Improved Trace Buffer Observation via Selective Data Capture Using 2-D Compaction for Post-Silicon Debug

Joon-Sung Yang, *Member, IEEE*, and Nur A. Touba, *Fellow, IEEE*

***Abstract*—This paper presents a novel technique for extending the capacity of trace buffers when capturing debug data during post-silicon debug. It exploits the fact that is it not necessary to capture error-free data in the trace buffer since that information can be obtained from simulation. A selective data capture method is proposed in this paper that only captures debug data during clock cycles in which errors are present. The proposed debug method requires only three debug sessions. The first session estimates a rough error rate, the second session identifies a set of suspect clock cycles where errors may be present, and the third session captures the suspect clock cycles in the trace buffer. The suspect clock cycles are determined through a 2-D compaction technique using multiple-input signature register signatures and cycling register signatures. Intersecting both signatures generates a small number of suspect clock cycles for which the trace buffer needs to capture. The effective observation window of the trace buffer can be expanded significantly, by up to orders of magnitude. Experimental results indicate very significant increases in the effective observation window for a trace buffer can be obtained.**

***Index Terms*—2-D compaction, observation window, post-silicon debug, selective data capture, trace buffer.**

I. Introduction

With the advancement of process technology, larger and more complex devices are being manufactured. Along with the increased complexity, there are shorter time to market requirements. Moreover, simulation models are increasingly less accurate in nanometer technologies. Consequently, post-silicon debug has become a critical stage in the design process.

Pre-silicon verification techniques play a significant role to provide an equivalence check between the implemented design and its specification using techniques such as functional simulation, formal verification, etc. [5], [6], [13]. However, given the difficulty in modeling complex ICs and the limited computational resources for simulation or verification [21], the first silicon may not be error-free.

Manuscript received February 18, 2011; revised September 13, 2011; accepted December 15, 2011. Date of publication January 31, 2012; date of current version January 17, 2013. This work was supported in part by the National Science Foundation under Grant CCF-0916837.

J.-S. Yang is with Intel Corporation, Austin, TX 78746 USA (e-mail: js21.yang@gmail.com).

N. A. Touba is with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712-1084 USA (e-mail: touba@ece.utexas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2012.2183399

Many bugs and errors such as logic errors, timing errors, physical design errors, etc., may go undetected in the pre-silicon verification phase and may cause nonconforming chip behavior. Post-silicon debug starts when the first silicon arrives [17]. [2] shows that post-silicon debug has become a bottleneck which could consume more than 35% of the chip development cycle. The International Technology Roadmap (ITRS) for Semiconductors indicates that labor intensive post-silicon techniques such as mechanical probing could bring an exponential time increase [16]

Design-for-debug (DFD) methodologies can be used to add debug support for a more speedy and accurate process. They are inserted in the design to extract the information from the logic and to enhance the internal signal observability. Finding information about when (temporal) and where (spatial) failures occur is the key issue in post-silicon debug. Scan chains and on-chip memories have been used to provide internal signal information for the silicon debug process.

Scan chains are widely used to support manufacturing test by allowing a dump of the system state. Acquisition of internal signal information is the key issue of post-silicon debug. Therefore, scan chains are reused for post-silicon debug. Scan-based debug techniques [7], [8], [9], [21] can achieve high observability of internal signals. However, it requires halting the system to scan out responses from the circuit-under-debug (CUD). Circuit misbehavior can be identified via internal system states which are read out through the scan chains.

To monitor continuous operation of the CUD, trace buffers are also commonly used to capture data from a limited number of signals during system operation [2], [4], [18], [19]. They are very helpful as they provide real-time at-speed observation of signals across many clock cycles. Unfortunately, they are a limited resource and store a limited amount of data. The amount of data that can be acquired is determined by the size of trace buffers. There have been some techniques proposed to increase the effectiveness of the trace buffer by compressing the data for debug. As suggested in [3], the width and depth of the observation window in the trace buffer can be defined. One can view the width of the observation window provided by a trace buffer as the number of signals observed each clock cycle and the depth of the observation window as the number of clock cycles over which the signals are observed.

In this paper, a new post-silicon debug technique for electrical bugs that utilizes a trace buffer is presented. Preliminary results were presented in [19]. A key feature of the proposed approach is that it significantly expands the observation window size by selectively storing the debug data.

Section II provides a review of the related work. Section III gives an overview of the proposed debugging scheme. Section IV discusses the three pass debug procedure in detail. Section V describes hardware architecture of the proposed scheme. Experimental results are shown in Section VI and conclusions are given in Section VII.

II. Related Work

In [1] and [10], post-silicon debug techniques are proposed to enhance the signal visibility for combinational signal values. They try to reconstruct the values of more internal signals than captured each clock cycle in the trace buffer and hence these techniques expand the effective width of the observation window, but not its depth. [12] shows an automated data restoration method for sequential circuits. They try to restore the missing states which are not captured in a trace buffer by defining forward restorability in the circuit.

In [3], a debug architecture is proposed for embedded logic analysis using lossless compressions techniques. Lossless compression methods based on different dictionary coding algorithms are studied and implemented using content-addressable memory (CAM) to compress the debug data stored in a trace buffer. This compression technique can expand the depth of the observation window in embedded logic as well. Experiments were run on an MP3 decoder [15] and results for MP3 stereo data show that the observation window size can be expanded up to 3.45 times larger. However, the achievable compression ratio via dictionary coding varies greatly depending on how correlated the data is. While the amount of compression is modest, a useful feature of the compression architecture in [3] is that it is a one-pass debugging scheme which does not require re-running the debug sessions. Debugging in circuits that have sources of non-determinism like asynchronous interfaces may not allow reproducing the circuit operation, hence, a one-pass debugging method is necessary to help finding the root cause of errors.

If the behavior is deterministic and repeatable, as stated, scan chains can be used. A debug module with scan chains can provide start, stop, resume and single-step operations [20]. Scan dumps provide high observability of internal states after the triggering event. Scan chains can be used in a binary search manner which iteratively divides the search space in half until the first cycle that the error is activated and captured. Hence, multiple debug sessions may be required to narrow down the temporal and spatial error information with scan-based debug. Trace buffers can store a limited number of consecutive real-time internal signals. [4] proposes a debug method using trace buffers that is applicable to repeatable debug cases. For example, automatic test equipment (ATE) and a target application board that can operate synchronously to allow cycle deterministic debug. The debug architecture in [4] requires re-running the debug session many times which compacts the observed signals in a MISR and stores MISR signatures in the trace buffer over progressively finer resolutions of time in each debug session. This approach implements an accelerated binary search that gradually zooms in on clock cycles in which errors may occur. When multiple trace buffer data and golden signature comparisons leave the size of debug data search range small enough to capture in the trace buffer, then the trace buffer is used to capture all the data in the remaining portion of the current search. This is nice and effective idea for accelerating debug methods based on binary search, however, it may not be a suitable replacement for more conventional applications of trace buffers because it can require a large number of debug sessions.

In this paper, on a cycle-accurate deterministic test environment, a new debug method for expanding the depth of the observation window for a trace buffer is proposed to enhance the efficiency of trace buffer usage. The proposed method requires only three debug sessions. It can expand the depth of trace buffer by orders of magnitude which can greatly speed up the debug process. It is also compatible with other methods for expanding the width of the observation window. The proposed method exploits the fact that it is not necessary to capture error-free data in the trace buffer since that information is obtainable from simulation. The trace buffer need only capture data during clock cycles in which errors are present. During the first debug session, the rough error rate is measured, in the second debug session, a set of suspect clock cycles where errors may be present is determined, and then in the third debug session, the trace buffer captures only during the suspect clock cycles. The suspect clock cycles are determined through a 2-D compaction technique using a combination of multiple-input signature register (MISR) signatures and cycling register signatures. By intersecting the signatures, the proposed 2-D compaction technique leaves only a small set of remaining suspect clock cycles for which the trace buffer needs to capture data.

III. Overview of Proposed Scheme

The proposed scheme involves adding a debug module to a trace buffer which is able to support three operations which are executed in separate debug sessions. The signals being sampled in each clock cycle will be collectively referred to here as the "$data\ word$". Three debug sessions are needed to capture possibly erroneous data. The $error\ rate$ (i.e., data word errors per clock cycle) is estimated using lossy compression with a parity generator in the first debug session. Based on the estimated error rate, the maximum possible expanded observation window size can be computed for the given trace buffer size. The following equation shows the rough window size:

$$window\_size \leq \frac{buffer\_size}{error\_rate}.$$

$window\_size$ is the expanded observation window size, $buffer\_size$ is the number of data words that can be stored in the trace buffer, and $error\_rate$ is the number of data word errors (any data bit errors in a data word) per clock cycle. Since all the erroneous data words must be stored in the trace buffer, the observation window cannot contain more errors than can fit in the trace buffer.

In the second debug session, during the clock cycles in the maximum expanded observation window range from the first debug session, the 2-D compaction is activated to determine the suspect set of clock cycles in which errors may occur. The 2-D compaction consists of using both a MISR and a cycling register and data words are compacted by both compactors. Signatures from both compactors are intersected to identify the suspects. From the MISR, k signatures are generated and each signature

compacts $window\_size/k$ consecutive data words. A cycling register of length m compacts the data words such that every mth data word is XORed together in each signature. Because data words cycle with the length (m) of the cycling register, the cycling register indicates whether erroneous data exists in each module m set of data words. An erroneous data word produces an erroneous MISR signature and erroneous cycling register signature, respectively. Since erroneous data words corrupt signatures in the compactors, the suspect clock cycles in which errors occurred can be identified by observing the intersections of the faulty signatures from both compactors (MISR and cycling register).

In the third debug session, during the suspect clock cycles, debug data is captured in the trace buffer. If there is no aliasing in the compactors, then capturing all suspect clock cycles guarantees that all errors in the expanded observation window will be captured in the trace buffer. The proposed debug method depends on the signatures from lossy compressors, however, as will be shown, the probability of aliasing is extremely small for low error rates (i.e., error rates below 1%). For debug, where the part has already passed a manufacturing test, errors may manifest at certain corner cases such as some specific voltage levels and frequencies of the system. The proposed scheme exploits debug cases with low error rates allowing selective capture by lossy compactors to achieve significant observation window size expansion which greatly enhances visibility.

IV. Details of the Three Debug Sessions

The following subsections describe each of the debug sessions in detail.

A. Session 1—Estimating Expanded Observation Window Size Based on Parity

In the first debug session, the debug module computes the parity of the data word each clock cycle and stores it in the trace buffer. When the trace buffer gets full, the older data is overwritten, so at the end of the debug session, the tracefer contains the parity information for the last set of data words. This information is downloaded to a workstation and compared with the fault-free parity values computed through fault-free simulation. By comparing the fault-free parity with the observed parity, the number of erroneous data words can be roughly estimated. Because single-bit parity detects only the odd errors in the data word, only roughly half of errors in the data words are probabilistically detected during the first debug session. A rough estimate of the error rate can be obtained by multiplying the number of parity errors by 2 and dividing by the total number of parity bits stored in the trace buffer. For example, if two parity bits in a 512 byte trace buffer are erroneous, then the error rate is $(2\ \text{bit}*2)/(512*8) = 0.097\%$. The trace buffer size divided by the error rate is used to estimate the maximum trace buffer observation window size as explained in Section III. Note that the achieved observation window size may be considerably smaller than the maximum. The reasons for this will become clear later and will be discussed in Section VI. The maximum window size as used as the starting point for 2-D compaction in the second session.

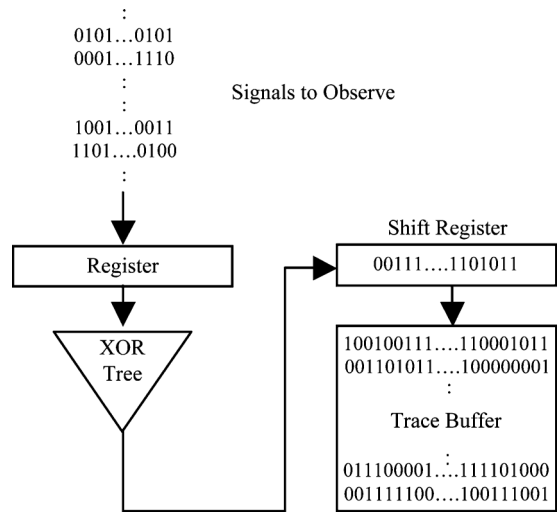


Fig. 1. Session 1: Parity generation.

Fig. 1 illustrates the operation of the debug module in the first session. Note that the XOR tree can be pipelined as necessary to meet timing requirements.

B. Session 2—Determining Suspects

2-D compaction is performed in the second debug session. Signatures are generated using the MISR and cycling register beginning from the starting point of the maximum observation window estimated in the first debug session. The trace buffer is used to store both the MISR signatures and the cycling register signatures. Assume k locations are allocated to store the MISR signatures and m locations are allocated to store cycling register signatures. The MISR signatures are stored every $window\_size/k$ clock cycles and the MISR is reset at that time so that the signatures are independent. The cycling register signatures are generated by XORing together the data word coming in with one of the m locations in the trace buffer pointed to by a mod-m address counter. In this manner, the cycling register will generate m signatures which consist of the XOR of every mth data word.

1) 2-D Compaction:

2-D Compaction Example: Fig. 2 illustrates the operation of the 2-D compactor. A phase shifter is placed before the MISR. The purpose of this phase shifter is to eliminate shift correlation among the data feeding into the MISR (and it can also be used to perform space compaction if the MISR is smaller than the data word). Each MISR signature compacts a consecutive sequence of $window\_size/k$ data words. A symbolic expression of the data words compacted in the signatures is shown in Fig. 3 for a small example with a total of 20 clock cycles of data words with $k = 5$ and $m = 5$. A MISR signature is generated every $(window\_size/k) = 4$ clock cycles. MS_1 represents the first MISR signature and C_1 denotes the data word in clock cycle 1. MISR signature 1 compacts the data words in cycles 1 through 4 which is expressed as $MS_1 = \{C_1, C_2, C_3, C_4\}$. The cycling register compacts every mth data word. The first signature in the

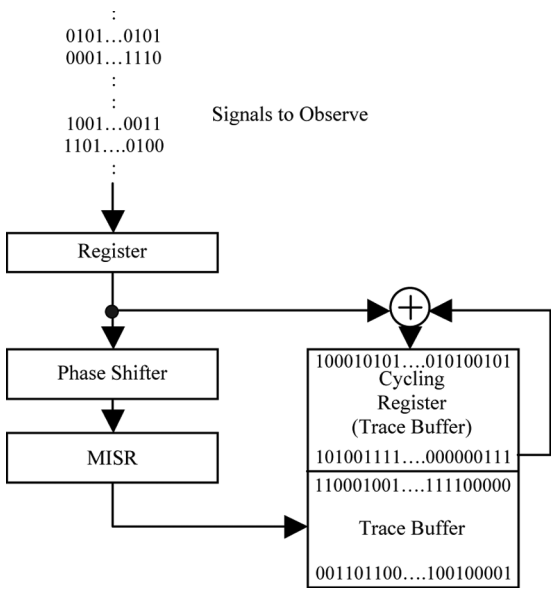


Fig. 2. Session 2: General concept for 2-D compaction.

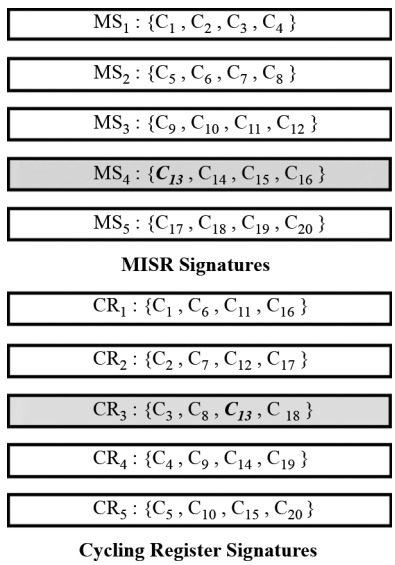


Fig. 3. Example of symbolic 2-D compaction using MISR with $k = 5$ and cycling register with $m = 5$ for 20 clock cycles.

cycling register in Fig. 3 is denoted as CR_1 and is expressed as $\{C_1, C_6, C_{11}, C_{16}\}$ since $m = 5$.

If C_{13} is faulty, then MS_4 and CR_3 will mismatch with the fault-free signatures assuming there is no aliasing. The mismatching signatures, MS_4 and CR_3 are highlighted in gray in Fig. 3.

Aliasing Probability Analysis: The probability of aliasing in the MISR depends on the size of MISR. For a 32 bit MISR, it is 2^{-32}, and for a 16 bit MISR, it is 2^{-16}. Hence, for a sufficiently large MISR, this aliasing probability is negligible. Aliasing in a cycling register signature occurs when an even number of bit errors occur in the same bit position. The probability of aliasing in a cycling register signature when the error rate is low is approximately equal to the probability of a two-bit error occurring in the same bit position in a cycling register signature (the probability of 4-bit and higher even bit errors are negligibly small compared with 2-bit errors) which is equal to

$$P(Aliasing)= 1-\{1-(C_2^{NECR})(Bit\ Error\ Rate)^2(1-Bit\ Error\ Rate)^{NECR-2}\}^{WORDSIZE}$$

where $NECR$ denotes the number of $data\ words$ compacted in the cycling register signature. For low bit error rates, the aliasing probability is negligible for the cycling register as well.

The reason why the cycling register is used instead of a MISR is that more read ports from the trace buffer would be needed to implement a MISR feedback which has multiple tap points. The cycling register feedback has only one tap point plus the data coming in, hence, one word from the trace buffer is read, XORed with one word coming in and written back. However, if a MISR is used, multiple words from the trace buffer are needed to generate the feedback each clock cycle which would require more read ports.

2) Cycle Tag Data Generation: As shown in Fig. 3, erroneous data in C_{13} corrupts signatures in the MISR and cycling register and they are highlighted in gray color. By finding the intersection of the mismatching signatures, the suspect clock cycles can be identified. In Fig. 3, intersecting MS_4 with CR_3 gives C_{13}.

At the end of second session, all the MISR signatures and cycling register signatures in the trace buffer are downloaded to a workstation where they are compared with the fault-free signatures obtained from simulation. The set of suspects are formed by intersecting all mismatching MISR signatures with all mismatching cycling register signatures and including any clock cycle that is in the intersection.

In the third session, the trace buffer must capture during the suspect clock cycles. The information about when to capture is downloaded into the trace buffer before the start of the third session. The information is represented as a set of "cycle tag bits", one for each clock cycle in the observation window. Each suspect clock cycle is indicated by setting its corresponding cycle tag bit to 1 and each vindicated clock cycle is denoted by setting its corresponding cycle tag bit to 0. For the example in Fig. 3, the cycle tag bit for C_{13} is set to 1, and 0 is assigned to the rest of the clock cycles. In this case, the 20 bit tag information is generated as $0000000000001_{(C13)}0000000$. In the third session, the tag bits are cycled through and used to trigger the trace buffer to capture during the suspect clock cycles.

Fig. 4 shows the algorithm for computing the cycle tag bits. Each tag bit has a value of 1 only when the corresponding clock cycle belongs to both a mismatching MISR signature and mismatching cycling register signature. Assume the ith MISR signature is mismatching, then the suspect clock cycles in the ith MISR signature are $[\{(i-1)^*(window\_size/k)+1\}$ cycle $\{i^*(window\_size/k)\}$ cycle]. With these clock cycles, (cycle $mod\ m$)th cycling register signature is checked. If the corresponding cycling register signature is faulty, 1 is assigned to the corresponding cycling tag bit. The following shows the

```
Input: MISR signature (MS), Cycling Register
       signatures(CR), Golden MISR signature(GMS),
       Golden Cycling Register signatures(GCR)
Output: Cycle Tag Bits

currentMR = 0; currentGMS = 0;
cycletagbit[numData] = 0;
while( current_MS < last_MS ){
   List all the element MS[current_MS];
   if( equality(MS[current_MS], GMS[currentGMS]) ){
      while( !visited all the element ){
         cycletagbit[element] = 0;
         next_element;
      }
   }
   else{
      while( !visited all the element ) {
         if( equality(corresponding_CR, GCR) )
               cycletagbit[element] = 0;
         else  cycletagbit[element] = 1;
       next_element;
      }
   }
   current_MS++; current_GMS++;
}
```

Fig. 4. Cycle tag bit generation algorithm.

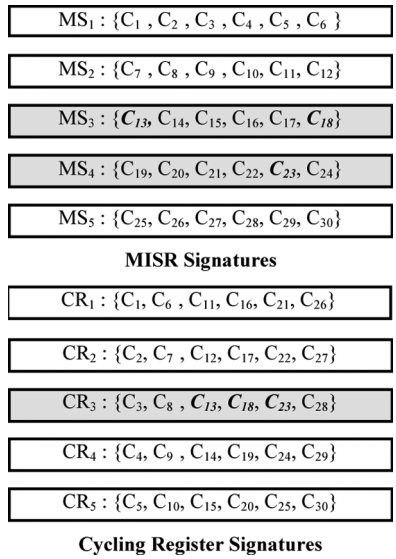


Fig. 5. Example of symbolic 2-D compaction using MISR with $k = 5$ and cycling register with $m = 5$ for 30 clock cycles.

elements of the ith MISR signature and how the corresponding cycling registers are checked.

$$
\begin{aligned}
&i\text{-}th\ MISR\ Signature\text{:}\\
&\quad \left\{(i-1)*\left(\tfrac{window\_size}{k}\right)+1\right\}\ cycle \sim \left\{i*\left(\tfrac{window\_size}{k}\right)\right\}\ cycle\\
&Corresponding\ Cycling\ Registers\text{:}\\
&\quad \left\{(i-1)*\left(\tfrac{window\_size}{k}\right)+1\right\}\ mod\ m\\
&\qquad\qquad \vdots\\
&\quad \left\{(i)*\left(\tfrac{window\_size}{k}\right)+1\right\}\ mod\ m.
\end{aligned}
$$

One complication that arises is that since the cycle tag bits are stored in the trace buffer to capture suspect clock cycles in the third session, the size of a trace buffer could become a limiting factor on the size of expanded observation window. If a cycle tag bit corresponds to one clock cycle, then the maximum number of cycle tag bits that can be stored in the trace buffer sets an upper boundon the observation window size. For example, if a 1 kB trace buffer is used, it can only store tag information for up to 8192 bits and hence the observation window would be limited to maximum 8192 clock cycles. This may be lower than necessary.

To avoid this limitation, it may be necessary to compress the cycle tag bits. A simple way to do this is to have each tag bit correspond to a consecutive sequence of clock cycles rather than a single clock cycle. The cycle tag bits can be initially computed one per clock cycle, and then successive cycle tag bits can be grouped and compressed into a single bit. One compressed bit is used to represent the whole group. Since a single bit represents multiple clock cycles, the observation window can be extended with the same number of tag bits stored in a trace buffer. To form the cycle tag bit compression, successive cycle tag bits are investigated. A compressed tag bit has value 0 when there are no 1 s in a group, and it has 1 if there is at least one 1. If the compressed tag bit is 1, the trace buffer must capture during all the corresponding clock cycles.

Fig. 5 shows a small example of 2-D compaction with a total of 30 clock cycles with $k = 5$ and $m = 5$. C_{13}, and C_{23} are erroneous and corrupt MS_3, MS_4, and CR_3. Intersecting the signatures identifies C_{13}, C_{18}, and C_{23} as suspects. The following 30 bit tag data is generated:

$$0000000000001_{(C13)}00001_{(C18)}00001_{(C23)}0000000.$$

If tag compression is used to group 2 cycle tag bits into one compressed tag bit, then the 30 bit tag data is compressed into the following 15-bits $0000001_{(C13,C14)}01_{(C17,C18)}001_{(C23,C24)}000$ which is also illustrated in Fig. 8.

C. Session 3—Capturing Suspect Clock Cycles

In the third debug session, suspect clock cycles are selectively captured in a trace buffer using the tag information. The cycle tag data generated in Session 2 is stored in the trace buffer at the start of Session 3. During session 3, when in the expanded observation window from Session 1, the trace buffer captures data whenever the cycle tag bit (or compressed cycle tag big) for the corresponding clock cycle has a value of 1 indicating it is a suspect.

The tag data generated in Session 2 is stored in the trace buffer at the start of Session 3. During Session 3, when in the expanded observation window, the trace buffer captures data whenever the tag bit (or compressed tag bit) for the corresponding clock cycle has a value of 1 indicating it is a suspect. As illustrated in Fig. 6, both the tag bits and the captured data are stored in the trace buffer. As the tag data is read out of the trace buffer, it can be overwritten in the trace buffer by the captured data. Enough slack has to be incorporated so that the captured data never overwrites any unread tag data.

In the 2-D compaction example in Fig. 3, C_{13} is identified as a suspect clock cycle and the cycle tag bits were generated

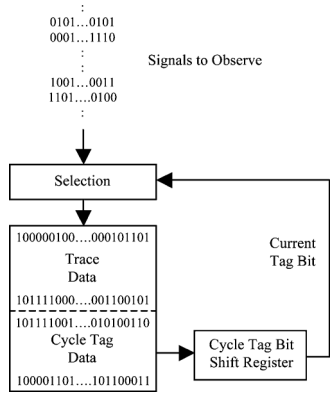


Fig. 6. Session 3: Selective debug data capture with cycle tag data.

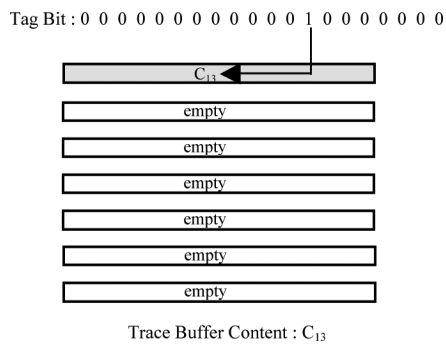


Fig. 7. Data in trace buffer for 15 tag bits from example in Fig. 3.

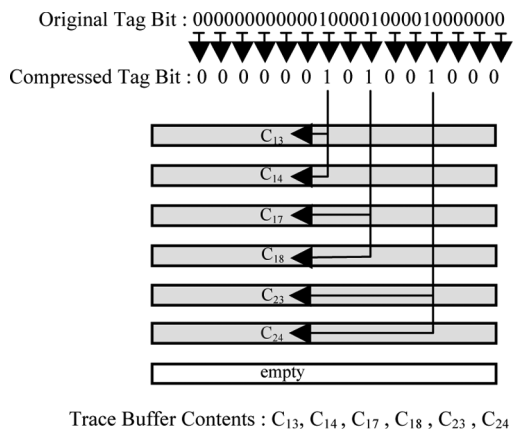


Fig. 8. Data in trace buffer for 15 compressed tag bits from example in Fig. 5.

without compression as $0000000000001_{(C13)}0000000$. The selective data capture process is shown in Fig. 7 after the third session. For the example in Fig. 5, the 30 cycle tag bits are generated and compressed down to 15 tag bits as illustrated in Fig. 8. As a result of this compression, in addition to the suspects (C_{13}, C_{18}, and C_{23}) from the original 30 cycle tag bits, three additional clock cycles are also captured in the trace buffer, namely (C_{14}, C_{17}, and C_{24}).

The proposed scheme uses the information from 2-D compaction to significantly increase the size of observation window by the selective data capture. Expanding the trace buffer observation window gives visibility over wider range of data. Hence the proposed approach reduces the overall silicon debug time.

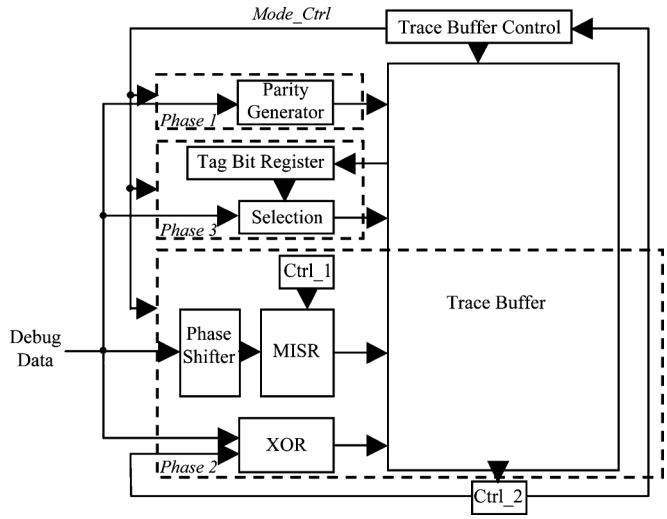


Fig. 9. Hardware architecture of aroposed debug module for three debug sessions.

V. Hardware Architecture of Debug Module

The hardware architecture for a proposed debug module is illustrated in Fig. 9. To perform the operations discussed in Section IV, the debug module activates different functional blocks using the $Mode\_Ctrl$ signal.

In Session 1, the $Mode\_Ctrl$ signal enables the parity generation mode and the phase 1 block shown in Fig. 9 performs the operation. In this mode, the debug data is compressed via an XOR tree to generate a single parity bit each clock cycle. The single parity bits are stored in the trace buffer and used for estimating the error rate in the data words. This information is used to roughly estimate the expanded observation window size. As discussed in Section IV, a parity generator can be pipelined to avoid timing issues.

Once session 1 is finished, the $Mode\_Ctrl$ signal activates the 2-D compactors (a MISR and a cycling register) in the phase 2 block. The MISR and cycling register signatures are generated and stored in the trace buffer. Since the number of intersections is generally minimized when using an equal number of MISR signatures and cycling register signatures, half of the trace buffer is used to store MISR signatures and the other half is used to operate as a cycling register by a modulo operation and to store cycling register signatures. $Ctrl\_1$ block periodically resets a MISR so as to generate an independent signature and $Ctrl\_2$ reads out the intermediate cycling register signature to run the modulo operation with incoming debug data. Cycling tag bits are generated using both signatures by finding the intersections from mismatching signatures. With a low error rate in a debug data, the tag information compression helps to significantly expand the observation window size.

In Session 3, the $Mode\_Ctrl$ signal activates the selection logic in the phase 3 block which selectively captures the debug data based on the tag information. A cycle tag bit shift register is used to provide serial access to the tag bits so they can be checked one bit at a time each clock cycle. The suspect clock cycles are selectively captured whenever the tag bit is 1. The $Selection$ block is programmed with the tag bit compression

TABLE I
RESULTS FOR PROPOSED METHOD FOR DIFFERENT SIZE TRACE BUFFERS AND ERROR RATES

Size of Trace Buffer	Error Rate Percentage	Conventional Observation Window	Expanded Observation Window	Expansion Ratio	Error Aliasing Percentage
ARM Based Design [14]					
512 Byte	0.016	128	19456	152	0
	0.051	128	12032	94	0
	0.097	128	8576	67	0
	0.513	128	3072	24	0
	1.387	128	1792	14	0
1K Byte	0.016	256	28672	112	0
	0.051	256	19968	78	0
	0.097	256	17152	67	0
	0.513	256	5376	21	0
	1.387	256	3328	12	0
2K Byte	0.016	512	61440	120	0
	0.051	512	33792	66	0
	0.097	512	26112	51	0
	0.513	512	9216	18	0
	1.387	512	5632	11	0
4K Byte	0.016	1024	132096	129	0
	0.051	1024	61440	60	0
	0.097	1024	39936	39	0
	0.513	1024	17408	17	2.43
	1.387	1024	10240	10	0
NOC Design [11]					
512 Byte	0.037	64	4864	76	0
	0.072	64	3392	53	0
1K Byte	0.037	128	9088	71	0
	0.072	128	7552	59	0
2K Byte	0.037	256	16384	64	0
	0.072	256	13568	53	0
4K Byte	0.037	512	34816	69	0
	0.072	512	24064	47	0

information so that successive clock cycles are captured depending on the compression ratio.

Trace buffers are usually located at multiple locations in a chip [2]. The proposed debug hardware can be added to each trace buffer and obtain debug information at various sites. Different locations may capture different system misbehavior, hence, based on the error rates from different locations, each debug module can be independently activated at different cycles in the same debug session to increase the debugging capability. For example, assuming two debug modules are placed, debug module 1 can start executing 2-D compaction from cycle 100 and debug module 2 can perform 2-D compaction from cycle 1000. Since the hardware cost is relatively small and the $Mode\_Ctrl$ signal can disable the debug module, this 2-D compression debug methodology can be incorporated with other silicon debug techniques [2], [9], [20], [21], to enhance the post-silicon debug process as well.

VI. EXPERIMENTAL RESULTS

In this section, experimental results are presented for an ARM based processor design [14] and a NOC (network-on-chip) design on MPEG-4 Video Object Place Decoder [11]. Faults were randomly injected in circuits to produce misbehavior in the system and to generate erroneous data with a low error rate. By changing the injected faults, a set of experiments for different error rates were generated. A 32-bit data bus is used in an ARM-based design, and the NOC design uses a 64-bit data bus. The data bus was assumed to be observed by the trace buffer to perform the proposed debug technique.

Table I shows the results for different error rates using four different size (512, 1 K, 2 K, and 4 kB) trace buffers. The error rates in the data bus are shown in the second column. The error rates are computed as the number of data bus words (32-bit and 64-bit) with errors divided by the total number of data bus words and represented as a percentage. The third column shows the conventional observation window size in terms of the number of clock cycles worth of 32-bit/64-bit data words that could be stored in the trace buffer. If a 512 byte trace buffer is used, it only can capture 128 clock cycles worth of 32-bit words and 64 clock cycles worth of 64-bit words from the data bus. The fourth column shows the expanded observation window size that can be obtained using the proposed debugging method. The fifth column shows the expansion ratio which is computed as the expanded observation window size divided by the conventional observation window size. The proposed method shows that the observation window can be significantly expanded by orders of magnitude depending on the error rates. The last column shows the error aliasing percentage which is a critical index representing how accurately the suspect debug data can be captured. As can be seen from the results, the lower the error rate, the fewer the number of mismatching signatures from the MISR and cycling register. The 2-D compaction yields fewer intersections between both mismatching signatures and, hence, this results in storing fewer suspect clock cycles and significantly expands the observation window. The experimental results had

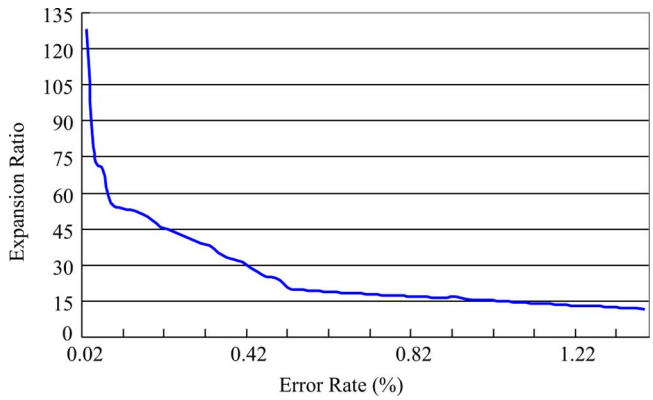


Fig. 10. Expansion ratio estimation with different error rate.

only one case where aliasing occurred and this resulted in a loss of 2.43% of the erroneous data words. As shown in Section IV, the aliasing probability depends on the bit error rate and the number of data words compacted in a compactor. With a given error rate in the system, the aliasing probability can always be reduced by using a less aggressive expanded observation window size. The aliasing probability of the proposed debugging method is negligible, however, the suspect clock cycles can be captured more accurately with less observation window size.

Fig. 10 shows a trace buffer expansion ratio estimation with different error rate. As graph shows, the expansion ratio increases as the error rate goes down. If error data is sparsely distributed, the amount of error-free data that needs to be captured increases because of the cycle tag bit compression. If the error data is clustered in the debug data, the expansion ratio will increase. Because the debug data captured by cycle tag bit compressions is likely to contain clustered errors, the number of captured error-free data will be smaller than the sparsely distributed error data case.

As discussed in Section III, the maximum possible expanded observation window size is equal to the trace buffer size divided by the error rate. The observation window size is estimated using the rough error rate by single parity information. In this observation window, the data with odd bit errors can be captured in a trace buffer. The trace buffer size is not considered when estimating the maximum possible expanded observation window size.

The expanded observation window size actually achieved with the proposed method is considerably less than the estimation in Section III. There are two reasons for this. One is that the 2-D compaction generally yields more suspects from the mismatching signature intersections than the actual erroneous clock cycles as shown an example in Fig. 5. The other reason is that the cycle tag bits may need to be compressed to increase the observation window which reduces the suspect resolution thereby increasing the number of clock cycles that need to be captured. Because the maximum expanded observation window size is not achievable, one way to reduce the search space for the 2-D compaction would be to compute a tighter upper bound on the expanded observation window size. This can be done by estimating the number of 2-D signature intersections and the amount of tag bit compression based on the trace buffer size and the estimated error rate. Using this information, a tighter upper bound on the expanded window size can be computed as follows:

$$window\_size \leq \frac{buffer\_size}{error\_rate * ANI * cycle\ tag\ bit\ group\ size}$$

where ANI denotes the average number of intersections and cycle tag bit group size represents the number of original cycle tag bits that need to be compressed together as discussed in Section IV. This tighter upper bound on the expanded window size can be used to determine when to begin the 2-D compaction.

VII. Conclusion

The key in the post-silicon debug process is to maximize observability of internal signals. The experimental results indicate that the proposed debug methodology in this paper only uses three debug sessions to expand the observation window for a trace buffer by one to two orders of magnitude via the selective debug data capture based on 2-D compaction. This provides much greater visibility of the real-time at-speed system operation.

The proposed methodology is compatible with other debug techniques [2], [9], [20], [21] and trace buffer compression techniques [4]. Moreover, it can also be applied even when a trace buffer is only triggered during certain events which may not necessarily be in consecutive clock cycles. From the debug modules viewpoint, the stream of data that is being observed can be relative to only the clock cycles when the trace buffer would normally be triggered. The expanded observation window in this case would be expanded only across the clock cycles when the trace buffer would normally be triggered.

The proposed method can be applied to a selected part of a design such as newly implemented and unverified modules that require more debugging effort. It should also be noted that if a design contains distributed multiple trace buffers, the proposed methodology could be concurrently or independently applied to all the trace buffers. However, the total number of debug sessions would still be three regardless of how many trace buffer observations windows are being expanded. This helps to isolate the bug location and to narrow down the error cycles, hence, this will speed up the post-silicon process.

References

[1] M. Abramovici and Y.-C. Hsu, "A new approach to silicon debug," presented at the Int. Silicon Debug Diagnosis Workshop (SDD), Austin, TX, 2005.

[2] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-Debug infrastructure for SoCs," in *Proc. Design Autom. Conf.*, 2006, pp. 7–12.

[3] E. Anis and N. Nicolici, "On using lossless compression of debug data in embedded logic analysis," in *Proc. IEEE Int. Test Conf.*, 2007, pp. 1–10.

[4] E. Anis and N. Nicolici, "Low cost debug architecture using lossy compression for silicon debug," *Proc. Design, Autom., Test Euro.*, pp. 1–6, 2007.

[5] D. Van Campenhout, H. Al-Asaad, J. P. Hayes, T. Mudge, and R. B. Brown, "High-level design verification of microprocessors via error modeling," *ACM Trans. Design Autom. Electron. Syst.*, vol. 3, no. 4, pp. 581–599, 1998.

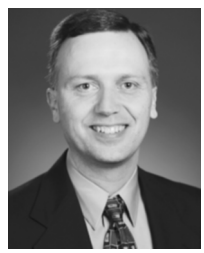
[6] G. Parthasarathy, M. K. Iyer, T. Feng, L.-C. Wang, K.-T. Cheng, and M. S. Abadir, "Combining ATPG and symbolic simulation for efficient validation of embedded array systems," in *Proc. IEEE Int. Test Conf.*, 2002, pp. 203–212.

[7] H. Fang, Z. Wang, X. Gu, and K. Chakrabarty, "Mimicking of functional state space with structural tests for the diagnosis of board-level functional failures," in *Proc. IEEE Asian Test Symp.*, 2010, pp. 421–428.
[8] H. Fang, Z. Wang, X. Gu, and K. Chakrabarty, "Deterministic test for the reproduction and detection of board-level functional failures," in *Proc. IEEE/ACM Asia South Pacific Design Autom. Conf.*, 2011, pp. 491–496.
[9] A. Hopkins and K. McDonald-Maier, "Debug support for complex systems on-Chip: A review," *IEEE Proc. Comput. Digit. Techn.*, vol. 153, no. 4, pp. 197–207, Jul. 2006.
[10] Y.-C. Hsu, F. Tsai, W. Jong, and Y.-T. Chang, "Visibility enhancement for silicon debug," in *Proc. Design Autom. Conf.*, 2006, pp. 13–18.
[11] W. Jang, D. Ding, and D. Pan, "A voltage-frequency island aware energy optimization framework for networks-on-Chip," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2008, pp. 264–269.
[12] H. F. Ko and N. Nicolici, "Automated trace signals identification and state restoration for improving observability in post-silicon validation," *Proc. Design, Autom., Test Euro.*, pp. 1298–1303, 2008.
[13] M. N. Velev, "Collection of high-level microprocessor bugs from formal verification of pipelined and superscalar designs," in *Proc. IEEE Int. Test Conf.*, 2003, pp. 138–147.
[14] J. Shen and J. A. Abraham, "Verification of processor microarchitectures," in *Proc. IEEE VLSI Test Symp.*, 1999, pp. 189–194.
[15] S. Hacker, *MP3: The Definitive Guide*. Sebastopol, CA: O'Reilly & Associates, Inc., 2000.
[16] Semiconductor Industry Association, "The International Technology Roadmap for Semiconductors," 2005.
[17] L.-T. Wang, C. E. Stroud, and N. A. Touba, *System-on-Chip Test Architectures*. Boston, MA: Morgan Kaufmann, 2008.
[18] F.-C. Yang, C.-L. Chiang, and I.-J. Huang, "A reverse-encoding-based on-chip AHB bus tracer for efficient circular buffer utilization," in *Proc. IEEE/ACM Asia South Pacific Design Autom. Conf.*, 2009, pp. 721–726.
[19] J.-S. Yang and N. A. Touba, "Expanding trace buffer observation window for in-system silicon debug through selective capture," in *Proc. IEEE VLSI Test Symp.*, 2008, pp. 345–351.
[20] B. Vermeulen, S. Oostdijk, and F. Bouwman, "Test and debug strategy of the PNX8525 NexperiaTM digital video platform system chip," in *Proc. IEEE Int. Test Conf.*, 2001, pp. 121–130.
[21] B. Vermeulen, T. Waayers, and S. K. Goel, "Core-based scan architecture for silicon debug," in *Proc. IEEE Int. Test Conf.*, 2002, pp. 638–647.

Joon-Sung Yang (S'05–M'09) received the B.S. degree from Yonsei University, Seoul, Korea, in 2003, and the M.S. and Ph.D. degrees from the University of Texas at Austin, Austin, in 2007 and 2009, respectively, all in electrical and computer engineering.

He is currently with Intel Corporation, Austin, TX. His research interests are VLSI testing, silicon debug and nanometer scale test and design methodologies.

Dr. Yang was a recipient of Korea Science and Engineering Foundation (KOSEF) Scholarship in 2005 and the Best Paper Award at the 2008 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems.



Nur A. Touba (SM'05–F'09) received the B.S. degree from the University of Minnesota, Minneapolis, in 1990, and the M.S. and Ph.D. degrees from Stanford University, Stanford, CA, in 1991 and 1996, respectively, all in electrical engineering.

He is currently a Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin.

Dr. Touba was a recipient of the National Science Foundation Early Faulty CAREER Award in 1997, the Best Paper Award at the 2001 VLSI Test Symposium, and the 2008 Defect and Fault Tolerance Symposium. He served as program chair for the 2008 International Test Conference and general chair for the 2007 Defect and Fault Tolerance Symposium. He currently serves on the program committee for the Design Automation and Test in Europe Conference, International On-Line Test Symposium, European Test Symposium, Asian Test Symposium, and Defect and Fault Tolerance Symposium.