop debug methodology even more difficult.

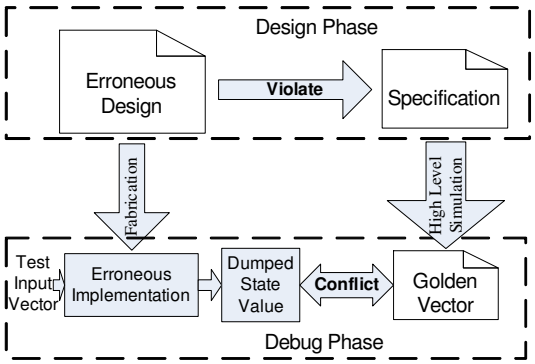

Figure 1. Silicon Debug: An Overview.

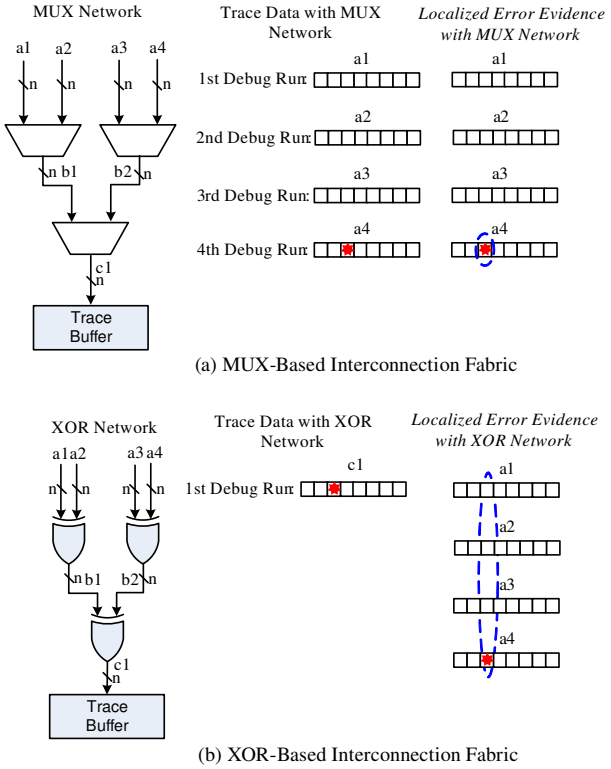

Figure 2. Existing Trace Interconnection Fabric Designs.

Tricky errors may take a long period to be activated in "corner cases" or certain electrical environment, and it is difficult to be caught with run-stop debug solutions [15]. After activation, the error will leave its evidences in the circuit, and the evidences will further propagate forwardly and leave their tracks on some other flip-flops(FFs). Based on this observation, if we are able to observe the error evidences by properly tracing relevant FFs for some time, the region with error is greatly zoomed in, and hence the bug root-cause effort is significantly reduced.

Fig. 3 depicts a conceptual infrastructure for trace-based debug techniques. As signal tracing involves non-trivial DfD overhead, only some "key" signals in the circuit are tapped,

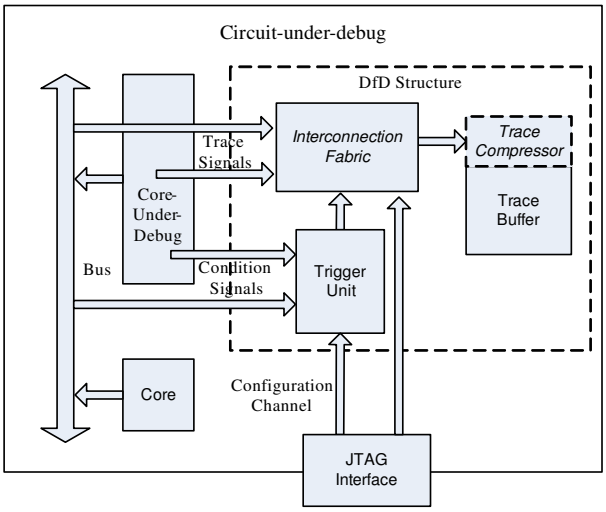

Figure 3. Trace-Based Debug Infrastructure.

typically in the thousand range for million-gate designs [5]. An interconnection fabric is then used to link the tapped signals to the trace buffers/ports. Within the fabric, signals are gradually concentrated according to the limited trace bandwidth requirement. Trace compressor is optionally included to extend the traced information with limited trace buffer. The trigger unit controls the start and stop of signal tracing, in which the triggering mechanism can be configured through JTAG interface [16].

One important issue in trace-based silicon debug is how to select tapped signals, which determines the effectiveness of the debug solutions, i.e., whether we are able to effectively observe bugs' erroneous effects in the circuit. Many solutions have been proposed in the literature to tackle this problem [17–24]. On the other hand, due to the limited trace bandwidth, we are not able to trace all the tapped signals concurrently in each debug run. The trace interconnection fabric design and the corresponding signal tracing methodology therefore determines how efficient we can observe the bugs' erroneous effects (if any), e.g., the number of debug runs to locate them.

One widely-used trace interconnection fabric is the MUX-based fabric, wherein we trace a subset of the tapped signals in each debug run until we find out the erroneous effects. An example MUX-based fabric is shown in Fig. 2(a), wherein we need four debug runs to observe the erroneous effects in $a4$.

To reduce the number of debug runs during silicon debug, [9] proposed an XOR-based trace interconnection fabric (as depicted in Fig. 2(b)), wherein all tapped signals are compacted into signatures[1] and then transfer to trace buffers/ports. For the same example, with XOR-based trace interconnection fabric, we can observe erroneous evidence in the traced signature in the first debug run, but we have four suspicious tapped signals and further diagnosis is needed to identify the actual one (see Fig. 2(b)). Meanwhile, the effectiveness of the XOR-based fabric relies on the existence of *clean* "golden vector" to generate reference signatures for comparison. However, this is usually not the case during silicon debug, rendering XOR-based fabric less effective. This is because: (i). it is often too time-consuming to run gate-level simulation for failed silicon test[2], designers often resort to high-level simulator to generate "golden vectors" and many unknown bits (X-bits) are obtained when they are mapped onto gate-level vectors; (ii). asynchronous clock domains and uninitialized state elements also result in many X-bits in the vectors. In test response compaction techniques, X-bits can also reduce fault coverage by corrupting signature. Many techniques have been proposed to resolve them. One approach called X-blocking is to eliminate the sources of X-bits by interrupting the normal behavior of the circuit [25]. It is not applicable for trace-based debug that targets on the bugs occurring in the CUD's normal operation. X-masking hardware is introduced to mask the X-bits at the input of test response compactor [26]. As masking data is required for every input channel with X-bit in each shift cycle, the hardware overhead for storing masking control data can be expensive. For trace-based debug application, the problem becomes more severe as the number of tapped signals is usually more than that of scan chains and the number of trace cycles is more than that of shift cycle, so that much more control data is required. X-tolerant test response compaction techniques are introduced to inherently tolerate these X-bits [10]. Mitra and Kim proposed a combinational X-tolerant compactor namely X-Compact [11]. By using redundant channel for connecting each input with multiple outputs in the compactor, X-Compact is able to keep a few outputs uncorrupted in the presence of limited X-bits. For this kind of compactor, the associated hardware cost from redundant XOR channels is also expensive to guarantee high X-tolerant capability. More importantly, these XOR-based fabrics cannot restore the original trace data due to the lossy compaction during data transfer. While in debug process, it is essential to localize the exact erroneous signal to conduct further root-cause effort. Above limitations demonstrates it is necessary to introduce a dedicated X-tolerant fabric for debug purpose.

As indicated above, both MUX-based and XOR-based trace interconnection fabric have their own advantages and disadvantages. *A relevant question is whether we can design a hybrid fabric and its corresponding tracing methodology that have the benefits of both solutions while overcoming their limitations?* This has motivated the proposed technique in this paper.

[1]The likelihood for aliasing is quite small, and for the sake of simplicity, it is ignored in this paper.

[2]Running silicon for one second may take days for gate-level simulator to complete.

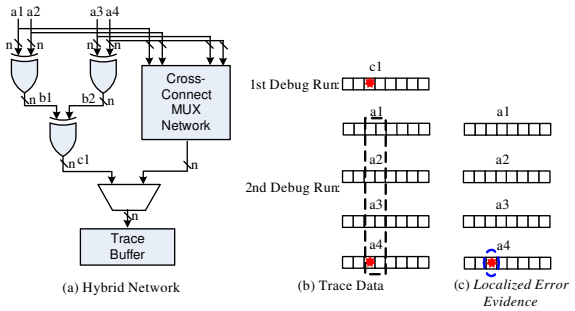

Figure 4. Hybrid Trace Interconnection Fabric: A Straightforward Solution.

3 Proposed Trace Interconnection Fabric

Our initial thought is to design a straightforward hybrid trace interconnection fabric as depicted in Fig. 4. In addition to the XOR network used to trace compacted signatures, a cross-connected MUX network is used to trace tapped signals directly. For the same example shown earlier, after we localize the error evidence on the third-bit of trace data in the first debug run with the trace signature, the fabric enables us to trace the original tapped signals directly and we can find the actual error evidence immediately in the second debug run (see Fig. 4(b)). While the number of debug runs can be reduced with this fabric design, it involves significant routing overhead as tapped signals that may be far from each other in the CUD need to be cross-connected. Moreover, if the "golden vectors" are not clean, we are not able to identify suspicious error signals via the XOR-based tracing in the first debug run and the efficiency of the debug solution is essentially the same as MUX-based fabric.

To tackle the above problem, we propose to have an XOR-MUX cell as the basic unit in our trace interconnection fabric. As depicted in Fig. 5(a), from two groups of input signals, the cell is able to selectively transfer one group of signals or the parities from the two groups. Then, by replacing the MUX or XOR cell in Fig. 2 with the the newly-introduced XOR-MUX cell, we are able to obtain our proposed trace interconnection fabric with the same tree-like structure (see Fig. 5(b)) and the routing cost is similar to existing solutions.

There are two methods to configure the XOR-MUX cells: (i). control every single signal in the cell (i.e., $m = n$ in Fig. 5(a)), referred to as *fine-grained control*); (ii). control a group of signals with one selection signal ($m = 1$ in Fig. 5(a), referred to as *coarse-grained control*). The former one involves more hardware overhead, but it enables more efficient silicon debug with higher flexibility; while the later one is the opposite. As the example shown in Fig. 5(b), with coarse-grained control, we can only trace either $b1$ or $b2$ in each debug run, while with fine-grained control, we can select to trace some signals in $b1$ and some others in $b2$ concurrently. Note that, the configuration is again performed with JTAG interface and it does not incur additional routing overhead.

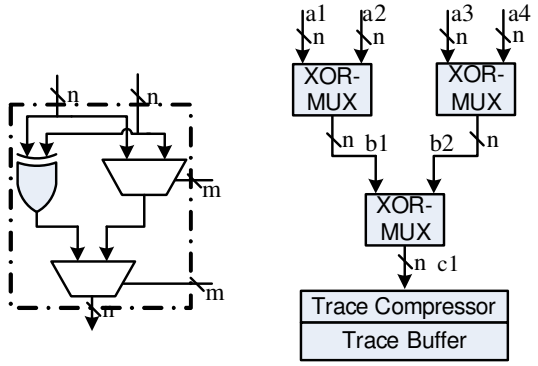

Figure 5. Proposed Trace Interconnection Fabric.

Moreover, our proposed trace interconnection fabric design takes advantage of spatial compaction on trace data only. On the other hand, it is also possible to compress trace data temporally using multiple input signature register (MISR) [27]. To be specific, the MISR will periodically compress several cycles of trace data from interconnection fabric into a signature, and it will dramatically expand the tracing time. In particular, [28] presents a novel X-tolerant temporal compression methodology for trace-based debug. As shown in Fig. 5(b), we also include such temporal trace compressor in the proposed design to selectively compress trace data. Together with the interconnection fabric, we further develop an efficient flow with extensive trace window for error evidence localization in trace-based debug.

Note that, the focus of this work is on how to design the trace interconnection fabric and its corresponding tracing methodology. How to select trace signals (e.g., [18, 19]) and how they are grouped together (e.g., [6]) is hence beyond the scope of this work. Designers can either designate them manually or rely on automated solutions to resolve these problems.

4 Proposed Error Evidence Localization Methodology

With the proposed DfD design, we introduce our signal tracing methodology that enables efficient silicon debug in this section.

As discussed earlier, after a bug in the CUD is activated, it will take some time to propagate its error effects in the circuit and leave error evidences in one or more FFs. Suppose the trace time is sufficiently long and at least one of the tapped signals are left with such error evidence during tracing, the objective of our technique is to pinpoint the erroneous signal and the error occurrence cycle accurately. Moreover, as one debug run can cost significant runtime, it is beneficial to conduct the above process with as few debug runs as possible.

In the proposed error evidence localization methodology, we first configure the DfD design introduced in Section 3 to compress the trace data spatially and temporally. By doing this, we are with the largely extended capability to trace the internal behavior of the CUD. Consider the case that we have trace buffer with N_{period} (e.g., $64k$) depth and the MISR will periodically compress N_{depth} (e.g., $64k$) cycles of trace data into a signature, then we are able to monitor the CUD for $N_{period} \times N_{depth}$ (e.g., $4G$) cycles in total, which is capable to tackle those errors that take extremely long latency to manifest. By running the CUD once only, we can localize the error evidence existing on some tapped signals within the earliest monitored N_{depth} (e.g., $64k$) cycles. Next we will configure the DfD design to bypass the trace compressor and utilize the proposed interconnection fabric flexibly to localize the exact error evidence within the earliest N_{depth} (e.g., $64k$) cycles.

Algorithm for Error Evidence Localization with Clean "Golden Vectors"	
1	Configure the interconnection fabric as XOR network
2	Trace parities from all tapped signals
3	**While** exact error evidence is not localized on tapped signals
4	Configure the fabric to trace the upper level for one group of signals with erroneous parity
5	Calculate the parities for the other group of signals with traced value

Figure 6. Algorithm for Error Evidence Localization with Clean "Golden Vectors".

First, let us consider the simple case that we have clean "golden vectors". Under this circumstance, as shown in Fig. 6, in the first debug run, we will configure the fabric to be an XOR network as shown in Fig. 7 (a), so that we can trace the parities from all tapped signals. Based on that, we can find some error evidences existing in tapped signals with the applied test vectors, if any, by comparing the traced parities and parities from "golden vectors". To further localize the exact erroneous signal, in the following debug runs, we configure the XOR-MUX cell to selectively trace one group of parities at upper level (e.g., $b1$ as shown in Fig. 7 (b)) containing possible error evidence. As for this example, we notice $b1$ is with error in the second debug run and we will continue to configure the fabric to trace $a1$ as shown in Fig. 7 (c). This time, $a1$ is error-free and we can conclude that $a2$ is with error. The actual values can be calculated with $b1 \bigoplus a1$. By conducting the above procedure recursively, we can localize the exact error evidence in $a2$ by taking d debug runs (d is the depth of tree-like fabric). Within the procedure, however, if we choose to trace b2 in the second debug run, we will find that parities in $b2$ are correct, and we can conclude that all signals at the upper levels of $b2$ are also correct. Consequently, we will still trace the upper level signals of $b1$ (i.e., $a1$ or $a2$) and we will localize the same error evidence in $a2$ in the third debug run. Therefore, no matter whether we

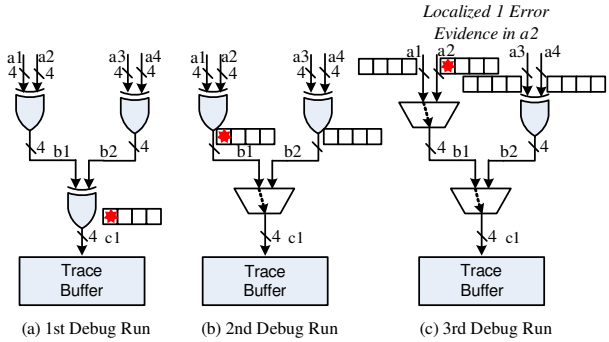

Figure 7. Error Evidence Localization with Clean "Golden Vectors".

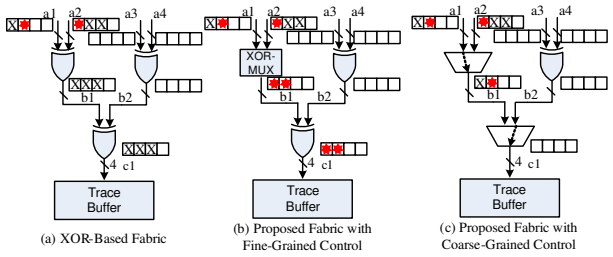

Figure 8. Error Evidence Localization with X-bits in "Golden Vectors".

can directly trace the signal group with erroneous parity or not for each debug run, the above binary search-like procedure guarantees to localize error evidence with d debug runs [3]. Note that, the actual structure may not be the ideal case that each group are with the same amount of tapped signals and each sub-tree are with the same depth. However, the proposed signal tracing methodology is able to work in the same manner.

Next, let us consider the case when X-bits exist in "golden vectors". Unlike XOR-based fabric that cannot tolerate any X-bit (See Fig. 8 (a)), our proposed fabric is able to avoid parity corruption naturally by blocking X-bits during their transfer. As the example shown in Fig. 8 (b), wherein $a1$ and $a2$ are with X-bits in "golden vectors", with fine-grained control fabric that can freely direct every signal, we will configure the XOR-MUX cell to transfer signals with known values only during signal tracing, so that X-bits would have no impact. For this particular example, we are able to tolerate all the X-bits before they corrupt any possible error evidence. Suppose that more signals are with X-bits, we may not be able to block them at the first level (e.g., the same bit position in $a1$ and $a2$ are with X-bits), but eventually we can mask them at lower levels (e.g., in $b1$). For the case with coarse-grained control fabric that can only direct every group of signals together, we can also block the X-bits in one group by configuring the XOR-MUX cells to transfer the other group,

[3] As the proposed interconnection fabric is tree-like structure, d increases logarithmically with the number of tapped signals.

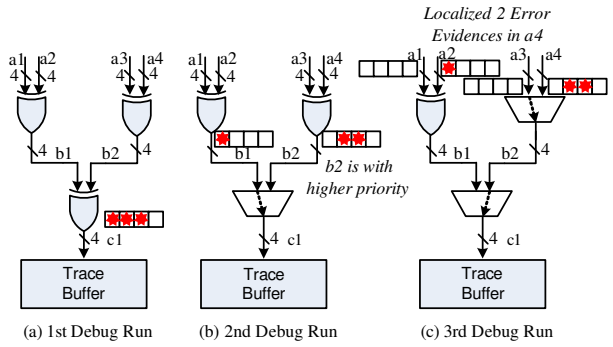

Figure 9. Collecting More Error Evidences during Signal Tracing.

as shown in Fig. 8 (c). However, the error evidences in the masked group will not be observable and may affect the error detection quality.

If the bug manifests itself on multiple tapped signals, it is beneficial to collect more error evidences during signal tracing, as the diagnosis effort will be reduced by studying the possible root-cause region that affects all collected error evidences. For the simple case that we can use fine-grained control to direct every signal, we can directly adopt the earlier approach so that most error evidences are finally collected. This is achieved by repeatedly using the error localization flow for each bit of trace data. As the example shown in Fig. 9, the proposed approach is able to collect three error evidences in $a2$ and $a4$ as these evidences are on different bit position in trace data. For the other case that we are equipped with coarse-grained control fabric, to collect as many error evidences as possible, we will further improve the procedure introduced earlier (see Fig. 7). The main difference is on how to determine the signals to trace in the next debug run. In previous method, we simply decide to trace one signal group on the upper level if the current signal group is with any erroneous parity. Now, for the case that both signal group are with erroneous parity, we will give higher priority to trace the upper level of the signal group with more erroneous parities, so that eventually more error evidences are likely to be collected. As the example shown in Fig. 9 (b), by tracing $b1$ in the second debug run and calculating $b2$ from $b1 \bigoplus a1$, we find $b1$ and $b2$ is with one and two erroneous parities, respectively. We will then choose to trace the upper level of $b2$ as it is with 1 more erroneous parity than $b1$. Finally, we can localize on $a4$ with two error evidences.

5 Experimental Results

5.1 Experimental Setup

We conduct experiments on several large ISCAS'89 and IWLS'05 benchmark circuits to evaluate the effectiveness of the proposed solution. In terms of the interconnection fabrics, we consider MUX-based fabric, XOR-based fabric and the proposed one with coarse-grained control and fine-grained control, respectively. For circuits s38417, s38584 and *usb*, we randomly select 300 tapped signals, while for larger circuits *des* and *ethernet* we tap 1000 signals.

The corresponding signal tracing solutions work as follows. For the one equipped with MUX-based fabric, it simply traces different subsets of signals in each debug run until any error evidence is detected. For the one using XOR-based fabric, it uses one single debug run to trace compacted signatures from all tapped signals. While for the proposed trace interconnection fabric, we try to collect as many error evidences as possible with coarse-grained control and fine-grained control, respectively.

To evaluate these tracing methods' capability on error detection, we randomly generate 1000 errors for each circuit based on the widely-used "cell replacement" model in the literature [13], and each time, we inject one of the errors into the original netlist to obtain the erroneous netlist. We assume for all cases the error has been localized in the same $16k$ cycles by trace compressor. Simulation of $16k$ cycles with random stimuli is then conducted to dump the actual states. These states are compared against the "golden vector" obtained from the simulation with the original netlist to get the propagated error evidences, by finding the difference between actual states and "golden vector". Finally, with different signal tracing methods, a bug is regarded to be detected if the signal tracing solution finds any error evidence.

5.2 Results and Discussion

Tables 1-3 present the experimental results on error detection quality of various tracing solutions when trace buffer width is 8, 16 and 32, respectively. In these tables, Column 2 and Column 3 shows the number of activated errors and the number of detected errors. We can observe that, in average, 66.3% of the injected errors are activated in the experiment, and the conditions to activate other injected errors are not met with simulation for $16k$ cycles. Within the activated errors, around 70% are detected with trace-based debug. This is due to the fact that we only tap a small portion of the signals in the circuits (9.5% to 20.5%) and hence we will miss the error evidences if the bug propagates to other signals. This is also because the tapped signals in our experiment are randomly selected and the detection quality cannot be guaranteed[4].

Columns 4 to 7 present the average number of debug runs (denoted by "# of Debug Runs") for different tracing solutions. Among them, MUX-based fabric (denoted by "MUX" in Column 5) requires the largest number of debug runs (26.5 on average), which means the required tracing time is more than other solutions. With the increase of buffer width, how-

[4]From this perspective, it is essential to develop high-quality trace signal selection methods to increase error detection capability, but this is out of the scope of this work.

Circuit	# of Acti. Error	# of Det. Error	# of Debug Runs				# of Acual/Sus. Error Evidences				Min./Max. Latency			
			XOR	MUX	Pro.C	Pro.F	XOR	MUX	Pro.C	Pro.F	XOR	MUX	Pro.C	Pro.F
s38417	751	447	1	27.6	3.23	3.64	2.58/73.1	1.03/1.03	1.12/1.12	2.24/2.24	2.90/11.6	7.00/7.00	7.06/7.13	6.49/7.80
s38584	825	684	1	19.1	4.47	5.06	3.41/91.3	1.05/1.05	1.12/1.12	2.94/2.94	2.97/17.9	7.30/7.59	7.30/7.59	5.98/9.80
usb	634	222	1	33.5	2.20	2.32	1.78/57.4	1.00/1.00	1.05/1.05	1.70/1.70	3.14/7.83	5.76/5.76	5.91/5.93	5.62/6.09
des	879	854	1	22.8	6.95	6.98	32.2/489.4	1.15/1.15	1.72/1.72	7.28/7.28	2.80/16.2	9.21/9.46	7.85/10.4	4.08/13.6
ethernet	227	88	1	120.3	1.61	1.61	1.89/167.6	1.00/1.00	1.01/1.01	1.43/1.43	1.52/7.04	1.68/1.68	1.68/1.70	1.61/1.82

Table 1. Experimental Results on Error Detection Quality Evaluation (Buffer Width=8).

Circuit	# of Acti. Error	# of Det. Error	# of Debug Runs				# of Acual/Sus. Error Evidences				Min./Max. Latency			
			XOR	MUX	Pro.C	Pro.F	XOR	MUX	Pro.C	Pro.F	XOR	MUX	Pro.C	Pro.F
s38417	751	447	1	14.2	2.85	3.23	2.77/45.5	1.08/1.08	1.23/1.23	2.65/2.65	3.65/10.4	7.00/7.04	6.95/7.07	6.49/7.91
s38584	825	684	1	9.9	3.89	4.39	3.57/59.2	1.20/1.20	1.12/1.12	3.70/3.70	3.27/17.2	7.28/7.89	7.11/8.03	5.76/10.0
usb	634	222	1	17.2	1.99	2.11	1.86/33.3	1.05/1.05	1.10/1.10	1.91/1.91	3.45/7.58	5.76/5.78	5.89/5.97	5.70/6.15
des	879	854	1	11.7	6.12	6.12	32.1/482.3	1.56/1.56	2.68/2.68	13.2/13.2	2.75/16.1	8.62/10.2	6.55/11.5	3.61/14.5
ethernet	227	88	1	60.6	1.53	1.53	2.06/95.2	1.04/1.04	1.04/1.04	2.06/2.06	1.55/6.44	1.68/1.68	1.68/1.68	1.61/1.82

Table 2. Experimental Results on Error Detection Quality Evaluation (Buffer Width=16).

Circuit	# of Acti. Error	# of Det. Error	# of Debug Runs				# of Acual/Sus. Error Evidences				Min./Max. Latency			
			XOR	MUX	Pro.C	Pro.F	XOR	MUX	Pro.C	Pro.F	XOR	MUX	Pro.C	Pro.F
s38417	751	447	1	7.76	2.50	2.79	2.96/25.9	1.19/1.19	1.41/1.41	3.02/3.02	4.43/9.96	7.00/7.14	6.95/7.21	6.68/8.00
s38584	825	684	1	5.50	3.30	3.73	4.33/38.9	1.40/1.40	1.80/1.80	4.56/4.56	3.68/16.4	7.06/8.08	6.71/8.52	5.67/10.3
usb	634	222	1	9.40	1.77	1.89	2.02/15.6	1.12/1.12	1.25/1.25	2.09/2.09	4.09/7.54	5.77/5.78	5.73/5.89	5.71/6.15
des	879	854	1	6.38	5.27	5.27	32.6/442.1	1.96/1.96	4.20/4.20	22.1/22.1	2.73/16.2	7.96/10.8	5.51/12.2	3.23/15.3
ethernet	227	88	1	31.3	1.44	1.44	2.02/53.2	1.07/1.07	1.07/1.07	1.85/1.85	1.55/5.94	1.68/1.68	1.68/1.68	1.68/1.82

Table 3. Experimental Results on Error Detection Quality Evaluation (Buffer Width=32).

ever, the associated tracing capability is enhanced and the required debug runs for this solution is decreased in linear manner. XOR-based fabric (denoted by "XOR" in Column 4) conducts signal tracing for one debug run for all cases. While for both of our proposed solutions with coarse-grained control (denoted by "Pro.C" in Column 6) and fine-grained control (denoted by "Pro.F" in Column 7), only 3.27 and 3.47 debug runs are needed for detecting error evidences. Both are much less than MUX-based fabrics. This is because for the case that no error evidence exists, the proposed solution will stop at the first debug run as no erroneous parity is found, while the one with MUX-based fabric takes many debug runs for tracing every tapped signal to make the conclusion. On the other case that some error evidences do exist, the number of required debug runs is also small as it is strictly bounded by the depth of the tree-like interconnection fabric.

The average number of actual/supicous error evidences obtained from various tracing solutions is shown in Columns 8 to 12 (denoted by "# of Acual/Sus. Error Evidence"). Clearly, the tracing solution with MUX-based fabric and our proposed solutions are able to guarantee every suspicious error evidence is the actual one. While for the method with XOR network, only 5.9% of the suspicious error evidences are the actual ones, which results in higher diagnosis effort. In terms of the capability of collecting actual error evidence, the proposed solution with coarse-grained control and the one with fined-grained control collect 29.1% and 303% more error evidences than MUX-based fabric, respectively.

Columns 13 to 17 present the average value of minimum/maximum latency from the suspicious error evidences to the injected bugs (denoted by "Min./Max. Latency") for various tracing solutions. We observe that the evidences from the solution with XOR-based fabric are with the highest latency variance due to the large amount of suspicious errors. While for other solutions that already obtain the actual error evidences, the diagnosis process can be conducted immediately and the diagnosis effort is usually determined by the error with the minimum latency from the bug. As observed from Columns 14 to 17, within these tracing solutions, the proposed one with fine-grained control obtains the closest error evidences from the injected bug, which means the corresponding diagnosis workload is the lightest.

Buffer Width	Δ_{XOR}	$\Delta_{Pro.\ C.}$	$\Delta_{Pro.\ F.}$
8	-0.09%	3.88%	9.90%
16	0.07%	1.86%	5.13%
32	0.06%	0.91%	2.63%

$$\Delta = \frac{Extra\ Hardware\ Cost}{MUX-based\ Hardware\ Cost} \times 100\%$$

Table 4. Area Overhead of Different Tracing Solutions.

The next experiment evaluates the error detection quality of various tracing solutions when X-bits exist in provided "golden vectors". Here, we randomly inject a certain ratio of X-bits in trace data to compare the X-tolerant capability with different tracing solutions for benchmark circuit s38417. As shown in Fig. 10, by applying different tracing solutions on the original trace data without X-bits, the number of detected error are similar. After that, as more X-bits are injected,

Interconnection Fabric	Estimated Hardware Cost
MUX-based	$(N_{tap}/N_{width}-1)(9N_{width}+20)$
XOR-based	$(N_{tap}/N_{width}-1)(11N_{width})$
Pro.C.-based	$(N_{tap}/N_{width}-1)(29N_{width}+40)$
Pro.F.-based	$(N_{tap}/N_{width}-1)(69N_{width})$

N_{tap}: the number of tapped signals
N_{tap}: trace buffer width

Table 5. Area Estimation of Different Interconnection Fabrics.

the error detection quality with the MUX-based fabric is not significantly reduced, because the method uses many debug runs to trace the data from different signals, and any remaining error evidences (i.e., not X-bits) will still be detected. However, for the tracing solution using XOR network, the X-bits will be easily corrupted and its error detection quality is greatly reduced with the growth of X-bits. As shown in Fig. 10, the method will not detect any error if the ratio of X-bits is greater than 10%. On the other hand, our proposed solution shows its strong capability on tolerating X-bits. With our fine-grained control tracing method, the error detection quality is slightly affected with X-bits. Even for the case with 20% X-bits in trace data, the number of detected errors is almost as high as the one with MUX-based fabric with much fewer debug runs. Even for our coarse-grained control tracing method that consumes less DfD overhead, its error detection quality is slowly dropped with the growth of X-bits, and remains to be considerably high with 5% X-bits in trace data.

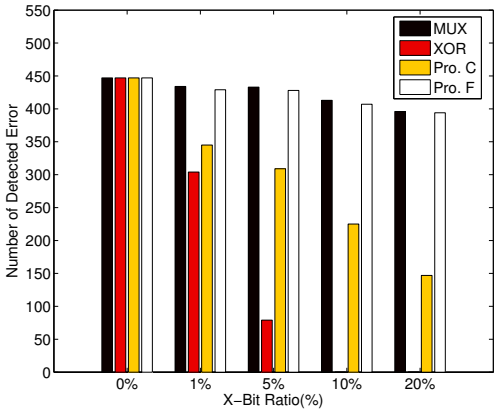

Figure 10. Error Detection Quality Evaluation with X-Bits on s38417.

Finally, to evaluate the area cost for the different tracing solutions, we implement and synthesize the DfD hardware for circuit *des* using commercial EDA tools for the conventional MUX-based fabric, XOR-based fabric and the proposed one with coarse-grained and fine-grained control, respectively. The number of tapped signals is fixed as 1000. As shown in Table 4, we compare the hardware cost (in NAND2 gate equivalent) of the other solutions against the conventional MUX-based fabric, assuming the trace buffer depth to be 16k and the trace buffer width to be 8, 16 and 32, respectively. As indicated in Column 2, XOR-based fabric consumes similar hardware cost as MUX-based fabric. Meanwhile, the proposed solution with coarse-grained control (in Column 3) takes less than 4% extra cost of the total DfD cost by the conventional MUX-based fabric, which is quite small. Even for the solution equipped with fine-grained control (in Column 4), the extra overhead is kept to be less than 10%. This is due to the fact that interconnection fabric only takes a small portion of the DfD area in the trace-based debug infrastructure, and although the proposed solutions require more hardware cost on interconnection fabric, the overall DfD area will not increase significantly. On the other hand, as the trace buffer width increases, the tree-like interconnection fabric of proposed solution requires less XOR-MUX cells to conduct the signal concentration and hence the corresponding hardware cost further decreases. We also list the estimated area cost for the different interconnection fabric as shown in Table 5. The cost is calculated based on the synthesis result of basic units (e.g., MUX, XOR, etc.). Firstly, we can observe that the cost of all fabrics increases linearly when more signals are tapped (i.e., N_{tap} increases). This demonstrates the cost efficiency of the tree-like interconnection fabrics. Secondly, we can conclude that the proposed fabric introduces about 2.2 and 6.7 times more area cost than the conventional MUX-based fabric, for with coarse-grained control and fine-grained control respectively. Again, as discussed on the results in Table 4, because the interconnection fabric only takes a small part in the trace-based debug infrastructure, the overall DfD area only increases slightly.

6 Conclusion

In this paper, we propose a flexible and low-cost trace interconnection fabric design for silicon debug. Combining with the corresponding automatic signal tracing method, the proposed solution is able to tolerate X-bits in "golden vectors" and facilitates to accurately locate erroneous signals and their occurrence cycles within a few debug runs. Experimental results on benchmark circuits demonstrate the efficacy of proposed technique.

7 Acknowledgements

This work was supported in part by the General Research Fund CUHK418111 from Hong Kong SAR Research Grants Council (RGC) and in part by RGC Grant Direct Allocation under grant No. 2050488.

References

[1] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A Reconfigurable Design-for-Debug Infrastructure for SoCs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 7–12, July 2006.

[2] A. B. Hopkins and K. D. McDonald-Maier. Debug Support for Complex Systems On-chip: A Review. In *IEE Proceedings, Computers and Digital Techniques*, pages 197–207, July 2006.

[3] D. Josephson and B. Gottlieb. Debug Methodology for the McKinley Processor. In *Proceedings IEEE International Test Conference (ITC)*, pages 451–460, October 2001.

[4] Q. Xu and X. Liu. On Signal Tracing in Post-Silicon Validation. In *Proceedings IEEE Asia South Pacific Design Automation Conference (ASP-DAC)*, pages 262–267, 2010.

[5] M. Abramovici. In-System Silicon Validation and Debug *IEEE Design & Test of Computers*, 25(3):216–223, May-June 2008.

[6] X. Liu and Q. Xu. Interconnection Fabric Design for Tracing Signals in Post-Silicon Validation. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 352–357, 2009.

[7] S.B. Park, T. Hong, and S. Mitra. Post-Silicon Bug Localization in Processors Using Instruction Footprint Recording and Analysis (IFRA). *IEEE Transactions on Computer-Aided Design*, 28(10):1545–1558, 2009.

[8] M. Boule, J. Chenard, and Z. Zilic. Assertion Checkers in Verification, Silicon Debug and In-Field Diagnosis. In *Proceedings International Symposium on Quality of Electronic Design (ISQED)*, pages 613–620, 2007.

[9] J.-S. Yang and N. A. Touba. Enhancing Silicon Debug via Periodic Monitoring. In *Proceedings IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 125–133, 2008.

[10] S. Mitra, M. Mitzenmacher, S.S. Lumetta, and N. Patil. X-Tolerant Test Response Compaction. *IEEE Design & Test of Computers*, 22(6):566–574, 2005.

[11] S. Mitra and K.S. Kim. X-Compact: An Efficient Response Compaction Technique. *IEEE Transactions on Computer-Aided Design*, 23(3):421–432, 2004.

[12] N.A. Touba. X-canceling MISR ł An X-tolerant methodology for compacting output responses with unknowns using a MISR. In *Proceedings IEEE International Test Conference (ITC)*, pages 1–10, 2007.

[13] K. H. Chang, I. L. Markov, and V. Bertacco. Fixing Design Errors with Counterexamples and Resynthesis. In *Proceedings IEEE Asia South Pacific Design Automation Conference (ASP-DAC)*, pages 944–949, 2007.

[14] G. Rootselaar and B. Vermeulen. Silicon Debug: Scan Chains Alone Are Not Enough. In *Proceedings IEEE International Test Conference (ITC)*, pages 892–902, September 1999.

[15] K. H. Chang, V. Bertacco, and I. L. Markov. Simulation-based Bug Trace Minimization with BMC-based Refinement. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 1045–1051, 2005.

[16] 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.

[17] K. Basu and P. Mishra. Efficient Trace Signal Selection for Post Silicon Validation and Debug. In *Proceedings IEEE International Conference on VLSI Design (ICVD)*, 2011.

[18] H. F. Ko and N. Nicolici. Automated Trace Signals Identification and State Restoration for Improving Observability in Post-Silicon Validation. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1298–1303, 2008.

[19] X. Liu and Q. Xu. Trace signal selection for visibility enhancement in post-silicon validation. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1338–1343, 2009.

[20] X. Liu and Q. Xu. On Signal Tracing for Debugging Speedpath-Related Electrical Errors in Post-Silicon Validation. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 243–248, 2010.

[21] X. Liu and Q. Xu. On Multiplexed Signal Tracing for Post-Silicon Debug. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1–6, 2011.

[22] S. Prabhakar and M. S. Hsiao. Multiplexed Trace Signal Selection Using Non-Trivial Implication-Based Correlation. In *Proceedings International Symposium on Quality of Electronic Design (ISQED)*, pages 697–704, 2010.

[23] H. Shojaei and A. Davoodi. Trace Signal Selection to Enhance Timing and Logic Visibility in Post-Silicon Validation. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 168–172, 2010.

[24] J.-S. Yang and N. A. Touba. Automated Selection of Signals to Observe for Efficient Silicon Debug. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 79–84, 2009.

[25] L.T. Wang, C.-W. Wu, and X. Wen. In *VLSI Test Principles and Architectures*. Morgan Kaufmann, 2006.

[26] M. C.-T. Chao, S. Wang, S.T. Chakradhar, and K.-T. Cheng. Response Shaper: A Novel Technique to Enhance Unknown Tolerance for Output Response Compaction. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 80–87, 2005.

[27] E. Anis and N. Nicolici. Low cost debug architecture using lossy compression for silicon debug. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 225–230, 2007.

[28] X. Liu F. Yuan and Q. Xu. X-Tracer: A Reconfigurable X-Tolerant Trace Compressor for Silicon Debug. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 555–560, 2012.