Expanding Trace Buffer Observation Window for In-System Silicon Debug through Selective Capture

Joon-Sung Yang and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712

Abstract

Trace buffers are commonly used to capture data during in-system silicon debug. This paper 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. A three pass methodology is proposed. During the first pass, the rough error rate is measured, in the second pass, a set of suspect clock cycles where errors may be present is determined, and then in the third pass, the trace buffer captures only during the suspect clock cycles. In this manner, the effective observation window of the trace buffer can be expanded significantly, by up to orders of magnitude. This greatly increases the effectiveness of a given size trace buffer and can rapidly speed up the debug process. The suspect clock cycles are determined through a two dimensional (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 generates a small set of remaining suspect clock cycles for which the trace buffer needs to capture data. Experimental results indicate very significant increases in the effective observation window for a trace buffer can be obtained.

1. Introduction

Post-silicon debug is a major time consuming challenge that has significant impact on the development cycle of a new chip. The most difficult aspect is in-system at-speed debug where there is a need to extract data while the system is running. Trace buffers are commonly used to capture data from a limited number of signals during in-system debug [Abramovici 06], [Hopkins 06]. 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 can only store a limited amount of data in one session.

Some techniques have been proposed to compress the data stored in the trace buffer to increase its effectiveness. As suggested in [Anis 07a], 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 [Abramovici 05] and [Hsu 06], techniques are proposed for reconstructing the values of more internal signals than are captured each clock cycle in the trace buffer and hence these techniques expand the effective width of the observation window, but not its depth.

In [Anis 07a], lossless compression methods based on dictionary coding implemented with content-addressable memory were investigated for compressing the data stored in a trace buffer. This approach can expand the depth of the observation window as well. Results in [Anis 07a] for MP3 data show that the observation window can be increased up to 3.45 times larger. However, the amount of compression provided by dictionary coding varies greatly depending on how correlated the data is. While the amount of compression is modest, a nice feature of the method in [Anis 07a] is that it is a one pass scheme which does not require re-running the debug session and hence is useful for debugging non-deterministic behavior that is not repeatable.

If the behavior is deterministic and repeatable, then the method in [Anis 07b] which requires re-running the debug session many times can be used. This approach 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 progressively zooms in on clock cycles in which errors occur. When the size of the current search range becomes small enough to fit 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 a nice and effective idea for accelerating debug methods based on binary search, but it may not be a suitable replacement for more conventional applications of trace buffers because it can require a large number of debug sessions.

1093-0167/08 $25.00 © 2008 IEEE
DOI 10.1109/VTS.2008.41

In this paper, a new method for expanding the depth of the observation window for a trace buffer is proposed which requires only 3 debug sessions. It can expand the depth of the 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 two dimensional (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.

This paper is organized as follows. Sec. 2 gives an overview of the proposed scheme. Sec. 3 discusses the three pass debug procedure in detail. Sec. 4 describes the hardware architecture of the proposed scheme. Experimental results are shown in Sec. 5 and conclusions are given in Sec. 6.

2. 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". In the first debug session, the *error rate* (i.e., data word errors per clock cycle) is estimated using lossy compression. Based on the estimated error rate, the maximum expanded observation window size is computed as follows:

$$window\_size \leq buffer\_size / error\_rate$$

where *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 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, the 2-D compaction is activated during the clock cycles in the maximum expanded observation window range 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 intersecting the information obtained from them to identify the suspects. The MISR is used to generate k signatures where each signature compacts *window_size/k* consecutive data words. A cycling register of length m compacts the data words such that every m-th data word is XORed together in each signature. The cycling register indicates whether erroneous data exists in each modulo m set of data words. An erroneous data word produces a corresponding erroneous MISR signature and erroneous cycling register signature. Faulty signatures from both compactors (MISR and cycling register) are used to identify the suspect clock cycles by finding the intersections of the signatures.

In the third debug session, the trace buffer captures data during the suspect clock cycles. 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. As will be shown, the probability of aliasing is extremely small for low error rates (i.e., error rates below 1%). For in-system debug, where the part has already passed a manufacturing test, error rates are typically low as errors occur only at certain corner cases under at-speed operation of the system. The proposed scheme exploits this low error rate property allowing selective capture to achieve significant observation window size expansion which greatly enhances visibility.

3. Details of the Three Debug Sessions

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

3.1 Session 1 - Estimating Expanded Observation Window Size

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 trace buffer 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 (*2bit * 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 Sec. 2. 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 Sec. 5. The maximum window size as used as the starting point for 2-D compaction in the second session.

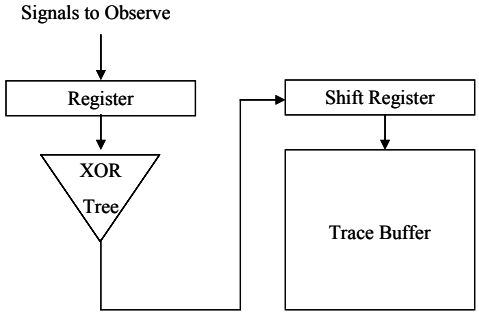

Figure 1. Session 1: Parity Generation

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.

3.2 Session 2 - Determining Suspects

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 session 1. 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 m-th data word.

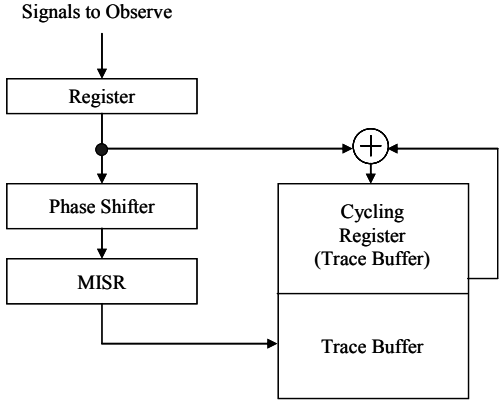

Figure 2. Session 2: 2-D Compaction

3.2.1 2-D Compaction

Fig. 2 illustrates the operation of the 2-D compactor. 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 15 clock cycles of data words with $k=5$ and $m=5$. A MISR signature is generated every *(window_size/k)=3* 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 though 3 which is expressed as $MS_1 = \{C_1, C_2, C_3\}$. The cycling register compacts every m-th data word. The first signature in the cycling register in Fig. 3 is denoted as CR_1 and is expressed as $\{C_1, C_6, C_{11}\}$ since $m=5$.

If C_{13} is faulty, then MS_5 and CR_3 will mismatch with the fault-free signatures assuming there is no aliasing. The mismatching signatures, MS_5 and CR_3 are highlighted in gray in Fig 3. 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

$$P(Aliasing) \approx 1 - \{1 - (C_2^{NECR})(Bit\ Error\ Rate)^2 (1 - Bit\ Error\ Rate)^{NECR-2}\}^{WORD\ SIZE}$$

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.

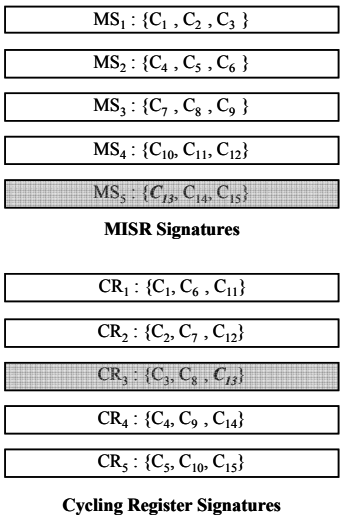

Figure 3. Example of 2-D Compaction using MISR with $k=5$ and Cycling Register with $m=5$ for 15 clock cycles

3.2.2 Tag Data Generation

As shown in Fig. 3, erroneous data in C_{13} corrupts signatures in the MISR and cycling register. By finding the intersection of the mismatching signatures, the suspect clock cycles can be identified. In Fig. 3, intersecting MS_5 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 "tag bits", one for each clock cycle in the observation window. Each suspect clock cycle is indicated by setting its corresponding tag bit to 1 and each vindicated clock cycle is denoted by setting its corresponding tag bit to 0. For the example in Fig. 3, the tag bit for C_{13} is set to *1,* and *0* is assigned to the rest of the clock cycles. In this case, the 15 bit tag information is generated as $0000000000001_{(C13)}00$. 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 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.

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

To avoid this limitation, it may be necessary to compress the 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 tag bits can be initially computed one per clock cycle, and then successive tag bits can be grouped and compressed into a single bit. One compressed bit is used to represent the whole group. A compressed tag bit has value *0* when there are no *1s* 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.

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

currentMR = 0; currentGMS = 0;
tagbit[numData] = 0;
while( currentMR < lastMR ){
   List all the element MR[currentMR];
   if( equality(MR[currentMR], GS[currentGMS]) ){
      while( !visited all the element ){
         tagbit[element] = 0;
         next_element;
      }
   }
   else{
      while( !visited all the element ) {
         if( equality(correspondingCR, GCR) )
               tagbit[element] = 0;
         else   tagbit[element] = 1;
       next_element;
       }
   }
   currentMR++; currentGMS++;
}
```

Figure 4. Tag Data Generation Algorithm

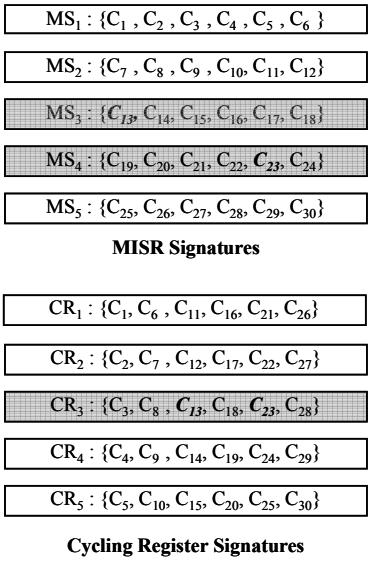

Figure 5. Example of 2-D Compaction using MISR with *k=5* and Cycling Register with *m=5* for 30 clock cycles

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 tag bits into one compressed tag bit, then the 30 bit tag data is compressed into the following 15-bits *000000101001000* which is also illustrated in Fig. 8.

3.3 Session 3 - Capturing Suspect Clock Cycles

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.

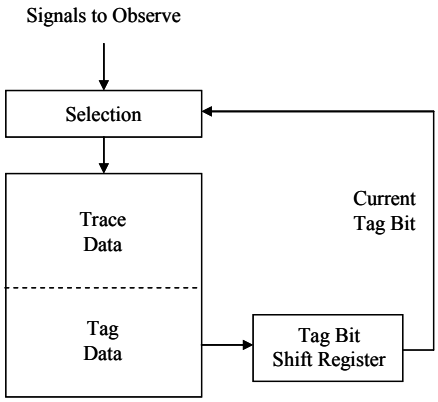

Figure 6. Session 3: Selective Capture with Tag Bit

In the example in Fig. 3, C_{13} is identified as a suspect and the tag bits were generated as $0000000000001_{(C13)}00$. Fig. 7 shows the trace buffer after the third session. For the example in Fig. 5, the 30 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 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. Expanding the trace buffer observation window gives visibility over wider range of data. Hence the proposed approach reduces the overall silicon debug time.

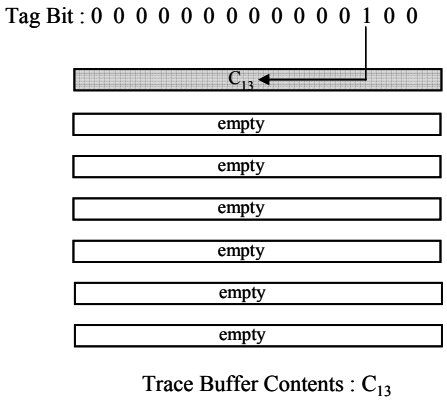

Figure 7. Data in Trace Buffer for 15 Tag Bits from Example in Fig. 3

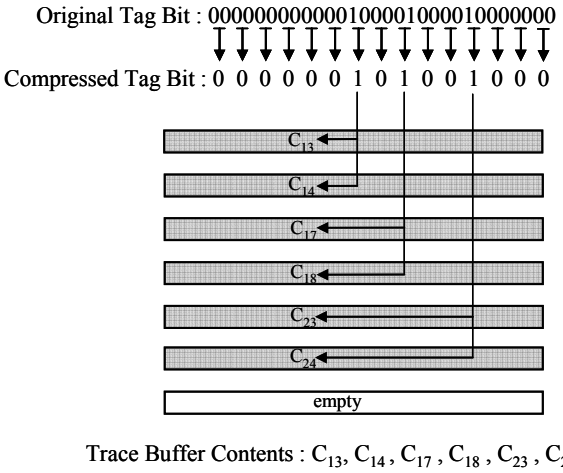

Figure 8. Data in Trace Buffer for 15 Compressed Tag Bits from Example in Fig. 5

4. Hardware Architecture of Debug Module

The hardware architecture for a proposed debug module is shown in Fig. 9. To perform the operations discussed in Sec. 3, the debug module activates different functional blocks using the *Mode_Ctrl* signals.

In session 1, the *Mode_Ctrl* signals select the *phase 1 block* in Fig. 9 which is the parity generation mode. 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.

In session 2, the 2-D compactors in the *phase 2 block*, are activated by the *Mode_Ctrl* signals. The MISR and cycling register signatures are generated and stored in the

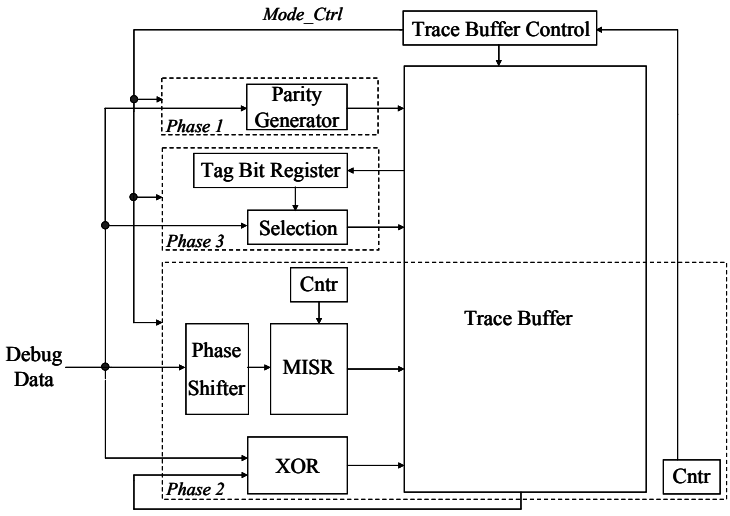

Figure 9. Hardware Architecture of Proposed Debug Module

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 cycling register signatures and the other half is used to store MISR signatures.

In session 3, the *Mode_Ctrl* signals activate the selection logic in the *phase 3 block* which selectively captures the debug data based on the tag information. A 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*.

5. Experimental Results

In this section, experimental results are presented for an ARM based processor design [Shen 99]. Faults were injected into a processor design to generate erroneous data. The 32-bit data bus was assumed to be observed by the trace buffer. By changing the injected faults, a set of experiments for different low error rates were generated. Table 1 shows the results for different error rates using different size trace buffers. The first column shows the size of the trace buffer. The second column shows the error rate computed as the number of 32-bit data bus words with errors divided by the total number of 32-bit data bus words and expressed as a percentage. The third column shows the conventional observation window size in terms of the number of clock cycles worth of 32-bit data words that could be stored in the trace buffer. For example, a 512 byte trace buffer can only capture 128 clock cycles worth of 32-bit words from the data bus. The expanded observation window size that can be obtained using the proposed method is shown in the fourth column. The fifth column shows the expansion ratio which is computed as the expanded observation window size divided by the conventional observation window size. The last column shows the error aliasing percentage.

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, and hence the 2-D compaction yields fewer suspects resulting in greater observation window expansion. The experimental results had only one case where aliasing occurred and this resulted in a loss of 2.4% of the erroneous data words. Note the aliasing probability can always be reduced by using a less aggressive expanded observation window size.

As discussed in Sec. 2, the maximum possible expanded observation window size is equal to the trace buffer size divided by the error rate since the trace buffer must store all the erroneous data words. The expanded observation window size actually achieved with the proposed method is considerably less than that. There are two reasons for this. One is that the 2-D compaction generally yields more suspects than the actual erroneous clock cycles, and the other is that the tag bits may need to be compressed 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:

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

Table 1. 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
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	2.4
	1.387	1024	10240	10	0

where *ANI* denotes the average number of intersections and *tag bit group size* represents the number of original tag bits that need to be compressed together (as discussed in Sec. 3.2.2). This tighter upper bound on the expanded window size can be used to determine when to begin the 2-D compaction.

6. Conclusions

The experimental results indicate that the methodology proposed in this paper can use 3 debug sessions to expand the observation window for a trace buffer by one to two orders of magnitude. This provides much greater visibility of the real-time at-speed operation during in-system silicon debug. The proposed methodology is compatible with other trace buffer compression techniques. 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 module's 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.

It should also be noted that if a design contains multiple trace buffers, the proposed methodology could be concurrently applied to all the trace buffers. So the total number of debug sessions would still be 3 regardless of how many trace buffer observations windows are being expanded.

References

[Abramovici 05] Abramovici, M., and Y.-C. Hsu, "A New Approach to Silicon Debug," *Proc. of Int. Silicon Debug and Diagnosis Workshop (SDD),* 2005.

[Abramovici 06] Abramovici, M., P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A Reconfigurable Design-for-Debug Infrastructure for SoCs," *Proc. of Design Automation Conference,* pp. 7-12, 2006.

[Anis 07a] Anis, E., and N. Nicolici, "On Using Lossless Compression of Debug Data in Embedded Logic Analysis," *Proc. of Int. Test Conference,* Paper 18.3, 2007.

[Anis 07b] Anis, E., and N. Nicolici, "Low Cost Debug Architecture using Lossy Compression for Silicon Debug," *Proc. of Design, Automation, and Test in Europe*, pp. 1-6, 2007.

[Hopkins 06] Hopkins, A., and K. McDonald-Maier, "Debug Support for Complex Systems on-Chip: A Review," *IEEE Proc. on Computers and Digital Techniques*, Vol 153, No. 4, pp. 197-207, Jul. 2006.

[Hsu 06] Hsu, Y.-C., F. Tsai, W. Jong and Y.-T. Chang, "Visibility Enhancement for Silicon Debug," *Proc. of Design Automation Conference*, pp. 13-18, 2006.

[Shen 99] Shen J., and Abraham, J. A., "Verification of Processor Microarchitectures," *Proc. of VLSI Test Symposium*, pp. 189-194, Apr. 1999.