# On Signal Tracing for Debugging Speedpath-Related Electrical Errors 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
[‡]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
Email: {xliu,qxu}@cse.cuhk.edu.hk

## ABSTRACT

*One of the most challenging problems in post-silicon validation is to identify those errors that cause prohibitive extra delay on speedpaths in the circuit under debug (CUD) and only expose themselves in a certain electrical environment. To address this problem, we propose a trace-based silicon debug solution, which provides real-time visibility to the speedpaths in the CUD during normal operation. Since tracing all speedpath-related signals can cause prohibited design for debug (DfD) overhead, we present an automated trace signal selection methodology that maximizes error detection probability under a given constraint. In addition, we develop a novel trace qualification technique that reduces the storage requirement in trace buffers. The effectiveness of the proposed methodology is verified with large benchmark circuits.*

## 1. INTRODUCTION

With the ever-increasing design complexity for today's integrated circuit (IC) products, it is increasingly difficult to ensure the design correctness of the first silicon solely through pre-silicon verification techniques, such as simulation and formal 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, 7].

The duty of post-silicon validation is to eliminate various errors escaped from pre-silicon verification, which can be broadly classified as functional errors and electrical errors. Functional errors, being repeatable, are relatively easy to be identified and resolved by designers [13, 15]. Electrical errors, however, only occur in certain electrical environment during normal operation and they may take millions of cycles to expose themselves. Hence, they are extremely challenging to be repeated and root-caused. For example, conducting silicon debug in test environment (using automatic test equipment) may simply result in "No Trouble Found (NTF)" because the CUD's behavior is quite different in system's functional environment. Existing solutions mainly resort to designers' own experiences to identify such errors (e.g., by analyzing voltage-frequency shmoo plot [8]), which cannot guarantee the debug quality and efficiency.

To alleviate the limitation of the above manual process, we propose to monitor and trace those internal signals in the CUD that are related to its electrical abnormal behavior. Such real-time visibility facilitates us to identify the root cause of the electrical errors. At the same time, however, since designers cannot afford very large design for debug (DfD) overhead used for tracing [4, 10, 14], we can only tap a small number of signals in the design and trace a subset of them for a limited amount of cycles in each debug iteration.

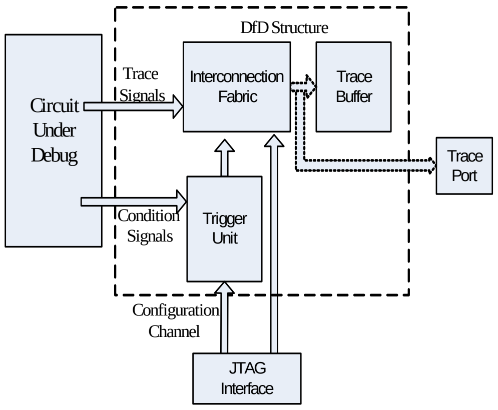

**Figure 1: Trace-Based Debug Infrastructure.**

Electrical errors often lead to reduced operational frequency of the CUD, due to parasitic coupling noises between wires, power supply noise, and/or insufficient driving strength, etc. In this work, we first model the behavior of such speedpath-related electrical errors. Accordingly, with given DfD constraint, we propose an automated trace signal selection methodology to maximize the error detection probability with a subset of signals relevant to the targeted speedpaths. Moreover, to reduce the storage requirements for tracing, we develop a novel trace qualification technique that employs reconfigurable logic to store useful traced data only when the errors are detected. The proposed technique hence significantly improves the utilization of the trace buffer. To the best of our knowledge, this is the *first* trace-based solution for debugging electrical errors in general logic circuits in post-silicon validation. With the proposed technique, we can detect speedpath-related electrical errors at its root-caused site, on the exact error occurrence cycle, without requiring any supporting "golden vector", which are often unavailable when debugging tricky errors.

The remainder of this paper is organized as follows. Section 2 reviews related work and motivates this work. Section 3 describes the speedpath-related electrical error model and equations to obtain corresponding visibility. In Sections 4 and 5, we illustrate the proposed signal selection and data qualification techniques in detail. Experimental results on benchmark circuits are then shown in Section 6. Finally, Section 7 concludes this work.

## 2. PRELIMINARIES AND MOTIVATION

Trace-based debug solution facilities designers to observe abnormal behavior of circuits in operational mode and conduct root-cause analysis, and hence has been widely adopted in the industry (e.g., [1, 3, 5, 16]).

Fig. 1 depicts the general trace-based debug architecture [17]. At design stage, a number of internal signals are selected to be tapped. Then, during debug phase, part of the tapped signals are transferred through interconnection fabric (usually MUX tree) to on-chip trace buffer or off-chip trace port. To save DfD cost, designers can only afford to tap a small number of signals. Therefore, it is essential to select those signals that can provide a *better view* of the CUD to help designers root-cause bugs [9, 11, 18]. In addition, to save trace bandwidth requirement, trace qualification techniques (e.g., using trigger unit to control the start/stop of tracing) and trace data compression methods are often utilized [4, 7].

During post-silicon validation, electrical bugs are the most difficult to resolve because their occurrences are sensitive to certain electrical environment. In [12], the authors proposed to locate electrical errors in microprocessor by tracing the footpoint of instruction flow. However, this method cannot be applied to debugging general logic circuits. In this paper, we consider those electrical bugs that cause the performance degradation for general logic circuits. Such bugs often occur on the critical paths in the circuit that determine its maximum operational frequency, known as speedpaths. To understand such electrical bugs and root cause them, it is essential to make the relevant signals visible during post-silicon validation.

Due to DfD cost considerations, designers can only afford to tap a subset of the relevant signals to speedpaths and trace some of them concurrently during each debug run. Consequently, the effectiveness of this debug strategy highly relies on which signals are selected to trace in the circuit. Existing trace signal selection methods (e.g., [9, 11]), however, are not applicable because they implicitly assume the timing correctness of the circuit so that the traced circuit state can be used to reason a large amount of untraced signals. At the same time, speedpath-related bugs are activated only when the corresponding speedpaths are sensitized. Therefore, if we conduct continuous tracing, the trace buffer can easily become full without any useful traced data that actually activate bugs.

The above limitations motivate the proposed trace-based debug solution for speedpath-related electrical bugs, including new trace signal selection algorithms to maximize the detection probability for such bugs and novel trace qualification techniques to efficiently utilize trace bandwidth.

## 3. OBSERVING SPEEDPATH-RELATED ELECTRICAL ERRORS

### 3.1 Speedpath-Related Electrical Error Model

While speedpath-related electrical errors can be caused by various reasons (e.g., insufficient driving strength or excessive coupling noises), they all behave similarly as causing excessive delays on critical paths. Consider the circuit shown in Fig. 2 as an example. During one clock cycle in the circuit's operation, a falling transition is propagated along the path. That is, the logic value (e.g. '0' in Fig. 2) should be propagated through the path to the endpoint (i.e. output or flip-flop) and the value should be latched after the clock cycle. However, due to electrical bugs in the circuit, this value does not arrive at the endpoint at the end of the clock cycle, and if we observe an opposite latched value (e.g., '1' in Fig. 2), we can conclude that a speedpath-related electrical error occurs.

From this example, the detection of speedpath-related electrical error requires us to monitor the *propagation behavior* of the path. As for this example, to *determine* the propagation of value '0', together with the start point signal *a_in2*, we also need to observe side-input *b_in1* of Gate *b* on the path. This is because, to determine the value propagation through each logic gate on the path, when the to-propagate value is a "controlling" value (as '0' for Gate *a*, *c*), it can definitely propagate through the gate. Otherwise, the event only happens when all other "side-inputs" (e.g., *b_in1*) are visible as "non-controlling" values (they are referred as *objective signals* hereafter). Finally, we also need to observe *c_out* for error detection. To conclude, observing logic '0', '0', and '1' on signals *a_in2*, *b_in1* and *c_out* during the CUD's normal operation implies the occurrence of electrical bugs. In other words, observing the above three signals is necessary for us to detect the electrical error that causes slow propagation of logic '0' on this path when it occurs.

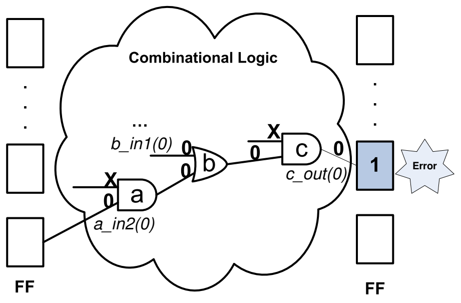

**Figure 2: Modeling Speedpath-Related Electrical Error: An Example**

### 3.2 Speedpath-Related Electrical Error Detection Quality

While several evaluation metrics for trace signal selection has been introduced in [9, 11], those works implicitly assume the timing correctness of the circuit and hence cannot be used to debug electrical errors. We therefore define a few new metrics in this work to evaluate the effectiveness of trace signals for electrical errors, as summarized in Table 1.

To be specific, we denote by $SV1/SV0$ the selective visibility of the signals, which obviously is '1' for traced signals and '0' for unselected ones. To note, above probabilities do not represent the probability that value '0'/'1' is *actually* observed on the node (denoted as *visibility* $V1/V0$). For example, suppose the signal remains '0' in most of time, we can barely observe value '1' on it provided it is traced. Hence, we define *visibility* with Eq. (1) and Eq. (2), where $P1/P0$ is the probability for the signal to be logic '1/0' in functional mode. In this work, we calculate $V1/V0$ by assuming some control inputs are pre-set as '1'/'0' to insure the CUD is working in functional mode, while all other inputs are with the probability 0.5 to be value '1'/'0'.

$$V1 = SV1 \times P1 \tag{1}$$

$$V0 = SV0 \times P0 \tag{2}$$

With these notations, we calculate the visibilities of internal signals with forward propagation based on those observable probabilities of FFs with the following equations. Note that, these equations are based on the assumption that all inputs are independent for each logic gate.

For AND gate (a,b–input, c–output)

$$V0(c) = V0(a) + V0(b) - V0(a) \times V0(b) \tag{3}$$

$$V1(c) = V1(a) \times V1(b) \tag{4}$$

For OR gate (a,b–input, c–output)

$$V0(c) = V0(a) \times V0(b) \tag{5}$$

$$V1(c) = V1(a) + V1(b) - V1(a) \times V1(b) \tag{6}$$

For XOR gate (a,b–input, c–output)

$$V0(c) = V0(a) \times V0(b) + V1(a) \times V1(b) \tag{7}$$

$$V1(c) = V0(a) \times V1(b) + V1(a) \times V0(b) \tag{8}$$

With this information, we then introduce a metric to evaluate the error detection quality. It is important to note that the occurrence of electrical error is not predictable at design stage, while can be

| | |
|---|---|
| Selected Visibility ($SV1/SV0$) | For selected trace nodes, $SV1 = SV0 = 1$; otherwise $SV1 = SV0 = 0$ |
| Functional Probability ($P1/P0$) | The probability that the node is '1'/'0' in functional mode |
| Visibility ($V1/V0$) | The probability that the value '1'/'0' is **actually** observed on the node |
| Propagation Visibility ($PV1/PV0$) | The probability to detect value('1'/'0') propagation based on traced signals |
| Propagation Occurrence Probability ($POP1/POP0$) | The probability that value('1'/'0') propagation occurs |
| Detection Quality ($DQ1/DQ0$) | The probability that the value '1'/'0' *propagation* is detected when the *propagation* occurs |

**Table 1: Terminologies for Visibility Calculation.**

detected only when the value propagation behavior is monitored. We therefore resort to the propagation detection quality to evaluate the error detection effectiveness. For this purpose, the Propagation Visibility (PV) is expressed by

$$PV0/PV1 = \prod V(i) \quad i \in \{\text{start point, objective signals}\} \tag{9}$$

Recall the example in Fig. 2. The required visible signals are start point of the path and the *objective signals*. In addition, as discussed earlier the corresponding to-be-visible value of objective signal should be "non-controlling" one with its relevant gate (e.g., '0' of $b\_in1$ with Gate $b$). In particular, the required visible of start point simply depends on the propagation type ('0'/'1'). With these observations, we define the Detection Quality(DQ) as the conditional probability that the value '0'/'1' *propagation* is detected under the condition that the *propagation* happens, namely,

$$DQ0 = \frac{PV0}{POP0} \tag{10}$$

$$DQ1 = \frac{PV1}{POP1} \tag{11}$$

wherein the probability that '0'/'1' *propagation* occurs (POP) is calculated by assuming all path relevant signals are independent as,

$$POP0/POP1 = \prod P(i) \quad i \in \{\text{start point, objective signals}\} \tag{12}$$

Clearly, the above value is 100% when all relevant signals in the driving cone of targeted path are visible.

## 4. TRACE SIGNAL SELECTION

With the terminologies defined in Section 3, the trace signal selection problem studied in this section is: *Given a set of targeted speedpaths*[1]*, how to set SV to be '1' for a constrained number ($N_{TAP}$) of tapped signals (FFs and/or input signals), so that the circuit's total error detection quality ($TDQ = \sum(DQ0 + DQ1)$) for all targeted potential speedpaths is maximized.*

We solve the above problem progressively as follows. Firstly, we extract the relation between to-be-selected signals and the objective signals required to be observed for error detection (e.g., $b\_in1$ in Fig. 2) to guide the selection procedure (Section 4.1). Based on this information, we then select a *minimum* set of signals to guarantee each targeted speedpath with *non-zero* error detection quality (Section 4.2). Finally, more signals are selected to increase error detection quality under the tapped signal quantity constraint (Section 4.3).

### 4.1 Relation Cube Extraction

Since the objective signals locate in internal combinational logic, while the to-be-traced signals are state elements surrounded by them, it is essential to extract the relationship between these two sets of signals to guide signal selection. Otherwise, we may blindly choose trace signals that have little impact on the visibility of objective signals. One straightforward thought is to conduct symbolic simulation to obtain the exact logic relationship. This method, however, is not only time-consuming, but more importantly, makes deriving the sensitization probability with a subset of the relevant signals difficult. In this work, we propose to use *relation cube* to represent the visibility for objective signals in a concise and effective manner.

Each relation cube denotes that when all signals in the cube are traced, the corresponding value of targeted signal can be visible. An example is shown in Fig. 3, where the relation cube corresponding to signal C(1) is $\{FF1, FF2\}$. We define three atomic operations (*merge*, *concatenate* and *copy*) on relation cube for its gate-level propagation. In our example, since A(1) and B(1) equals C(1), $FF1$ merges $FF2$ to be $\{FF1, FF2\}$ for observing C(1); while either A(0) or B(0) leads to C(0), $\{FF1\}$ *concatenates* $\{FF2\}$ and we have two relation cubes $\{FF1\}$, $\{FF2\}$ for observing C(0). As for *copy* operation, it is simply used when propagating the cube through an inverter or a BUFFER gate.

Based on the above, we conduct circuit-level structural analysis to extract the relation cubes for all objective signals. A pre-processing step is utilized for finding those candidate trace signals within the fan-in cones of objective signals. These signals are then initialized with two cubes corresponding to logic value '0'/'1', as shown in Fig. 3. We then propagate the cubes forwardly in the combinational logic. During the process, we conduct gate-level cube propagation on every newly reached logic element, until all required objective signals are processed. To note, in order to reduce the memory cost which grows exponentially with circuit size, we dynamically remove the relation cubes when the corresponding gate has fully propagated its relation cubes to all gates in its fan-out.

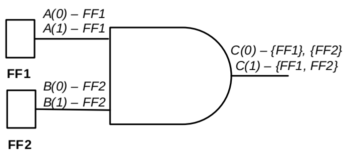

**Figure 3: Relation Cube Extraction: An Example.**

### 4.2 Signal Selection for Non-Zero-Probability Error Detection

After the above process, we have obtained a set of relation cubes containing candidate trace signals for observing '0'/'1' on each objective signal. With this information, this section is concerned with selecting a minimum set of signals out of all the candidates to guarantee every targeted path with **non-zero** detection quality.

From Eq. (9) and Eq. (10), a speedpath can be monitored only when the visibilities of all its objective signals are non-zero. In other words, at least all the signals in one relation cube of each objective signal should be traced. We propose a heuristic to achieve this objective with minimum number of selected signals, and its flowchart is shown in Fig. 4.

To effectively utilize trace signals for monitoring multiple paths at the same time, those paths within the same sequential level of the circuit are put into one group to be considered together. Then

[1]These speedpaths can be designated by designers or automatically extracted from the design. Considering the inaccurate delay model used in timing analysis and process variation effects, speedpath identification itself is a challenging problem, but it is beyond of the scope of this paper. Interested readers may refer to [6].

for each group, the starting point and ending point of every targeted speedpath are selected, which is the basic requirement for monitoring errors on the path. Then, we gather all the objective signals and their corresponding relation cubes. In each trace signal selection iteration, we first find the set of unselected candidate signals for every relation cube, and we record the sets with minimum size from all cubes of every objective signal. Then among all these recorded sets, we choose to select the signals in one type set so that most signals will become visible. We then go back to remove newly visible objective signals and select another set of signals. The process terminates when all objective signals are visible.

With the above selection procedure, we are able to guarantee nonzero detection quality for each targeted speedpath by tracing a small number of signals. More than that, inherently the detection quality is high. This is because, we take high priority to utilize small-sized relation cubes to observe each objective signal. The probability of such event to occur tends to be high since it depends on the combination of a small number of signals.

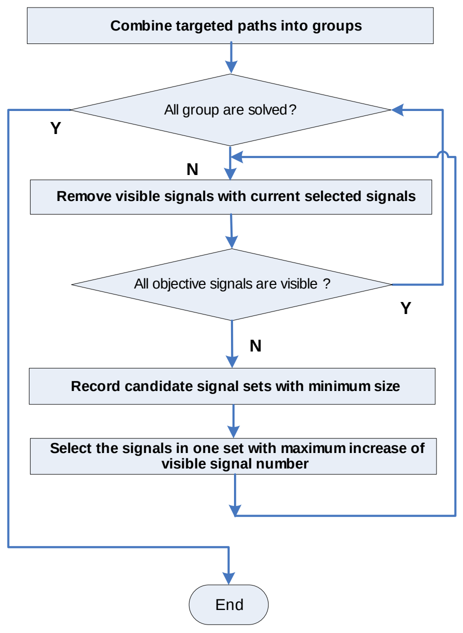
Combine targeted paths into groups
All group are solved?
Y
N
Remove visible signals with current selected signals
All objective signals are visible ?
Y
N
Record candidate signal sets with minimum size
Select the signals in one set with maximum increase of visible signal number
End

**Figure 4: Flow of Signal Selection for Non-Zero-Probability Error Detection.**

### 4.3 Trace Signal Selection for Error Detection Quality Enhancement

Suppose more signals are allowed to be traced, we can use them to further improve the total error detection quality, that is, to maximize TDQ. The selection process works in a greedy manner. Because the objective signals are affected by different sets of candidate signals, we cannot evaluate the detection quality increment induced by every candidate signal. Instead, with the relation cubes extracted previously, we first determine the number of candidate signals $N_{cs}$ that can be selected simultaneously for detection quality improvement. In other words, if the selected trace signal count is less than $N_{cs}$, the error detection quality is guaranteed not to increase. We obtain $N_{cs}$ by parsing all relation cubes of the objective signals to find out the minimum missing signal number, such that if these "missing" signals are further selected, relevant relation cubes can be completed. Sequentially, we parse the cubes again to find out the corresponding missing signal sets and evaluate their impact on the error detection quality. The signals in the candidate set with the maximum TDQ increment will be selected. This procedure repeats until the total selected number reaches the predefined quantity constraint $N_{TAP}$.

## 5. TRACE DATA QUALIFICATION

Speedpaths in the circuit may not be sensitized often. Consequently, if we simply trace their relevant signals continuously, it is very likely that we end up with the data stored in the trace buffer without any useful information for error detection. We hence design a novel trace qualification module to store traced data only when slow-propagation error is found to occur on speedpaths.

The block diagram of the proposed trace qualification module is shown in Fig. 5. We buffer the trace signals for two cycles inside this module. When tracing for a particular speedpath, the two buffered data for its start point are firstly compared to detect whether a transition occurs on it. If not, there is no need to store the traced data. Otherwise, we rely on the slow propagation detection module to detect error, one value propagation module ('0'/'1') decided by the start point of previous cycle will assert when error is detected ("Error Assert"=1 in Fig. 5), and a formatter is utilized to temporarily store the traced signals and align them into trace buffer. Meanwhile, the timestamp generated from counter is also stored into the buffer for recording the error occurrence cycle.

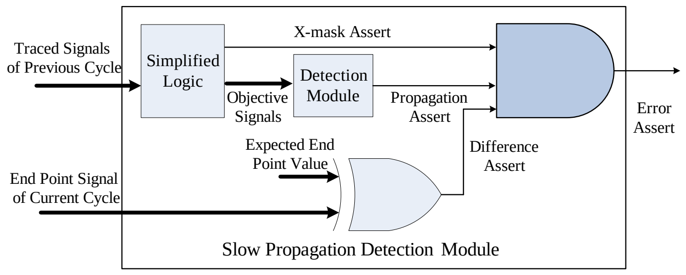

**Figure 6: Block Diagram of Slow Propagation Detection Module.**

To be specific, Fig. 6 describes the slow propagation detection module, which contains simplified logic and a detection block. The simplified logic can be treated as duplicating part of the CUD, while keeping all paths from traced signals to the objective ones. This is obtained by simply parsing the circuit twice, with marking propagated logic elements forwardly starting from traced signals and backwardly from objective ones. Then only relevant logic element marked by both propagations is kept in the simplified logic to achieve the above objective, as indicated in Fig. 7.

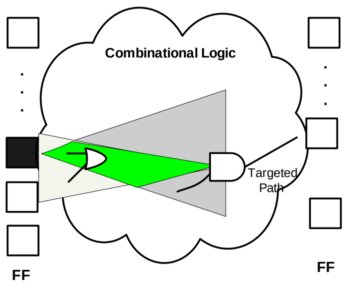

**Figure 7: Circuit Parsing for Generating Simplified Logic.**

In addition, due to the *unknown* side-inputs that may affect the logic calculation on the kept logic (e.g. *or* gate in Fig.7), we should modify the normal logic elements to facilitate "3-valued" logic calculation (e.g., $X$ *and* $1 = X$). To be specific, every 1-bit wire is replaced with 2-bit one, and logic '1' is encoded as "11", '0' is encoded as "00" and unknown side-input 'X' is encoded as "01" or "10". By duplicating traced signals to 2-bit width and replacing normal logic elements with corresponding enhanced ones (designed as standard module) , we can obtain "3-valued" states on objective signals. If any one of them is 'X', it means the value is invisible from untraced relevant signals, and the "X-mask" signal asserts as

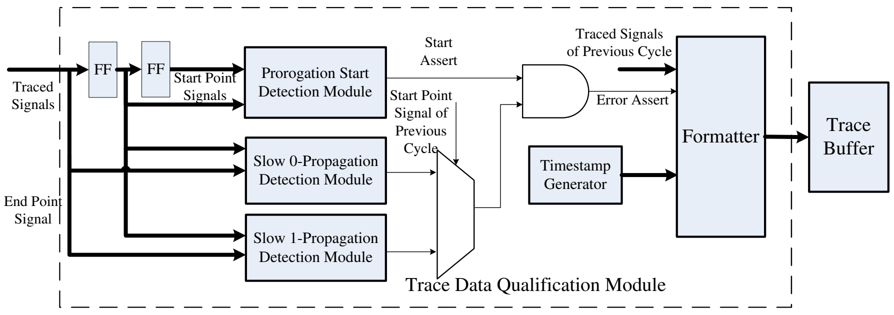
Traced Signals
End Point Signal
FF
FF
Start Point Signals
Prorogation Start Detection Module
Slow 0-Propagation Detection Module
Slow 1-Propagation Detection Module
Start Assert
Start Point Signal of Previous Cycle
Traced Signals of Previous Cycle
Error Assert
Timestamp Generator
Formatter
Trace Buffer
Trace Data Qualification Module

**Figure 5: Block Diagram of Trace Data Qualification Module.**

| Circuit | Total Signal # | Relevant Signal # | Sel. Signal # | Unmon. Pro. # | Detected Pro. Event # | Occurred Pro. Event # | Detection Quality | Time (s) |
|---|---|---|---|---|---|---|---|---|
| s38584 | 1464 | 198 | 69 | 1 | 7718 | 9519 | 81.1% | 45.5 |
| s38417 | 1664 | 394 | 157 | 50 | 630 | 1216 | 51.8% | 125.3 |
| DMA | 3818 | 1482 | 99 | 16 | 6827 | 7615 | 89.6% | 150.3 |
| usb | 2085 | 117 | 59 | 0 | 696 | 696 | 100% | 77.4 |
| des | 9341 | 132 | 95 | 46 | 67097 | 70145 | 95.6% | 668.3 |

**Table 2: Detection Quality Evaluation of Signal Selection for Non-Zero Visibility on 50 Paths.**

'0' to denote the path is not monitored. Otherwise, when the objective signals are all required values (e.g., $b_{in1} = 0$ in Fig. 2) that can determine value propagation, the detection module will output '1' on "propagation assert" signal. Meanwhile if the latched value of end point in current cycle is different from the expected propagated one (obtained with logic simulation), the module asserts error signal.

Since it is not possible to trace all the tapped signals to monitor all targeted speedpaths concurrently during the debug process, we propose to implement the trace qualification module (as shown in Fig. 5 and Fig. 6) with reconfigurable logic, which is configured to detect errors on *single* targeted speedpath in each debug run. Consequently, the size of the trace qualification module is constrained by tracing a single speedpath only (the most complex one), hence reducing the associated DfD area cost. In addition, this structure can be easily extended to monitor multiple speedpaths in each debug run. Instead of simply duplicating the original module into several copies for monitoring each path, we can design the simplified logic shared by multiple targeted paths. This is feasible when several speedpaths can be grouped to share lots of logic elements in their fan-in cones. Note that, we might need to pipeline the proposed fabric to guarantee timing correctness of the monitoring cricuit.

## 6. EXPERIMENTAL RESULTS

We conduct experiments on several large ISCAS'89 and IWLS'05 benchmark circuits to evaluate the effectiveness of the proposed solution. We consider 50 critical paths in each circuit and we simulate them for 20,000 clock cycles. These experiments are conducted on a 2.13 *GHz* PC with 2*GB* RAM.

Table 2 presents the result when we select the minimum number of signals to guarantee every targeted path with *non-zero* error detection quality. Column 1 shows the name of circuit; Column 2 is the total number of state elements in the circuit; Column 3 is the number of signals relevant to targeted speedpaths; Column 4 reports the number of selected signals with proposed method. We evaluate the detection quality by simulating both the original circuit and its internal behavior from the selected trace signals only, which results in a *partial* view of the circuit for each cycle. We then calculate the detection quality (Column 8) as the ratio between the detected propagation events on targeted paths (Column 6) and the total number of propagations that actually occur (Column 7). There are two different propagations on each critical path, that is, the start point can be '0' or '1'. The total number of propagation types that are completely missed to be monitored by tracing selected signals is shown in Column 5, referred to as Unmon. Pro. # in the table. Finally, Column 9 is the CPU time.

Generally speaking, the proposed method is able to achieve satisfactory detection quality with a small portion of speedpath-related signals. For circuit usb, all propagation events on the targeted paths are captured with only 59 signals (i.e., 2.83% of the total number of state elements in the circuit), and hence there is no need to select more signals for detection quality improvement. For circuit s38584, tracing 69 signals (34.8% of relevant ones) guarantees all paths are visible except for one value propagation event. For circuit s38417 and des, 50 and 46 paths are unmonitored, respectively. This is because the selected signals cover a few relation cubes in objective signals, while the happened events are caused by other cubes. For circuit DMA, with only 99 out of 1482 candidate signals, we detect nearly 90% of the propagation events, while 16 value propagations are not monitored. The selection on the largest circuit des cost 668.3s only, which demonstrates the efficiency of this procedure.

We then conduct further signal selection for detection quality improvement. The results on detection quality and unmonitored propagation quantity by incrementally selecting 10 more percent signals are plotted in Fig. 8 and Fig. 9 respectively. This procedure terminates when 95% detection quality is reached for circuits s38584, s38417 and DMA. For circuit s38584, the detection quality approaches 100% with only 30% more signals (i.e., 18 signals). Meanwhile, the propagation that is not monitored in previous step becomes visible. For circuit s38417, the detection quality increases dramatically with 10% more signals (from 51.5% to 70.8%). Also, the number of unmonitored propagations reduces sharply during the early stage of further selection. However, the detection quality grows up to 95% only when a great amount of signals (140%) are traced. We attribute this phenomenon to the fact that the missed propagations of targeted paths rely on a large number of signals and it can only be detected when all of those signals are traced. The similar situation happens on circuit DMA, wherein up to 1482 signals affect the detection of targeted paths. The detection quality does not increase and the unmonitored propagation number does not decease significantly even after 200% more signals (297 signals in total) are traced. Further selection is also conducted on circuit des, although its detection quality has exceeded 95% beforehand. As indicated in Fig. 9, 42 out of 46 unmonitored propagations become visible with 10% more signals.

Finally, we evaluate the DfD cost of the reconfigurable data qualification module. Since this module can be utilized to monitor different paths during each debug run, we consider the cost that is big enough to fit the largest detection logic for the most complex path

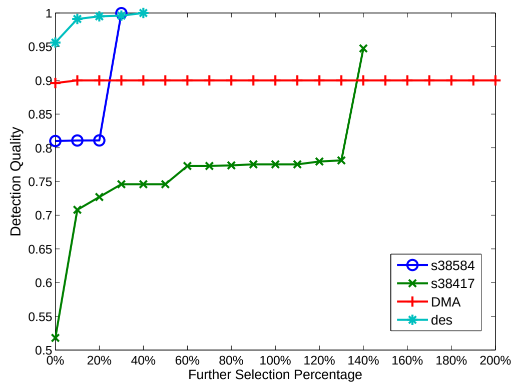

**Figure 8: Detection Quality Evaluation of Improving Quality Selection.**

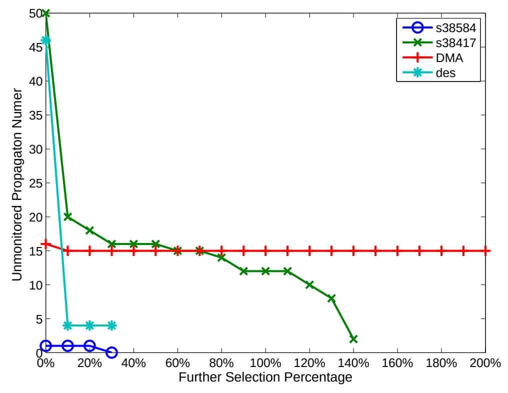

**Figure 9: Unmonitored Propagation Number Evaluation of Improving Quality Selection.**

among all the targeted ones. This module is generated automatically by parsing the circuit to obtain the simplified logic and inserting other sub-modules(e.g., formatter and timestamp generator). It is then synthesized through a commercial FPGA tool to evaluate the hardware cost. Here we choose the traced signal number to guarantee detection quality larger than 95% for circuits s38584 and s38417, while keeping unmonitored propagation number to be 4 for circuit des. As reported in Table 3, for all cases, the maximum DfD cost is 439 4-input LUT, which is acceptable. If multiple paths required to be monitored during one debug run, the cost will not grow dramatically when these paths can share large part of logic elements in simplified logic. More importantly, the DfD cost for this module usually does not increase with the increasing number of targeted paths, because its size is determined by the path(s) with maximum requirement instead of all the targeted paths. Therefore, the relative DfD cost will be lowered when we target more paths in industrial circuits.

| Circuit | Trace Signal # | 4-Input LUT # |
|---|---|---|
| s38584 | 87 | 272 |
| s38417 | 367 | 241 |
| DMA | 108 | 439 |
| usb | 59 | 179 |
| des | 104 | 324 |

**Table 3: DfD cost of data qualification module.**

## 7. CONCLUSION

In this work, we propose a novel trace-based solution for debugging speedpath-related electrical errors, including a new trace signal selection technique that maximizes detection quality and a novel trace qualification module that improves trace buffer utilization. Experimental results on benchmark circuits show that the proposed solution are able to detect a high percentage of speedpath-related electrical errors by tracing a small number of signals with affordable DfD cost.

## 8. ACKNOWLEDGEMENT

This work was supported in part by the General Research Fund CUHK417807 and CUHK418708 from Hong Kong SAR Research Grants Council (RGC), in part by National Science Foundation of China (NSFC) under grant No. 60876029, and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

## 9. 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, 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.

[3] Altera Inc. Design Debugging Using the SignalTap II Embedded Logic Analyzer. http://www.altera.com.

[4] E. Anis and N. Nicolici. On Using Lossless Compresstion of Debug Data in Embedded Logic Analysis. In *Proceedings IEEE International Test Conference (ITC)*, pages 1–10, October 2007.

[5] ARM. Embedded Trace Macrocell Architecture Specification. http://www.arm.com/.

[6] N. Callegari, L. C. Wang, and P. Bastani. Speedpath Analysis Based on Hypothesis Pruning and Ranking. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 346–351, 2009.

[7] A. B. T. Hopkins and K. D. McDonald-Maier. Debug Support for Complex Systems on-Chip: A Review. *IEE Proceedings, Computers and Digital Techniques*, 153(4):197–207, July 2006.

[8] D. Josephson. The Manic Depression of Microprocessor Debug. In *Proceedings IEEE International Test Conference (ITC)*, pages 657–663, 2002.

[9] 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.

[10] 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.

[11] 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.

[12] S. B. Park and S. Mitra. IFRA: Instruction footprint recording and analysis for post-silicon bug localization in processors. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 373–378, 2008.

[13] 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.

[14] S. Tang and Q. Xu. In-band Cross-trigger Event Transmission for Transaction-based Debug. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 414–419, 2008.

[15] B. Vermeulen, T. Waayers, and S. Bakker. IEEE 1149.1-Compliant Access Architecture for Multiple Core Debug on Digital System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 55–63, Baltimore, MD, Oct. 2002.

[16] Xilinx Inc. Chipscope Pro Software and Cores User Guide. http://www.xilinx.com.

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

[18] 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.