# A New Post-Silicon Debug Approach Based on Suspect Window

Jianliang Gao[1,2], Yinhe Han[1], Xiaowei Li[1*]
[1]Key Laboratory of Computer System and Architecture
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China
[2] Graduate University of Chinese Academy of Sciences, Beijing, P.R. China
{gaojianliang, yinhes, lxw}@ict.ac.cn

*Abstract*—**Bugs are tending to be unavoidable in the design of complex integrated circuits. It is imperative to identify the bugs as soon as possible by post-silicon debug. The main challenge for post-silicon debug is the observability of the internal signals. This paper exploits the fact that it is not necessary to observe the error free states. Then we introduce "suspect window" and present a method for determining its boundary. Based on suspect window, we propose a debug approach to achieve high observability by reusing scan chain. Since scan dumps take place only in suspect window, debug time is greatly reduced. Experimental results demonstrate the effectiveness of the proposed approach.**

***Keywords-post-silicon debug; suspect window; trace; scan; bug***

## I. INTRODUCTION

Before an integrated circuit (IC) is manufactured, pre-silicon verification techniques are used to eliminate functional bugs in the circuit. For the increasing system complexity, existing pre-silicon techniques such as simulation and formal verification cannot guarantee that first silicon will be bug free [1]. Moreover, electrical bugs are becoming more serious as the decrease of feature size. Over 50% of the total development cycle of a new product is spent on validating the system's behavior after the first silicon becomes available [2]. Given the increasing cost, it is imperative to identify any bug (including functional bug and electrical bug) that remains in the design as soon as the first silicon is available [3, 4].

Post-silicon debug begins once first silicon is received after the initial design is tapped out. There are two environments in which to debug the actual chip. The chip can be placed in its application environment (on a demonstrator board), which is called in-system debug, or it can be placed on a tester [5]. With various programs running on the system at speed, in-system debug reveals many corner cases which are not encountered on a tester. In-system debug meets more consistent requirement with practical applications. In this paper, we focus on the in-system debug.

*To whom correspondence should be addressed.

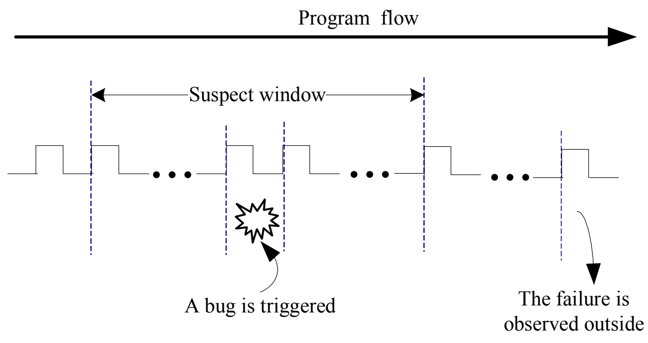


Figure 1. Program execution on the chip with bugs

Special programs are applied to the chip under debug during in-system debug. As shown in Figure 1, a bug in the circuit is triggered in some clock cycle during the program is running. Then the failure caused by the bug may be observed outside by monitoring software or particular device. Unfortunately, many cycles have passed before the failure is observed. In normal functional mode, trace-based techniques can be used to capture some signals in real-time. But the number of trace signals is limited by the on-chip trace resource. So the observability of internal signals is a main gap in the post silicon field [6]. Most of the internal activities cannot be seen because only the chip interface is visible and a small subset of internal states can be routed to the interface [7]. Reusing scan chain is a common technique to improve observability of internal signals in post-silicon debug [5]. Since scan dump takes place serially via special interface such as IEEE 1149.1 test access port (TAP) [8], it is too slow to get the values of flip-flops (FFs) in all clock cycles. So locating the bug (including time and place) remains a challenge in post-silicon debug even if the failure is observed.

To solve the problem of locating bugs fast, we propose a new debug mechanism based on "suspect window". We use suspect window to represent the range of the clock cycles in which the bug is triggered, as shown in Figure 1. We present an algorithm for selecting a set of signals to trace. By tracing the selected signals in real-time, the errors, which are caused by the bugs in the circuit under debug, are collected into trace buffer. Thus the boundary of the suspect window can be obtained by analyzing the trace data. Once the suspect window is determined, the program is re-executed to get all the internal states in the circuit cycle-by-cycle in suspect window. The mechanism achieves high visibility when a bug is triggered, and it is time efficient.

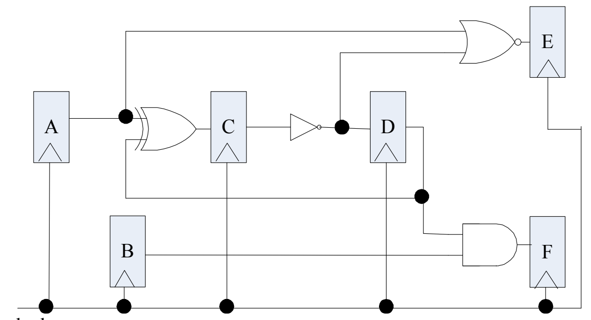
## II. Related Work

Existing post-silicon debug techniques are generally categorized as trace-based and scan-based [3, 9]. Pervious researches have provided various solutions to either of them.

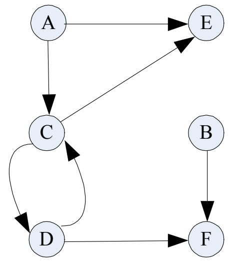
Trace buffer is commonly used to capture data during in-system debug [10]. There are several works to solve the problem of when to trace [11, 12]. Expanding trace buffer observation window for in-system silicon debug is proposed in [11]. They use three debug sessions to find out and save only the data in suspect clock cycles. The suspect clock cycles are determined through a technique using a combination of multiple-input signature register and cycling register signatures, named 2-D compaction technique. So they improve the utilization of trace buffer in depth. The other method to decide the time to trace is to use triggers (event detectors). The debug paradigms in [12] record a group of signals in a trace buffer. The trace operation starts or stops according to the triggers which are programmable. To the problem of which signals to trace, an approach of automating trace signals identification is presented in [9]. They define some principal operations for state restoration. Based on these metrics, an algorithm for identifying a small set of trace signals for state restoration is presented. By tracing these selected signals, a large number of states can be restored. They improve the utilization of the on-chip storage. But it is difficult to restore values for over 2,000 flip-flops using the approach. Another limitation is that the approach is used to process functional bug, and does not take electrical bug into consideration.

Scan chains are reused to improve the visibility of the internal signals in scan-based debug techniques [5]. The standard design for test infrastructure is to multiplex the scan in- and outputs over functional pins. However, when a chip is placed in its application environment for in-system debug, these pins are no longer free for scanning. In post-silicon debug, scan dump data are usually unloaded via serial interface such as JTAG (Joint Test Action Group) interface [8], which is different from manufacture test. In [5], scan chains are connected into a probe by a set of multiplexers. The probe makes the scan cells serially accessible through JTAG interface. They monitor the progress of execution by global control module. The module generates a trigger when a programmable condition occurs. Then scan dump can take place to get the internal states inside the chip. In [13], some scan-based debug methodologies are presented. The debugging related features include TAP, full scan design for all flips-flops, clock controller that enables scan-based control and so on. Scan dump data are compared with that from simulation to implement error isolation. They demonstrate that the error may manifest itself at the pins of device some number of cycles after the actual cycle where the bug is triggered. But they have not solved when to scan dump.

Unlike previous work, we propose a suspect window based post-silicon debug mechanism, which combines the advantages of both scan-based and trace-based techniques.



(a) Circuit-under-debug



|   | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 0 | 1 | 0 | 1 | 0 |
| B | 0 | 0 | 0 | 0 | 0 | 1 |
| C | 0 | 0 | 0 | 1 | 1 | 0 |
| D | 0 | 0 | 1 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 0 |

(b) Abstract graph (c) Neighbor table

Figure 2. Example of abstracting circuit-under-debug for trace signal identification and suspect window size computation

## III. Identify Trace Signals and Determine the Size of Suspect Window

Real-time trace provides a non-intrusive way to observe the signals inside the chip. However, the number of trace signals is limited by the bandwidth of on-chip trace buffer. We propose an algorithm to select the signals which can collect error information as much as possible under the limitation of on-chip trace resource. Furthermore, the procedure to get the size of suspect window for a selected signal is shown in this section.

### A. *Abstract the circuit under debug*

Since the entire chip is surrounded by flip-flops, an error must be recorded in some flip-flops after the bug is triggered. And the error will propagate cycle-by-cycle in the circuit. For example, in Figure 2(a), assuming the error caused by a bug is first recorded in flip-flop A, the error may propagate to flip-flop C and flip-flop E in the next cycle. To fascinate trace signal identification, we use abstract graph to represent the relationship between flip-flops in the circuit under debug. As shown in Figure 2(b), flip-flops are extracted as nodes of the abstract graph. Then a direct edge is added between two nodes if their represented flip-flops are connected through combinational logic. At last, neighbor table is obtained from the abstract graph. It reflects a direct connection relationship between the flip-flops, as shown in Figure 2(c). In the following algorithms, neighbor table is an input parameter as represents the circuit under debug.

## B. *Algorithm for identifying the trace signals*

The trace signals are used to record the errors, which may propagate from other flip-flops. To collect as many errors as possible, the flip-flops that can be infected by more flit-flops should be selected as trace signals. As shown in Figure 3, Algorithm 1 describes the process of selecting the trace signal in detail.

```
Algorithm 1: Identify the trace signals
        Input:   neighbor_table, TraceBuffer_width
        Output: list of selected FFs
 1      Compute Fan_In_Cone for each FF;
 2      Sinked_Set = NULL;
 3      Cur_width = 0;
 4      while Cur_width < TraceBuffer_width
 5          for each FF $i$
 6              Inc_Set($i$) = Fan_In_Cone($i$) -Sinked_Set;
 7          if more than one FF with the largest Inc_Set
 8              Call SizeOfSW for these FFs
 9              Add the FF $k$ with the smallest SizeOfSW
                    to list of selected FFs;
10          else
11              Add the FF $k$ with the largest Inc_Set to list
                    of selected FFs
12          Sinked_Set = Sinked_Set $\cup$ Inc_Set($k$)
13          Cur_width = Cur_width +1;
14      return list of selected FFs
```

Figure 3. Identifying the trace signals

A greedy approach is used to select flip-flops as trace signals. Fan_In_Cone in Line 1 is the set of the FFs that have at least one path to reach the destination FF. It can be computed by nighbor_table of the circuit. Sinked_Set is used to denote the union set of all Fan_In_Cone of the selected FFs. Inc_Set(k) is the set of increased observable FFs if FF $k$ is selected as trace signal. It is defined as:

$$\text{Inc\_Set(k)} = \{a \mid a \in \text{Fan\_In\_Cone(k)} \,\&\, a \notin \text{Sinked\_Set}\}$$

For greedy nature of the algorithm, one FF with the largest Inc_Set is selected as a trace signal in each iteration, as shown in Line 4-13. For example, flip-flop F will be selected in the first iteration for the circuit in Figure 2. Note that if more than one node has the largest Inc_Set size, as happens often in large circuits, we call Algorithm 2 (SizeOfSW) to calculate the size of suspect window for these candidate flip-flops. Then the flip-flop with the smallest distance is selected as trace signal as shown in Line 8-9 in Figure 3. It will lead to fewer single step operations and therefore save debug time. When a FF is added to the list of selected FFs in an iteration, Sinked_Set and Cur_width are subsequently updated.

## C. *Algorithm for determining the size of suspect window*

The bug might be triggered many cycles before the error is collected by trace signals. The size of suspect window represents the largest number of clock cycles which the error may have passed. Assuming no error mask happens when the failure has been observed outside, the largest number of clock cycles is the largest one of all the shortest paths to the trace signal.

Algorithm 2 describes the method of determining the size of suspect window for a trace signal. Line 1-5 is the initialization process. $n$ is the number of all flit-flops. Array mark presents whether the shortest path has been obtained for the flip-flop. Array dist denotes the current distance from a flip-flop to the destination flit-flop (denoted as sink in Figure 4). The process of computing the distances is described at Line 6-19. The shortest path from one flip-flop to sink is calculated and the other paths are updated in each iteration. At last, the maximum length in Array dist is returned as the size of suspect window for sink.

```
Algorithm 2: SizeOfSW, Get the suspect window size
        Input:   neighbor_table, sink
        Output: the size of suspect window for sink
 1      Set all 0 in neighbor_table as $+\infty$;
 2      for (i=1;i<=$n$;i++) {
 3          mark[i]=0;
 4          dist[i]=neighbor_table[i][sink];
        }
 5      mark[sink]=1;
 6      do{
 7          min=$+\infty$;
 8          k=0;
 9          for i=1;i<$n$;i++)
10              if ((mark[i]==0) && (dist[i]<min)){
11                  min=dist[i];
12                  k=i;
                }
13          if (k) {
14              mark[k]=1;
15              for (i=1;i<n;i++)
16                  if (disti)>min+neighbor_table[i][k])){
17                      dist[i]=min+neighbor_table[i][k])
                    }
19      }while(k)
20      return maximum value in dist
```

Figure 4. Getting the size of suspect window

Algorithm 2 is of order $O(n^2)$, where $n$ is the total number of flit-flops. Since procedure is applied to every selected signal, the final order for computing suspect window size is $O(n^2m)$ where $m$ is the number of trace signals. Furthermore, the sizes are computed only once before the circuit is fabricated. So the time complexity is acceptable.

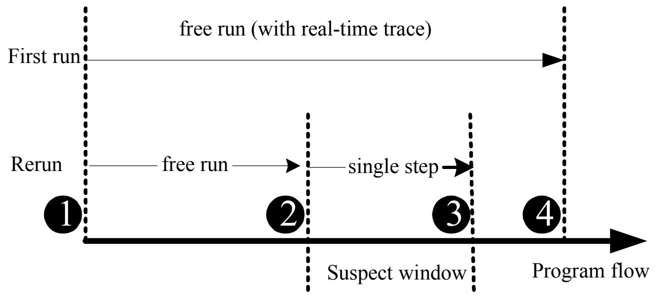


Figure 5. The proposed debug mechanism

## IV. PROPOSED DEBUG MECHANISM

Program is executed twice in the proposed debug mechanism, as shown in Figure 5. The first run is to execute the program without stop. It runs until a failure is observed outside (④ in Figure 5). The rerun process is divided into two partitions. In the former partition (①–②), program runs free in normal mode. Then single step takes place during suspect window (②–③).

### *A. Free run with real-time trace*

The objective of the first run is to find out the boundary of suspect window. Using the algorithm in Figure 3, the trace signals can be selected according to the bandwidth of trace buffer when design is finished. The states of the selected signals are traced into trace buffer non-intrusively during the system runs in normal functional mode. There are two techniques to deal with trace data. One is to interrupt the normal execution and unload trace data to off-chip storage via some interface such as JTAG [12]. The other technique is to adopt high-speed trace ports. Trace data are unloaded while maintaining the real-time execution [14]. Once trace data are available, it can be compared with simulation data. If they do not match, it means that the errors caused by bug propagate to the selected flip-flop. The first cycle in which the mismatch happens is the end boundary of suspect window (denoted as ③ in Figure 5).

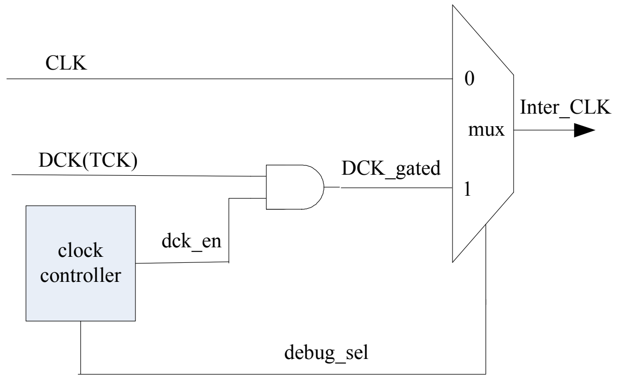
The next step is to get the start point of suspect window. $\gamma$ denotes the number of the clock cycles between the start point ① to the first mismatch point ③. Denoting the size of suspect window of the mismatched flip-flop as $\delta$, the start clock cycle of suspect window can be presented as

$$\eta = \gamma - \delta$$

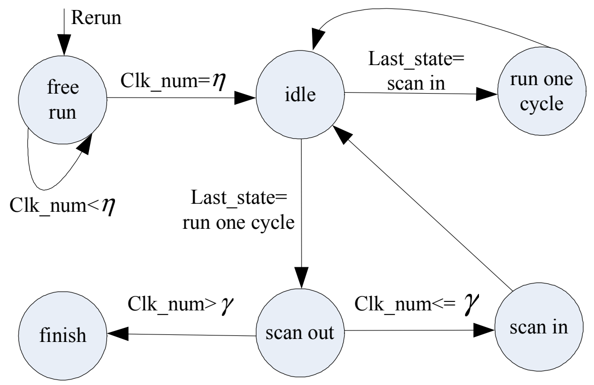
where $\eta$ is the start clock cycle denoted as ② in Figure 5. $\delta$ has been known ahead by Algorithm 2. Thus the start boundary of the suspect window is also obtained.

### *B. Rerun with single step*

The aim of rerun is to get all internal states in the circuit under debug during suspect window. Scan dumps provide direct observability of the scanable flip-flops in the circuit. It is possible to capture the snapshots of the chip in a system by use of scan dump. However, scan dump is a time-consuming operation. It is not feasible to scan all states in the whole process of the program execution. But once bug has been arrowed to a small range of clock cycles, it is possible to stop the functional clock in a specified clock cycle and scan out the contents of all the flip-flops for very fine-grained visibility to analyze the root cause of the failure.



(a) clock generation block diagram



(b) state transition of clock controller

Figure 6. Clock control and state transition. Suspect window is [$\eta, \gamma$]. Clk_num is the number of the clock cycles which program has passed.

Clock generation is a key component in the overall silicon debug methodology. The debug process can be automated by controlling the clock system in the chip. We design a clock control unit for the proposed debug mechanism as shown in Figure 6(a). As test clock (TCK) is not used in debug mode, it can be reused as debug clock (DCK). The internal clock (inter_CLK), which drives the flip-flops, is generated from functional clock (CLK) or DCK via a multiplexer. The inter_CLK is connected with CLK during free run. In single step mode, program execution allows to examine/modify internal states using scan chain. Normal operation can be resumed after the former states are recovered. Clock signal inter_CLK is alternately driven by CLK and DCK in single step mode. When debug_sel and dlk_en are asserted, inter_CLK is driven by DCK. When debug_sel becomes zero, inter_CLK is driven by CLK. These control signals are decided by "clock controller" in Figure 6(a).

Figure 6(b) demonstrates the state transition of clock controller during the rerun. The state transitions between "free run", "idle", "scan out", "scan in", "run one cycle" and "finish". Note that the transition between two clocks may introduce glitch, such as from "run one cycle" to "scan out" and from "scan in" to "run one cycle", so we use "idle" state to avoid glitch. State "idle" means that inter_CLK is disabled by asserting debug_sel as "1" and clk_en as "0". Inter_clk is connected with DCK in "scan out" and "scan in". In "scan out" state, the values of flip-flops are shifted out for full visibility analysis. The circuit is controlled by functional clock (CLK) in "run one cycle". When functional clock steps one cycle, the internal states in the chip are changed to the states of next cycle. These values are scanned out in "scan out" state. As scan dump changes the values of flip-flops, restoring the former data is necessary to resume the execution. In our design, the restore operation is carried out in "scan in" state.

The program runs free during the range from ① to ②. When the program execution reaches ②, it runs in single step mode. In this mode, "run one cycle", "scan out" and "scan in" are alternately executed. Rerun process can be automated by controlling clock system. The parameters for the automated process are the boundary of suspect window (② and ③).

When the internal states of the chip are scanned out in suspect window, the root cause of failure can be obtained by comparing the scan date with simulation values or analysis them according to design specification.

TABLE I. RESULTS OF BUG COVERAGE AND THE SIZE OF SUSPECT WINDOW

| Circuit | Bandwidth (bits) | Random | | Proposed | |
|---|---|---|---|---|---|
| | | Bug coverage (%) | Average size of suspect window | Bug coverage (%) | Average size of suspect window |
| s13207 | 8 | 71.61 | 17.00 | 89.03 | 16.68 |
| | 16 | 74.27 | 17.02 | 94.83 | 17.79 |
| | 32 | 77.95 | 17.27 | 98.12 | 17.39 |
| s15850 | 8 | 88.01 | 8.75 | 92.80 | 7.85 |
| | 16 | 88.30 | 8.85 | 94.47 | 7.70 |
| | 32 | 87.17 | 8.68 | 97.15 | 7.30 |
| s9234 | 8 | 74.95 | 6.46 | 100 | 6.75 |
| | 16 | 83.04 | 6.40 | 100 | 6.75 |
| | 32 | 91.80 | 6.45 | 100 | 6.75 |

## V. EXPERIMENTAL RESULTS

To evaluate the proposed approach, we have conducted experiments on three ISCAS'89 benchmark circuits, namely s13207, s15850 and s9234 [15]. Bug (embodied as error) coverage and debug time are two important metrics to measure the effectiveness of post-silicon debug mechanism. We will analyze them in this section.

First, we compare bug coverage between the method of selecting trace signals randomly and the proposed method (Algorithm 1). For random method, we repeat the process of selecting the required number of trace signals 100 times. Then the results of these times are averaged and listed in Table I. Bug coverage is obtained by

$$bug\_coverage = sizeof(Sinked\_Set) / Total\_FF\_Number$$

where $sizeof(Sinked\_Set)$ is the number of the flip-flops which have at least one path to the trace signals. $Total\_FF\_Number$ is the number of all flip-flops in the circuit under debug. The proposed algorithm for signal identification achieves higher bug coverage than that of random method. For example, only eight trace signals in s9234 can achieve 100% bug coverage using the proposed method. But it reaches 91.8% even though 32 flip-flops are traced by random method. Another advantage is that the proposed algorithm is more stable than random method. The bug coverage increases with the increase of the bandwidth of trace buffer. But it does not always increase for random method. For example, bug coverage of s15850 increases very little or even not as the increase of the bandwidth.

The trace signal with larger fan-in cone is easier to be infected by the errors in the circuit. So the sizes of suspect window can be obtained by weighted average of the sizes of selected signals according to the number of sinked flip-flops.

$$Avg\_SizeOfSW = \frac{\sum_{for\ each\ sink\ k} Sinked(k) \times SizeOfSW(k)}{\sum_{for\ each\ sink\ k} Sinked(k)}$$

where $sinked(k)$ is the number of the flip-flops which are sinked by the trace signal $k$. $SizeOfSW(k)$ returns the suspect window size of trace signal $k$. The average sizes of suspect window obtained by random method and the proposed method are listed in the fourth and sixth column of Table I respectively. The sizes are small opposed to the scales of the circuits. As has been described, scan dumps only take place during suspect window. So the benefit of smaller size of suspect window will be embodied in saving debug time.

As has been described in Section IV-A, trace data can be unloaded via high-speed trace ports while maintaining the real-time execution. So we compare only the debug time between scan-based technique and the proposed technique in the experiments. Their ratio $r$ is:

$$r = \frac{\left(N_s \times T_d \times 2 + T_f\right) \times C_f}{C_f \times T_f \times 2 + N_s \times S_w \times T_d \times 2}$$

where

$C_f$ : Cycle Number of program execution;

$S_w$ : Size of suspect window;

$T_f$ : Functional clock period;

$T_d$ : Debug clock period;

$N_s$ : Number of scanable flip-flops;

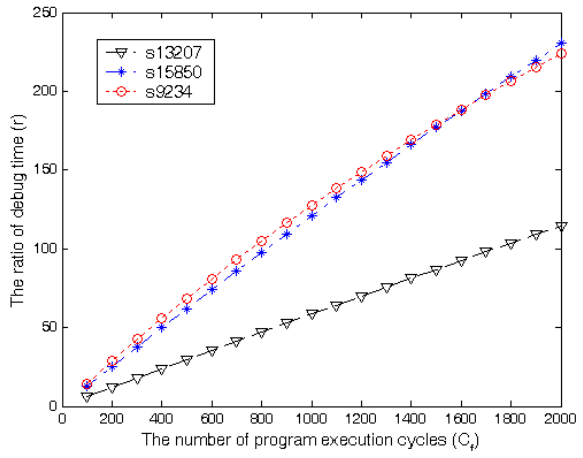


Figure 7. The comparison of debug time of scan-based technique and the proposed technique

The results of the ratio ($r$) are shown in Figure 7. The results are obtained assuming that the bandwidth is 8 and functional clock is four times faster than debug clock. It can be seen that the ratios increase as the increase of the number of program execution cycles. For example, the ratio is 127.52 when $C_f$ =1000 for s9234, while it increases to 223.76 when $C_f$ =2000. The ratio is also related to the size of suspect window. The average size of suspect window of s13207 is larger than that of s9234 and s15850, so the ratio for s13207 is the smallest. It reflects that scan dump is time-consuming.

## VI. Conclusions and Future Work

In this paper, we have proposed a suspect window based post-silicon debug mechanism. First, we have presented the algorithm for signal identification and the method for determining the suspect window. Then the detailed debug approach has been described, including the design of clock generation unit. Experimental results show that the proposed approach achieves great benefit in debug time and bug coverage.

Charactering the errors quantitatively including mask during propagation is our ongoing work. Furthermore, we plan to extend the suspect window based approach to multiple clock domain circuits.

## Acknowledgment

This work was supported in part by the High-Tech Research and Development Program of China under grant No. (2007AA01Z109, 2007AA01Z113, 2008AA01Z147) and National Natural Science Foundation of China under grant No. (60806014, 60633060, 60831160526, 60776031), and by the National Basic Research Program of China under grant No. (2005CB321604, 2005CB321605).

## References

[1] K. Constantinides, O. Mutlu, and T. Austin, "Online design bug detection: RTL analysis, flexible mechanisms, and evaluation," in *IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 282-293.

[2] J. Geuzebroek and B. Vermeulen, "Integration of Hardware Assertions in Systems-on-Chip," in *Proceedings IEEE International Test* Conference *(ITC)*, 2008, pp. 1-10.

[3] B. Vermeulen and S. K. Goel, "Design for debug: catching design errors in digital chips," *IEEE Design & Test of Computers,* vol. 19, pp. 35-43, 2002.

[4] L. Winemberg, D. Carder, M. Adadir, R. Aitken, R. Antley, and J. Carulli, "Debug War Stories," in *IEEE International Test Conference (ITC)*, 2008, pp. 1-1.

[5] G. J. Van Rootselaar and B. Vermeulen, "Silicon debug: scan chains alone are not enough," in Proceedings IEEE International Test Conference (ITC), 1999, pp. 892-902.

[6] H. Yu-Chin, "Maximizing Full-Chip Simulation Signal Visibility for Efficient Debug," in Proceedings International Symposium on VLSI Design, Automation and Test, 2007, pp. 1-5.

[7] T. Bojan, M. A. Arreola, E. Shlomo, and T. Shachar, "Functional coverage measurements and results in post-Silicon validation of Core™ 2 duo family," in IEEE International High Level Design Validation and Test Workshop, 2007, pp. 145-150.

[8] B. Vermeulen, R. Kuhnis, J. Rearick, N. Stollon, and G. Swoboda, "Overview of Debug Standardization Activities," Design & Test of Computers, IEEE, vol. 25, pp. 258-267, 2008.

[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 Conference (DATE), 2008, pp. 1298-1303.

[10] E. Anis and N. Nicolici, "On using lossless compression of debug data in embedded logic analysis," in Proceedings IEEE International Test Conference (ITC), 2007, pp. 1-10.

[11] Y. Joon-Sung and N. A. Touba, "Expanding Trace Buffer Observation Window for In-System Silicon Debug through Selective Capture," in Proceedings IEEE VLSI Test Symposium (VTS), 2008, pp. 345-351.

[12] M. Abramovici, P. Bradley, K. Dwarakanath, P. A. L. P. Levin, G. A. M. G. Memmi, and D. A. M. D. Miller, "A reconfigurable design-for-debug infrastructure for SoCs," in Proceedings ACM/IEEE Design Automation Conference (DAC), 2006, pp. 7-12.

[13] K. Holdbrook, S. Joshi, S. Mitra, J. Petolino, R. Raman, and M. Wong, "MicroSPARCTM: a case-study of scan based debug," in Proceedings IEEE International Test Conference (ITC), 1994, pp. 70-75.

[14] H. F. Ko, A. B. Kinsman, and N. Nicolici, "Distributed Embedded Logic Analysis for Post-Silicon Validation of SOCs," in *Proceedings IEEE International Test Conference (ITC)*, 2008, pp. 1-10.

[15] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proceedings IEEE International Symposium on Circuits and Systems (ISCAS)*, 1989, pp. 1929-1934.