PAPER

A Novel Post-Silicon Debug Mechanism Based on Suspect Window*

Jianliang GAO†, Yinhe HAN†, *Members*, *and* Xiaowei LI†, *Nonmember*

SUMMARY Bugs are becoming unavoidable in complex integrated circuit design. It is imperative to identify the bugs as soon as possible through post-silicon debug. For post-silicon debug, observability is one of the biggest challenges. Scan-based debug mechanism provides high observability by reusing scan chains. However, it is not feasible to scan dump cycle-by-cycle during program execution due to the excessive time required. In fact, it is not necessary to scan out the error-free states. In this paper, we introduce Suspect Window to cover the clock cycle in which the bug is triggered. Then, we present an efficient approach to determine the suspect window. Based on Suspect Window, we propose a novel debug mechanism to locate the bug both temporally and spatially. Since scan dumps are only taken in the suspect window with the proposed mechanism, the time required for locating the bug is greatly reduced. The approaches are evaluated using ISCAS'89 and ITC'99 benchmark circuits. The experimental results show that the proposed mechanism can significantly reduce the overall debug time compared to scan-based debug mechanism while keeping high observability.

key words: *debug, scan dump, real-time trace, suspect window*

1. Introduction

Before an integrated circuit (IC) is manufactured, pre-silicon verification techniques are used to eliminate functional bugs in the circuit. Due to the increasing system complexity, existing pre-silicon techniques such as simulation, emulation and formal verification cannot guarantee that first silicon will be bug free [1]. Moreover, electrical bugs are becoming more serious with the decreasing feature size. Whereas, pre-silicon methods don't address many deep-submicron electrical bugs that occur in the actual device [2]. Since operation needs to be checked over the entire voltage, temperature, and frequency expected for the design, post-silicon debug has become the most time-consuming part - 35% on average - of the development cycle of a new chip [3]. Considering the increasing cost, it is imperative to identify the bugs that remain in the chip as soon as the first silicon is available [4].

Post-silicon debug (simplified as silicon debug in the following) needs to identify both functional bugs and electrical bugs (e.g. [5], [6]). For all bugs, their symptoms are manifested as errors when the bugs are triggered. The errors are recorded and propagated in sequential logic. If the errors cause malfunction of the system, it is said that a failure happens. Typical flow of debugging the failure covers three steps: activate the bugs and detect the failure; isolate and identify the root cause of the failure; confirm and fix the failure in some situations [7], [8]. Since the bugs are sensitive to environmental parameters and the fault-masking effect is serious, various test and/or application programs are run on the chips in the first step. Once the failure is observed outside the chip, the second step begins to locate the root cause of the failure. This step involves determining when (temporally) and where (spatially) the bug is triggered. After the bug has been located, the final step is to use physical tools to confirm and fix it or return the information to designers for recovering the bugs. The objective of our work is to locate the bug temporally and spatially when the failure is observed outside the chip, i.e, belongs to the second step.

There are two kinds of environments to locate the bug in a real 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 [9]. With various programs (applications) running on the system to trigger the potential bugs, in-system debug reveals many marginal cases which are not encountered on a tester. In this paper, we focus on the in-system debug.

The observability of internal signals is a main gap for in-system debug [10]. As shown in Fig. 1, a bug is triggered at some clock cycle during the program execution. The failure caused by the bug may be observed by monitoring software or particular device such as logic analyzer. Unfortunately, many cycles have elapsed before the failure is observed outside [11]. Trace-based debug technique can capture some signals in real-time [12], but the number of trace signals is limited by the on-chip trace resource. Reusing scan chain is another technique to improve the observability of internal signals [13]. Scan dump can capture a snapshot of the chip by freezing the program execution and shifting out the contents. Since scan dump takes place serially via special interface such as IEEE 1149.1 (JTAG) test access port (TAP) [14] during in-system debug, it is too slow to get the states of flip-flops (FFs) in all clock cycles.

It remains a challenge to pinpoint both temporally (when) and spatially (where) a failure is occurring, even if the failure is observed outside. To solve the problem, we propose a new debug mechanism based on Suspect Window. The main contributions in this work, detailed in Sect. 3 and 4, are as follows:

Manuscript received July 24, 2009.

Manuscript revised December 1, 2009.

†The authors are with the Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, and with the Graduate University of Chinese Academy of Sciences, Beijing, China.

*A preliminary version of this paper was published in Proc. 27th IEEE VLSI Test Symposium (VTS), Santa Cruz, USA, May 3-7, 2009, pp. 85-90.

DOI: 10.1587/transinf.E93.D.1175

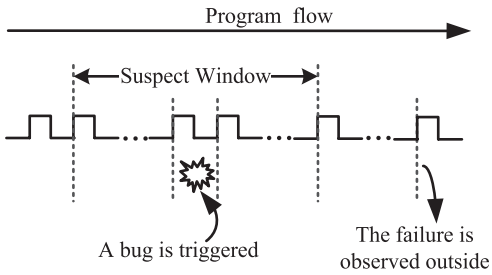


Fig. 1 Bug and failure.

- An efficient approach is proposed to determine Suspect Window quantitatively. We introduce Suspect Window to represent the range of the clock cycles in which the bug is triggered. To determine the suspect window quantificationally, we present an algorithm to select the trace signals, which can ensure the observability for errors in the circuits.
- A novel debug mechanism based on Suspect Window is proposed, which can quickly locate the bug both temporally and spatially. In this debug mechanism, scan dumps are only taken in the suspect window, and therefore the time for locating the bug is reduced greatly.

The remainder of this paper is organized as follows. Section 2 reviews related work in silicon debug. Section 3 presents the approach of determining the suspect window. Section 4 describes the debug mechanism based on Suspect Window. Experimental results and conclusions are given in Sect. 5 and 6, respectively.

2. Related Work and Motivation

Existing post-silicon debug techniques are generally categorized as scan-based and trace-based mechanisms [15]. For either of the categories, a number of technologies have been explored in recent years.

2.1 Scan-Based Debug Mechanism

Scan chains are commonly employed in manufacturing test as a Design-for-Test (DFT) technique [16], [25]. This hardware can be reused to improve the visibility of the internal signals during silicon debug. The standard scan chains are connected to tester via functional pins in test mode. However, when a chip is placed in its application environment for in-system debug, these pins are no longer free for scanning. In [13], a wrapper design for the scan chains containing a set of multiplexers is proposed, which connects all scan chains into a single chain. The chain makes the scannable sequential elements accessible through JTAG interface. Since the contents of the flip-flops (FFs) are shifted out via Test-Data-Output (TDO) serially, scan dump operation is time-consuming.

Different from on-tester debug, the chip need to be switched between normal functional mode and scan mode during in-system debug. A set of operations need to be performed in program execution. When the chip is stopped, its state can be read out by scan dump operation before continuing operation. Hardware breakpoint is proposed to determine when to stop the program execution [17]. The breakpoint typically contains two components: comparators, to monitor important application specific signals such as program counters and internal instruction or data buses; and counters, to allow more complex breakpoint sequences. However, most post-silicon bugs are not predictable in design phase [4]. The monitoring signals and triggering conditions are difficult, if not impossible, to know in pre-silicon design phase for the non-determination of electrical bugs. Caty *et al.* propose a failure propagation tracing method to locate the failed path [18]. When a failure is observed, scan dumps are taken to provide data for pass/fail divergence analysis. Whereas, the clock cycle range in which scan dumps are taken is uncertain during in-system debug. Scam dumps cost much time to scan out error-free data.

2.2 Trace-Based Debug Mechanism

As a complementary to scan, trace-based mechanism is used to capture data in real-time during silicon debug [19]. The captured data are transferred outside via trace port directly or stored in an embedded trace buffers. Limited by the on-chip resource, the size of trace buffers is usually small [20]. The trace ports, which provide real-time data transfer without needing on-chip memories, raise the issue of committing additional pins for debug. So the bandwidth is also limited. In general, the number of trace signals is extremely small relative to the number of the total signals in the chip.

Some proposals have focused on extending the trace data in both combinational logic and sequential logic (e.g. [15], [21]). A trace signal selection method that tries to maximize the number of restored states is firstly proposed in [15]. State restoration is taken place off-line using the captured data. The authors define the forward restorability and backward restorability for logic gates. Since forward and backward operations require that there exist no electrical errors, these methods can deal with only functional bugs, but not electrical bugs.

To reduce the additional area for large trace buffers, which are used only for the purpose of silicon debug, trace compression techniques are proposed to compress debug data on-chip before being stored into trace buffers [22]. Although trace compression can increase the number of samples for each trace signal, the amount of signals that can be monitored is still limited. Thus, it is desirable to find an alterative objective for trace signal selection, instead of observing the errors of the trace signals alone.

2.3 Motivation

Two conclusions can be drawn from the related work:

- Scan-based debug mechanism can achieve high observability of internal signals, but it is too time-consuming

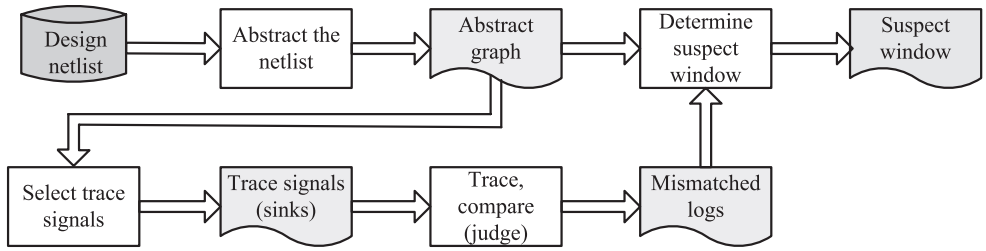


Fig. 2 Flow of determining suspect window.

to scan dump cycle-by-cycle during the program execution.

- Trace-based debug mechanism can achieve real-time visibility of the trace signals. However, the number of trace signals is limited by on-chip trace resource.

The difficulties in debug process motivate the work presented in this paper. Unlike previous methods, we use trace signals to collect the errors in the circuit. By analyzing the trace data and circuit topology, we can draw a quantitative range of clock cycle, named Suspect Window, to cover the time of bug triggered. Then, scan dumps are taken only in the suspect window. The new debug mechanism can quickly locate the bug when a failure is observed.

3. The Approach of Determining Suspect Window

3.1 Overview the Approach

Figure 2 depicts the flow of determining Suspect Window. It covers the following steps:

(i) Abstract the circuit under debug (design netlist) as abstract graph. Abstract graph describes the dependence relations between the sequential elements.

(ii) Trace signals are selected according to abstract graph. The selected signals will be used as collect the errors in the circuits.

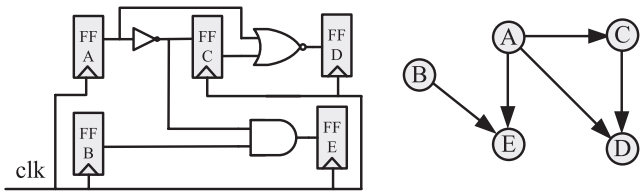
(iii) Trace the selected signals during program execution. Then, compare the trace data to simulation data and obtain mismatched logs (including mismatched clock cycle and mismatched trace signal).

(iv) Determine Suspect Window according to abstract graph and the mismatched logs.

The first two steps are taken before tape-out. The details are presented in Sect. 3.2 and 3.3, respectively. The selected signals are traced in real-time when programs run on the chip. Trace data are then compared with simulation data. The mismatched logs is used to determine Suspect Window, which is detailed in Sect. 3.4.

3.2 Abstracting the Circuit

Since the entire chip is surrounded by FFs, the errors can always be recorded in some FFs [23]. Then the errors are propagated between the FFs cycle-by-cycle until they are observed outside. We use abstract graph to represent the relationship between the FFs in the circuit.



(a) Circuit-under-debug (FF: flip-flop) (b) Abstract graph

Fig. 3 An example of abstract graph.

Define 1: *Abstract graph* $G(V, E)$ is a directed graph, with each vertex $v_i \in V$ representing a FF and the directed edge (v_i, v_j), denoted as $e_{i,j} \in E$, representing that FF_i are connected to FF_j via combinational logic.

Figure 3 shows an example that demonstrates how the abstract graph is obtained from a design netlist. In this example, five FFs are turned into five nodes in the abstract graph. If two FFs are connected via combinational logic, the corresponding nodes in the abstract graph are connected with a directed edge. The direction of the edge represents that the state of the source node may influence the state of the destination node in next clock cycle. For example, in Fig. 3, since the current state of FFA may influence the states of FFC, FFD, and FFE after one clock cycle, there is a directed edge from A to C, D, and E, respectively.

3.3 Trace Signal Identification

We use trace signals to collect the errors in the circuit under debug (CUD). For a trace signal in the circuit, its corresponding node in the abstract graph is named a sink. Denote the set of all sinks as S , i.e.,

$S = \{i \mid i \in V \text{ and } i \text{ is a sink}\}$

Fan-in cone of a sink i is the set of the nodes which have at least one path to sink i, denoted as $cone(i)$. The set of the nodes, which belong to any fan-in cone of the sinks, are named $monitored\_FF$, i.e.,

$monitored\_FF = \{j \mid j \in V \text{ and } j \in (\bigcup_{i \in S} cone(i))\}$

Define 2: *FF coverage* is the ratio of the number of the FFs which are fanned in by the sinks ($monitored\_FF$) to the total number of FFs in the CUD. It is computed as:

Algorithm 1: Identify trace signals

```
      Input: BW, TH, Abstract graph G(V, E)
      Output: The set of sinks S
1     Initiate S = Ø; cur_cov = 0; monitored_FF = Ø;
2     for each i ∈ V, calculate cone(i);
3     while (#S < BW) do
4        if cur_cov < TH then
5           for each node i, i ∈ (V\S) do
6              Inc_set(i) = cone(i)\monitored_FF
7           end for
8           k = argmax_i(#Inc_set(i))
9        else
10          for each node i, i ∈ (V\S) do
11             calculate PF assuming i is a sink
12          end for
13          k = argmin_i(PF(i))
14       end if
15       S = S ∪ {k};
16       monitored_FF = monitored_FF ∪ cone(k);
17       cur_cov = #monitored_FF/#V
18    end while
19    return S
```

$$FF\ coverage = \frac{\#monitered\_FF}{\#V} \tag{1}$$

where # means to return the number of the elements in the set and V is the set of the nodes in abstract graph. FF coverage reflects the ability of the sinks to collect the errors in the CUD. To collect more errors, the nodes which can fan in the most FFs are the best candidates as trace signals. For example, in Fig. 3, FFC can collect errors that come from FFA and FFC; FFD can collect errors that come from FFA, FFC, and FFD. It means $cone(D) > cone(C)$. So FFD is better than FFC as a trace signal. If some sinks have been selected, we choose the node which can increase the most FF coverage as a new sink. When all or the most FFs have been fanned-in by the selected trace signals, adding more trace signals has little or no increase in FF coverage. For this situation, we introduce Propagation-Factor (PF) as a metric for selecting trace signals:

$$PF = \sum_{i \in V} hop_i \tag{2}$$

where hop_i is the number of the hops from node i to its nearest sink in abstract graph. It represents the speed of the errors being caught by the sinks. The smaller PF is, the faster the sink can record the errors in the CUD.

Algorithm 1 details the process of trace signals identification. The input parameters BW and TH represent trace bandwidth and the required minimum FF coverage, respectively. The output of the algorithm is the set of selected trace signals. The algorithm is divided into two phases. In the first phase, the objective is to achieve the highest FF coverage. For each non-sink node i, ($i.e.,\ i \in (V\backslash S)$), the corresponding increased set of monitored nodes is calculated (Line 6). The node with the largest increased set is then selected as a sink (Line 8). In the second phase, the metric PF is used to guide the signal selection. For each non-sink node, compute the PF assuming that it is a sink (Line 11). Then, the node with the least PF is selected as a sink in this loop (Lines 13). As can be seen, if TH is set to 1, the algorithm will select the sinks toward the object of 100% FF coverage.

Algorithm 2: Get suspect window size for one sink

```
      Input: Sink, Abstract graph G(V, E)
      Output: suspect window size SW_Size
1     Initiate Q = Ø;
2     for each node i ∈ V, set Hop(i) = 1;
3     TAIL(Sink,Q)
4     while not EMPTY(Q) do
5        i = HEAD(Q)
6        SW_Size = Hop(i);
7        for all node j, j ∈ V and (i, j) ∈ E
8           if not EXIST(j, Q) then
9              TAIL(j, Q);
10             Hop(j) = Hop(i)+1;
11          end if
12       end for
13    end while
14    return SW_Size
```

3.4 Suspect Window Determining

When the errors are collected by sinks during real-time trace, we need to find out the root cause for the errors. Suspect Window represents the range of the clock cycles in which the bug is triggered. If the suspect window is too small, the root cause may escape examination, but if it is too broad, the operation time can be considerable. In the following, we present the approach of determining Suspect Window according to the number of mismatched sinks.

First, we consider the situation of one mismatched sink. The bug, which causes the error, must be triggered before the first mismatched clock cycle. We let the first mismatched clock cycle be the end point of the suspect window (e_T). The next objective is to specify the start point of the suspect window (s_T). Denote the suspect window size as δ, the start point s_T can be calculated as:

$$s\_T = e\_T - \delta \tag{3}$$

Thus, the objective is transformed to obtain the suspect window size. We present an efficient algorithm to get the suspect window size for one sink. In Algorithm 2, we use a queue (Q) to store the to-be-processed nodes. HEAD, TAIL, and EMPTY are three operations on queue. They represent to get the first element form the head, to add an element to the tail, and to decide whether the queue is empty or not, respectively. For each processed node, its next hop neighboring nodes are put into the queue if they are not in the queue already (Lines 8-11). Finally, the number of hops corresponding to the last processed node in the queue is returned as suspect window size.

Second, we consider the situation that more than one sink collects the error caused by the bug. Using the preceding approach for one sink, the suspect window of each mismatched sink can be obtained. The bug must be triggered after all start points and before all end points of these

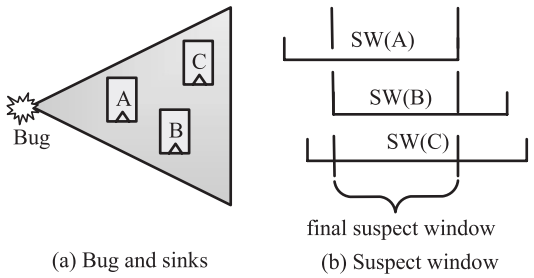


Fig. 4 Suspect window of multiple mismatched sinks.

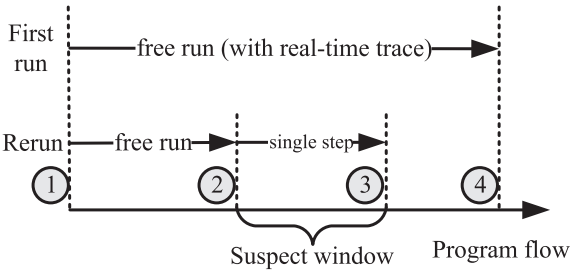


Fig. 5 Debug mechanism based on suspect window.

suspect windows. Thus we can take the latest start point as the final start point and the earliest end point as the final end point. For example, in Fig. 4, the error caused by the bug is collected by FFA, B and C. Three suspect windows (SW(A), SW(B), and SW(C)) are obtained using the preceding method. The intersection of SW(A),SW(B) and SW(C) is taken as the final suspect window. It can be seen that the final suspect window is smaller than any of the separate suspect windows.

For multiple clock domain SoC, the suspect window can be determined using "divide-and-conquer" method. For each clock domain, the trace signals are selected according to their abstract graphs respectively. The selected trace signals are monitored in real-time. Note that trace memory can be shared by the trace signals of different clock domains [29]. Once the mismatch occurs, debug engineer focuses on the corresponding clock domain and the preceding method which is applied to a single clock domain can be used to determine the suspect window for the mismatched clock domain.

4. The Proposed Debug Mechanism

Various programs (e.g., real applications or/and test programs) run on the chip to trigger the potential bugs that remain in chip. Once a failure is observed outside, the debug process of locating the root cause of the failure begins. As shown in Fig. 5, the proposed debug mechanism requires that the program executes twice. The first execution is free run from the start point ① to the point ④ at which the failure is observed. The objective of the first run is to determine the suspect window (②-③). Scan dumps are taken in the suspect window during rerunning the program. In the following subsections, we detail the debug mechanism.

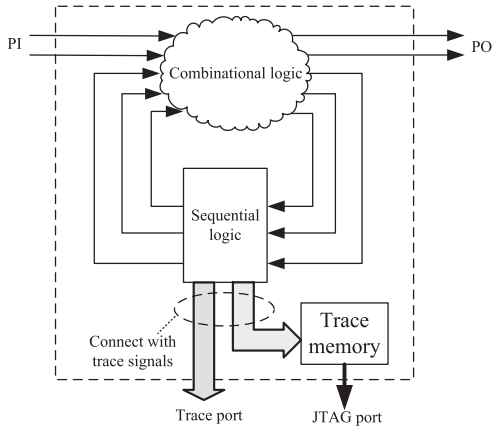


Fig. 6 Trace sequential logic in real-time.

4.1 Free Run with Real-Time Trace

In normal functional mode, programs are input to the chip via primary input (PI). The results observed on primary output (PO) can be used to judge whether the failure occurs or not. Since the sequential logic is not controllable via scan chains in normal functional mode, the trace signal is the only part of observable internal states. As shown in Fig. 6, the selected signals (belonging to sequential logic) are traced in real-time. The contents of trace signals can be obtained by two approaches. One is to transfer them via high-speed trace port. Trace port provides the capability of real-time data transfer. The other approach is to store trace data into on-chip trace memory. The data in trace memory can be exported outside via JTAG port. A design for debug can provides either or both of the approaches. When the trace data are obtained, they are compared with golden simulation results. The mismatched sinks and clock cycles can be obtained by this way. Then, the suspect window can be gotten by use of the method described in Sect. 3. Once the suspect window is known, the debug mechanism can take the rerun process.

4.2 Rerun with Scan Dump

The rerun process is divided into two parts. In the former part (①-②), program runs non-intrusively. In the second part, single step execution and scan dump are taken in the suspect window (②-③). The rerun process can be automated by controlling the clock system. Figure 7 shows the transfer among the operations. It includes the following operations:

- Free run
  The internal clock is driven by system clock, and the program run freely until the functional clock cycle number reaches the begin point of the suspect window ($clk\_num < \eta$).
- Stop
  When the clocks are stopped, the contents in FFs are

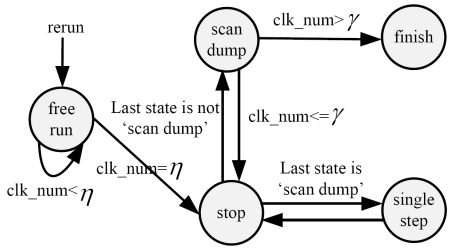


Fig. 7 Operations during rerun process (suspect window: $[\eta, \gamma]$).

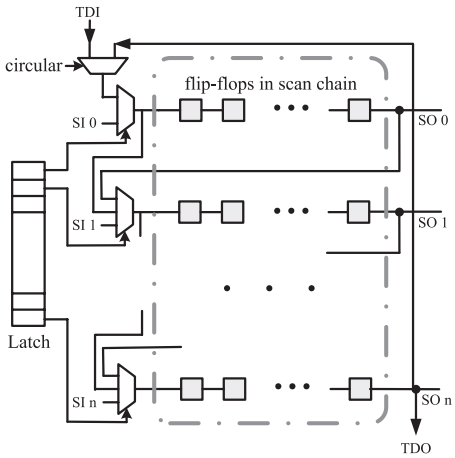


Fig. 8 Scan chain design for silicon debug.

frozen for scan dumps or continue to run. Clocks can be stopped via primary input, IEEE 1149.1 instruction, or system interrupts. Functional mode and scan mode are switched via stop state.

- Single step
  To get the internal contents in successive clock cycles, single step operation is implemented by controlling system clock. The clock is toggled a single clock pulse and applied to the chip. The contents of FFs become the corresponding states of the next clock cycle.
- Scan dump
  Scan chains are standard design for test. During post debug, the scan output (SO) is connected with scan input (SI), which forms a probe. When program runs in the suspect window ($\eta \leq clk\_num \leq \gamma$), the internal contents are scanned out cycle-by-cycle via the probe.

Scan dump contributes the main part of time cost for in-system debug. Two approaches are used to reduce the time cost of scan dump. One approach is to bypass the error-free scan chains when connecting scan chains into a probe. As shown in Fig. 8, we use a latch to control the selection of scan chains. For a mismatched sink, the error must come from its fan-in cone. Assuming only one bug is triggered at a time, the intersection of the fan-in cones of the mismatched sinks are the candidate error source. The FFs, which do not belong to the intersection, must be error-free and need not be scanned out. If all FFs of a scan chain are error-free, it can be bypassed. The other approach is to circle the probe. Since scan dump changes the contents of FFs, it must be recovered to support resume execution. As shown in Fig. 8, the probe output (TDO) is connected to the probe input (TDI), which forms a circle. When the contents of the FFs are obtained, the clock cycle at which the bug is triggered and the FFs which first catch the error are both known. Confirming and fixing the bug can be taken according to design specifications or via physical tools [24].

4.3 Discussion

The proposed debug mechanism combines the advantages of trace-based and scan-based debug mechanisms. The same as existing trace-based techniques such as [26], the proposed mechanism also assumes that the simulation results are available. In some applications, simulation results might be difficult to obtained for simulation speed limitation and non-determination. For simulation speed, FPGA-based emulator provides a much faster approach to obtain gold results before silicon [27]. For non-determination, there are several methods to eliminate it [31]. In fact, how to obtain and utilize the gold results is still an interesting research field today. When trace data cannot be compared with gold results, detecting the violation between trace signals [28] can be as a compromise to determine the end boundary of the suspect window consequently.

5. Experimental Results

In this section, we show the experimental results on ISCAS'89 (s9234, s13207, and s15850) and ITC'99 (b20, b22) benchmark circuits. We simulate the effect of triggered bugs by changing the values of the flip-flops in the benchmark circuits with the same probability. Trace signals are selected under different bandwidth and threshold constraints. We evaluate FF coverage (defined in Sect. 3.2), suspect window size, error-free FFs, debug time, and circuit volume through a series of experiments. The experimental results are described in the following subsections.

5.1 FF Coverage of Trace Signals

FF coverage is an important metric for trace signals selection. The higher FF coverage means the higher observability for detecting the errors. Table 1 shows the results of FF coverage under fixed bandwidth constrains. The trace signals are obtained by Algorithm 1 under given bandwidth and threshold constraints. First, we analyze the influence of the threshold to FF coverage. Before the FF coverage reaches the threshold, the threshold has no influence to FF coverage. For example, the FF coverage of b22 is 88.44% under three different thresholds (0.9, 0.95, and 1) when bandwidth is 8 bits. The highest FF coverage is achieved When threshold is set to 1 (4th, 7th, and 10th column in Table 1). Second, we analyze the influence of bandwidth to FF coverage. As can be seen from Table 1 that, the larger bandwidth is, the higher FF coverage can be obtained. Taking s13207 as

Table 1 Results of FF coverage under fixed bandwidth constraint (%).

	bandwidth = 8 (bits)			bandwidth = 16 (bits)			bandwidth = 32 (bits)		
Circuit	th=0.9	th=0.95	th=1	th=0.9	th=0.95	th=1	th=0.9	th=0.95	th=1
s9234	97.63	98.10	**100**	98.10	98.58	**100**	98.10	98.58	**100**
s13207	89.03	89.03	**89.03**	92.01	94.83	**94.83**	94.36	95.30	**98.12**
s15850	90.79	92.80	**92.80**	90.79	94.47	**94.47**	93.13	95.14	**97.15**
b20	92.04	93.47	**93.47**	92.04	95.10	**95.10**	92.04	95.10	**98.37**
b22	88.44	88.44	**88.44**	89.52	89.52	**89.52**	90.07	91.70	**91.70**

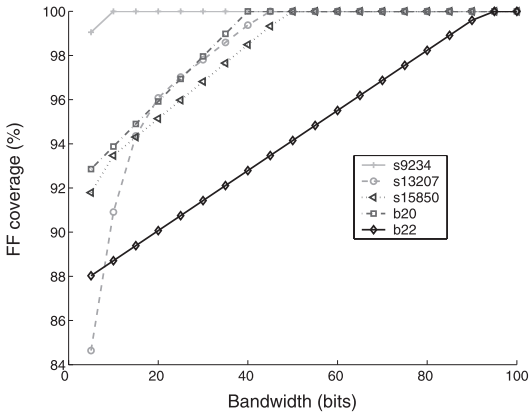


Fig. 9 The FF coverage when *th*=1.

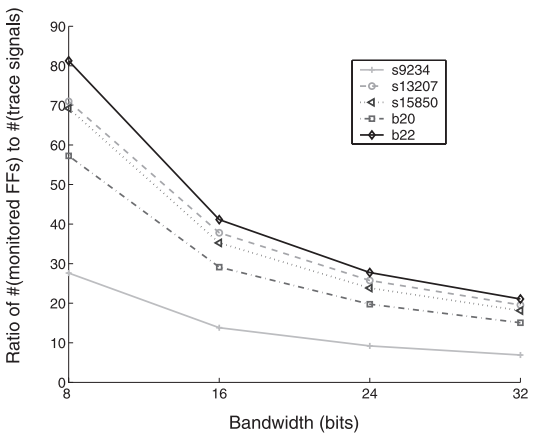


Fig. 10 The extended observability.

example, when the bandwidths are 8, 16 and 32 bits respectively, the corresponding FF coverage are 89.03%, 94.83%, and 98.12% (th=1).

In resource-limited design, the bandwidth need be as small as possible under the requirement of FF coverage. Figure 9 depicts the relationship between various bandwidth and the highest FF coverage (when th=1). The highest FF overages increase as the increase of bandwidth. The FF coverage of all the circuits except b22 is higher than 91% when bandwidth is 32 bits. The circuit b22 is different from the other circuits for its circuit topology. 4.35% FFs are not sinked by the primary outputs in b22. Considering the bugs which can be observed outside, the ratio of $(\#monitored\ FFs\ by\ sinks)/(\#monitored\ FFs\ by\ POs)$ can reach 95.87% when bandwidth is 32 bits. The ratio indicates the percent that the bugs lies in the suspect window. As can be seen in Fig. 9, the most ratios reach 100% with less than 50 bits bandwidth.

Trace signals collect the errors that propagate from other logic. The extended observability for errors is defined as the ratio of the number of monitored FF to the bandwidth. As shown in Fig. 10, all benchmark circuits achieve high extended observability. Since most FFs have been monitored by a small number of sinks, the ratio decreases as the increase of bandwidth. The results of extended observability illustrate the predominance of the proposed mechanism over trace-based mechanism.

5.2 Suspect Window Size

To obtain the suspect window size, bugs need to be inserted into the circuit under debug. In the experiments, it is assumed that bugs are triggered uniformly in the circuit. We trigger one bug at a time and obtain the corresponding suspect window. The average suspect window size is obtained by averaging them. Table 2 depicts the average suspect window size under various bandwidths and thresholds. All suspect window sizes are considerable small compared to the program lengths. It can be seen that, with the same threshold, the suspect window size decreases as the increase of bandwidth. Taking s9234 as example, the suspect window sizes are 4.44, 4.31, and 4.13 at 8, 16 and 32 bits bandwidth respectively when threshold (*th*) is 0.9 and the number of mismatched sink (# mismatched sink in Table 2) is 1. We get the suspect window sizes considering 1, 2, and 3 mismatched sinks (2nd column in Table 2). It can be seen that more mismatched sinks can reduced the suspect windows size. For example, for s13207, the suspect window sizes are 16.55, 15.27 and 14.89 respectively when the number of mismatched sinks increases from 1 to 3.

5.3 Error-Free FFs

As presented in Sect. 4.2, the FF must be error-free if it does not lie in the intersection of the fan-in cones of the mismatched sinks. It can reduces the candidates of bug source in spatial dimension. The scan chain whose FFs are all error-free can be bypassed when connecting the scan chains into a probe. This helps to reduce shift time during scan dump. Since the organization of scan chains is determined by many factors such as placing and routing, the number of

Table 2 Results of average suspect window size.

		bandwidth = 8 (bits)			bandwidth = 16 (bits)			bandwidth = 32 (bits)		
Circuit	# mismatched sinks	th=0.9	th=0.95	th=1	th=0.9	th=0.95	th=1	th=0.9	th=0.95	th=1
s9234	1	4.44	4.62	4.91	4.31	4.42	4.64	4.13	4.19	4.40
	2	4.06	4.09	4.18	3.93	3.98	4.07	3.84	3.82	3.98
	3	3.90	4.02	3.86	3.78	3.88	3.73	3.71	3.69	3.58
s13207	1	16.55	16.55	16.55	13.68	16.22	16.22	12.79	13.13	15.84
	2	15.27	15.27	15.27	12.98	15.01	15.01	12.29	12.30	14.43
	3	14.89	14.89	14.89	12.75	14.22	14.22	11.91	11.94	13.92
s15850	1	6.57	8.85	8.85	6.39	7.85	7.85	6.13	6.79	7.74
	2	6.30	6.60	6.60	6.15	7.69	7.69	5.90	6.04	7.58
	3	5.65	6.60	6.60	5.73	5.62	5.62	5.31	5.83	6.64
b20	1	3.99	4.00	4.00	3.99	4.00	4.00	3.99	3.99	4.00
	2	3.51	3.97	3.97	3.55	3.95	3.95	3.64	3.91	3.92
	3	3.51	3.96	3.96	3.54	3.94	3.94	3.62	3.90	3.90
b22	1	4.00	4.00	4.00	4.00	4.00	4.00	3.99	4.00	4.00
	2	3.98	3.98	3.98	3.97	3.97	3.97	3.93	3.94	3.94
	3	3.97	3.97	3.97	3.96	3.96	3.96	3.92	3.93	3.93

Table 3 Ratio of the number of error-free FFs to the number of total FFs (%).

		bandwidth = 8 (bits)			bandwidth = 16 (bits)			bandwidth = 32 (bits)		
Circuit	# mismatched sinks	th=0.9	th=0.95	th=1	th=0.9	th=0.95	th=1	th=0.9	th=0.95	th=1
s9234	1	35.26	31.95	29.09	38.08	35.64	32.14	43.33	41.18	39.29
	2	44.41	42.42	39.83	48.42	44.65	42.20	51.54	49.58	47.38
	3	44.41	42.42	40.08	48.42	44.65	42.45	51.57	49.58	47.60
s13207	1	44.78	44.78	44.78	49.15	43.31	43.31	52.58	51.25	42.83
	2	46.95	46.95	46.95	49.79	45.34	45.34	52.71	51.51	44.27
	3	48.48	48.48	48.48	49.79	46.88	46.88	52.71	51.51	44.68
s15850	1	33.70	21.10	21.10	36.80	21.37	21.37	39.84	35.91	22.22
	2	34.71	22.28	22.28	36.95	23.52	23.52	39.99	36.29	24.24
	3	35.15	22.28	22.28	37.39	23.96	23.96	41.88	36.29	24.67
b20	1	15.66	13.97	13.97	15.67	12.47	12.47	15.70	12.85	9.46
	2	15.85	14.16	14.16	15.85	12.65	12.65	15.85	13.03	9.65
	3	15.85	14.16	14.16	15.85	12.65	12.65	15.85	13.03	9.65
b22	1	22.63	22.63	22.63	21.68	21.80	21.80	21.56	19.78	19.78
	2	22.75	22.75	22.75	21.80	21.80	21.80	21.68	19.90	19.90
	3	22.75	22.75	22.75	21.80	21.80	21.80	21.68	19.90	19.90

bypassed scan chains correlates with the actual design requirement [25]. Here, we give the ratio of the number of error-free FFs to the number of total FFs, which can show the potential capacity of time reduction. Table 3 shows the results of the ratio under different constraints. It can be seen that the ratio increase as the increase of the number of mismatched sinks. For example, for s13207, the ratio increases as 44.78, 46.95 and 48.48 for 1, 2, and 3 mismatched sinks, respectively when th=0.9. It is because that the intersection becomes smaller as mismatched sinks increase.

5.4 Debug Time

Since trace-based mechanism can be taken with the program execution in real time (via high-speed trace ports), it has no additional time cost (the disadvantage of trace-based mechanism is low observability, as evaluated in Sect. 5.1). Here, we compare debug time between scan-based and Suspect Window based mechanism. Debug time consists of execution time and scan dump time. Let $N, C, C', Size_{sw}, f_n, f_s$ denote the program length (clock cycles), the number of total FFs, the number of FFs excluding the error-free FFs, suspect window size, functional clock frequency, and scan clock frequency, respectively. The execution time and scan dump time is represented in Table 4. Let T_scan, T_sw be

Table 4 The components of debug time.

	Execution time	Scan dump time
Scan-based	$\frac{1}{f_n} \times N$	$\frac{1}{f_s} \times N \times C$
Suspect Window based	$\frac{1}{f_n} \times 2N$	$\frac{1}{f_s} \times Size_{sw} \times C'$

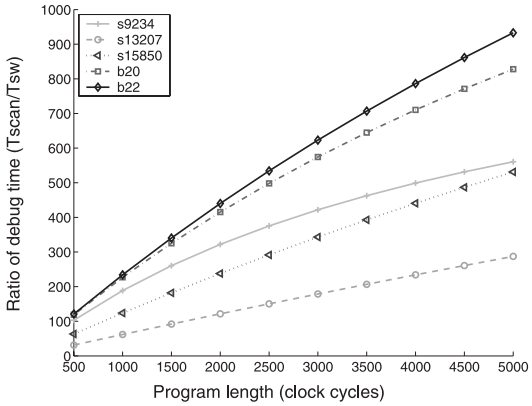


Fig. 11 Debug time comparison.

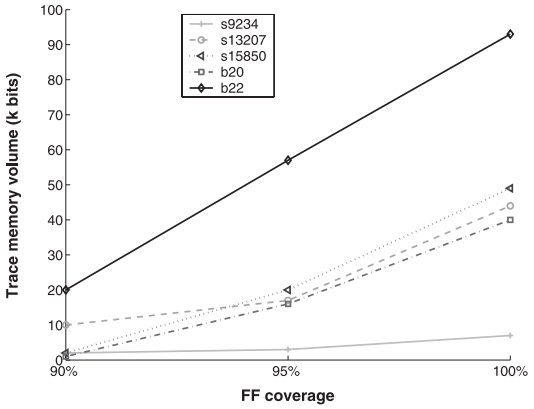


Fig. 12 Trace memory volume.

Table 5 Circuit volume of the sequential logic of the benchmark circuits.

Circuit	s9234	s13207	s15850	b20	b22
# of FFs	228	638	597	490	735

Table 6 Estimated wire area.

Bandwidth (bits)	8	16	32	64
Wire area (um^2)	2.16	4.32	8.64	17.28

the debug time of scan-based and Suspect Window based mechanism, respectively. As a case, Fig. 11 shows the ratio T_scan/T_sw when bandwidth, threshold, the number of mismatched sink are 32, 1, and 1 respectively (let $f_n/f_s = 10$ and $C = C'$ in this experiment for simplicity of explanation). It can be seen that the ratio increases as the program length increases. It is because that the program length influences only the scan time of scan-based mechanism, but not that of Suspect Window based mechanism. The scan dump time of Suspect Window based mechanism correlates with the suspect window, which is unrelated to program length. Since the lengths of real application programs are usually large, the debug time of scan-based mechanism is much longer than that of Suspect Window based mechanism.

5.5 Circuit Volume

The sequential logic in the benchmark circuits is the FFs that they contain. Table 5 shows the numbers of FFs in the benchmark circuits which are used in our experiments. The numbers of FFs range from 228 to 735, which demonstrates that the circuits are representative.

Trace memory is an essential infrastructure in trace-based technology. Experimental studies form [20] indicate that trace memory of hundreds of bits size are acceptable in practice today. Figure 12 shows the needed trace memory volume which meets the requirement of FF coverage assuming the depth of the trace memory is $1k$. As can be seen from Fig. 12, the needed trace memory is relatively small even if the required FF coverage is high. Furthermore, it is common that the trace memory is used as a time-shared resource [29] if one needs to debug larger logic blocks.

Many factors may influent wire area such as routing algorithm. We estimate wire area according to the model presented in [30]. In this model, wire width is 3λ and the line-to-line pitch is 6λ, where λ is the minimum feature size. In 90 nm technology, λ=45 nm. Assume the wire length is 1 mm, the estimated wire area is shown in Table 6. As can be seen, the wire area is very small even if the bandwidth is as many as 64 bits.

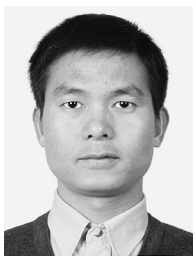
6. Conclusions

As design complexity continues to increase, structured debug methods become crucial for decreasing the length of debug cycle during silicon debug. In this paper, we have proposed a novel silicon debug mechanism based on Suspect Window. Suspect Window is introduced to represent the clock cycle range in which the bug is triggered. We present an efficient approach to determine the suspect window. By converting the circuit to abstract graph, an algorithm of selecting the trace signals, which can collect the most errors in the circuit, is presented. The selected signals are traced in real-time during the first run with the proposed debug mechanism. The suspect window can be determined according the trace information and abstract graph. In the rerun process, scan dumps are taken only in the suspect window to locate the bug exactly. The proposed mechanism achieves high observability compared to trace-based mechanism, and reduces the debug time greatly compared to scan-based mechanism. Experimental results demonstrate that the proposed mechanism can achieve high observability at small time cost.

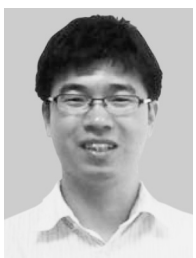
References

[1] D. Josephson and B. Gottlieb, "The crazy mixed up world of silicon debug," Proc. IEEE Custom Integrated Circuits Conference, pp.665–670, 2004.
[2] M. Abramovici, "In-system silicon validation and debug," IEEE Design & Test of Computers, vol.25, no.3, pp.216–223, 2008.
[3] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-debug infrastructure for SoCs," Proc. ACM/IEEE Design Automation Conference, pp.7–12, 2006.

[4] W.W. Hu and J. Wang, "Making effective decisions in computer architects real-world: Lessons and experiences with Godson-2 processor designs," J. Computer Science and Technology, vol.23, pp.620–632, 2008.
[5] K. Ho Fai and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.28, no.2, pp.285–297, 2009.
[6] P. Sung-Boem and S. Mitra, "IFRA: Instruction Footprint Recording and Analysis for post-silicon bug localization in processors," Proc. ACM/IEEE Design Automation Conference, pp.373–378, 2008.
[7] C. Yen, T. Lin, H. Lin, K. Yang, T. Liu, and Y. Hsu, "A general failure candidate ranking framework for silicon debug," Proc. IEEE VLSI Test Symposium, pp.352–358, 2008.
[8] D. Josephson, "The good, the bad, and the ugly of silicon debug," Proc. ACM/IEEE Design Automation Conference, pp.3–6, 2006.
[9] W.D. Boer and B. Vermeulen, "Silicon debug: Avoid needless respins," Proc. IEEE Electronics Manufacturing Technology Symposium, pp.1–5, 2004.
[10] Y.C. Hsu, "Maximizing full-chip simulation signal visibility for efficient debug," Proc. International Symposium on VLSI Design, Automation and Test, pp.1–5, 2007.
[11] D. Josephson, "The manic depression of microprocessor debug," Proc. IEEE International Test Conference, pp.657–663, 2002.
[12] C. Ciordasa, A. Hanssona, K. Goossensb, and T. Bastena, "A monitoring-aware network-on-chip design flow," J. Syst. Archit., vol.54, pp.397–410, 2008.
[13] G.J. van Rootselaar and B. Vermeulen, "Silicon debug: Scan chains alone are not enough," Proc. IEEE International Test Conference, pp.892–902, 1999.
[14] N. Minegishi, K. Asano, H. Suzuki, K. Okada, and T. Kan, "A debug system for heterogeneous multiple processors in a single chip for multimedia communication," IEICE Trans. Inf. & Syst., vol.E85-D, no.10, pp.1571–1578, Oct. 2002.
[15] K. Ho Fai and N. Nicolici, "Automated trace signals identification and state restoration for improving observability in post-silicon validation," Proc. IEEE/ACM Design, Automation and Test in Europe Conference, pp.1298–1303, 2008.
[16] Y. Higami, S. Kajihara, I. Pomeranz, S. Kobayashi, and Y. Takamatsu, "On finding don't cares in test sequences for sequential circuits," IEICE Trans. Inf. & Syst., vol.E89-D, no.11, pp.2748–2755, Nov. 2006.
[17] B. Vermeulen, M.Z. Urfianto, and S.K. Goel, "Automatic generation of breakpoint hardware for silicon debug," Proc. IEEE/ACM Design Automation Conference, pp.514–517, 2004.
[18] O. Caty, P. Dahlgren, and I. Bayraktaroglu, "Microprocessor silicon debug based on failure propagation tracing," Proc. IEEE International Test Conference, pp.1–10, 2005.
[19] C. MacNamee and D. Heffernan, "Emerging on-chip debugging techniques for real-time embedded systems," Computing & Control Engineering J., vol.11, pp.295–303, 2000.
[20] M. Abramovici, "Experience and opinion (design for debug)," Proc. IEEE International Workshop Silicon Debug Diagnosis, 2006.
[21] Y.C. Hsu, F. Tsai, W. Jong, and Y.T. Chang, "Visibility enhancement for silicon debug," Proc. ACM/IEEE Design Automation Conference, pp.13–18, 2006.
[22] E. Anis and N. Nicolici, "On using lossless compression of debug data in embedded logic analysis," Proc. IEEE International Test Conference, pp.1–10, 2007.
[23] H. Hao and K. Bhabuthmal, "Clock controller design in SuperSPARC II microprocessor," Proc. IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp.124–129, 1995.
[24] C. Boit, R. Schlangen, U. Kerst, and T. Lundquist, "Physical techniques for chip-backside IC debug in nanotechnologies," IEEE Design & Test of Computers, vol.25, pp.250–257, 2008.
[25] D. Wang, X. Fan, X. Fu, H. Liu, H. Li, Y. Hu, and X. Li, "The design-for-testability features of a general purpose microprocessor," Proc. IEEE International Test Conference, pp.1–9, 2007.
[26] J.S. Yang and N.A. Touba, "Expanding trace buffer observation window for in-system silicon debug through selective capture," Proc. IEEE VLSI Test Symposium, pp.345–351, 2008.
[27] C.L. Chuang and W.H. Cheng, C.N. J. Liu, and D.J. Lu, "Hybrid approach to faster functional verification with full visibility," IEEE Des. Test Comput., vol.24, no.2, pp.154–162, 2007.
[28] J. Geuzebroek and B. Vermeulen, "Integration of hardware assertions in systems-on-chip," Proc. IEEE International Test Conference, pp.1–10, 2008.
[29] X. Liu and Q. Xu, "Interconnection fabric design for tracing signals in post-silicon validation," Proc. ACM/IEEE Design Automation Conference, pp.352–357, 2009.
[30] Michael John Sebastian Smith, Application-Specific Integrated Circuits, (Chapter 17.2,) Addison Wesley Longman, 1997.
[31] S.R. Sarangi, B. Greskamp, and J. Torrellas, "CADRE: Cycle-accurate deterministic replay for hardware debugging," Proc. International Conference on Dependable Systems and Networks, pp.301–312, 2006.



Jianliang Gao received the B.S. degree in computer science from National University of Defense Technology, Changsha, China, in 2001. He is currently working toward the Ph.D. degree on computer science at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His research interests include debug, test, and verification for VLSIs.



Yinhe Han received the B. Eng. degree from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2001, and the M. Eng. and Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2003 and 2006, respectively. He is currently an Associate Professor at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include VLSI Design and Test, Reliable System and Architecture. Dr. Han was a recipient of the Test Technology Technical Council Best Paper Award at the Asian Test Symposium 2003. He is also a member of ACM/IEEE.

Xiaowei Li received his B.Eng. and M.Eng. degrees in Computer Science from Hefei University of Technology, China, in 1985 and 1988, respectively, and his Ph.D. degree in Computer Science from the Institute of Computing Technology, Chinese Academy of Sciences in 1991. From 1991 to 2000, he was an Assistant Professor and an Associate Professor (since 1993) in the Department of Computer Science, Peking University, China. During 1997 and 1998, he was a Visiting Research Fellow in the Department of Electrical and Electronic Engineering, the University of Hong Kong. During 1999 and 2000, he was a Visiting Professor in the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. He joined the ICT, CAS as a Professor in 2000. He is now the deputy director of the Key Lab. of Computer System and Architecture. Dr. Li served as IEEE Asian Pacific Regional TTTC (Test Technology Technical Council) Vice Chair since 2004. He served as the Steering Committee Vice-chair of IEEE Asian Test Symposium (ATS) since 2007, and served as the Program Committee Co-Chair for ATS'2003 and General Co-Chair for ATS'2007. He also served as the Steering Committee Chair of IEEE Workshop on RTL and High Level Testing (WRTLT), and served as the General Chair for WRTLT'2003 and WRTLT'2007. In addition, he serves on the Technical Program Committee of several IEEE and ACM conferences, including VTS, DATE, ASP-DAC, PRDC, etc. He also serves as member of editorial board of JCST, JOLPE, JETTA, etc.