International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 3, March - 2014

Dynamic Multi Resolution Tracer for on Chip Bus with Real Time Compression

Baderunnisa Begum[#1], Seema Deshmukh[*2]
[1]Mtech student (Vlsi and embedded design),VTU regional center Gulbarga,VTU belgaum university,India.
[2] prof. in E and CE department,PDA college of engg.,Gulbarga,India.

***Abstract*— This paper proposes a multiresolution AHB on-chip bus tracer named SYS-HMRBT (aHb multiresolution bus tracer) for debugging and monitoring of system on chip. The bus tracer captures the bus trace with different resolutions, with efficient built-in compression mechanisms, to meet a diverse range of needs. The users can switch the trace resolution dynamically so that appropriate resolution levels can be applied to different segments of the trace. SYS-HMRBT supports tracing after/before an event triggering, named post-triggering trace/pre-triggering trace also referred to as forward/backward trace respectively. SYS-HMRBT runs at 500 MHz and costs 42 K gates in TSMC 0.13- m technology, indicating that it is capable of real time tracing and is very small in modern SoCs. The bus tracer can achieves very good compression ratios of 79%–96%, depending on the selected resolution mode and abstraction level.**

Keywords—*AHB,AMBA,compression,multiresolution,post trace,pretrace.*

I. INTRODUCTION

Monitoring and debugging a highly integrated Systemon- Chip (SoC) has becomes a major challenge during recent years due to the lack of visibility into its internal status, there might not be sufficient spare I/O pins to access into internal signals then a straight forward method to overcome these problem is to embed a bus tracer in soc to capture the bus signals and store the trace in an on chip storage such as trace memory which can be off loaded to outside world for future analysis purpose. There are two types of traces related to the target event, forward trace and backward trace.The forward/backward trace refers to the trace captured after/before the matched event, respectively and this feature provides a more flexible tarcing to focus on the interesting points. multi-resolution AHB on-chip bus tracer which is capable of capturing and compressing both forward and backward traces in a circular buffer. . Unfortunately, the size of the bus trace grows rapidly. For example, to capture AMBA AHB 2.0 [1] bus signals running at 200 MHz, the trace grows at 2 to 3 GB/s. Therefore, it is highly desirable to compress the trace on the fly in order to reduce the trace size. However, simply capturing/compressing bus signals is not sufficient for SoC debugging and analysis, since the debugging/ analysis needs are versatile: some designers need all signals at cycle-level, while some others only care about the transactions. For the latter case, tracing all signals at cycle-level wastes a lot of trace memory. Thus, there must be a way to capture traces at different abstraction levels based on the specific debugging/analysis need. This paper presents a real-time multi-resolution AHB on-chip bus tracer, named SYS-HMRBT (aHb multiresolution bus tracer). The bus tracer adopts three trace compression mechanisms to achieve high trace compression ratio. It supports *multiresolution tracing* by capturing traces at different timing and signal abstraction levels. In addition, it provides the *dynamic mode change* feature to allow users to switch the resolution on-the-fly for different portions of the trace to match specific debugging/analysis needs. Given a trace memory of fixed size, the user can trade off between the granularity and trace length to make the most use of the trace memory.

II. RELATED WORK

Since the huge trace size limits the trace depth in a trace memory, there are hardware approaches to compress the traces. The approaches can be divided into lossy and lossless trace compression. The lossy trace compression approach achieves high compression ratio by sacrificing the accuracy; the original signals cannot be reconstructed from the trace. The purpose of this approach is to identify if a problem occurs. Anis and Nicolici [2] use the multiple input signature register (MISR) to perform lossy compression. The results are stored in a trace memory and compared with the golden patterns to locate the range of the erroneous signals. The locating needs rerunning the system several times with finer and finer resolution until the size of the search range can fit in the trace memory. Such approach is suitable for deterministic and repeatable system behaviors. However, for a complex SoC with multiple independent IPs, the on-chip bus activities are usually not deterministic and repeatable. Therefore, lossless compression approaches are more appropriate for real time on-chip bus tracing. Existing on-chip bus tracers mostly adopt lossless compression approaches. ARM provides the AMBA AHB trace macrocell (HTM) [3] that is capable of tracing AHB bus signals, including the instruction address, data address, and control signals. The instruction address and control signals are compressed with a slice compression approach .On the other hand, the data address is recorded by simply removing the leading zeros. The HTM supports a limited level of trace abstraction by removing bus signals that are in IDLE or BUSY state. The AMBA navigator [4] traces all AHB bus signals without compression. In the bus transfer mode, it also has a limited level of trace abstraction by removing bus signals which are in IDLE,BUSY,or non-ready state[13]. The AHB TRACE in GRLIB IP library [5] captures the AMBA AHB

signals in the uncompressed form. In addition, it does not have trace abstraction ability. There are many research works related to the bus signal compression. We characterize the bus signals into three categories:program address, data address/data and control signals.We then review appropriate compression techniques for each category. For **program addresses**, since they are mostly sequential, a straightforward way is to discard the continuous instruction addresses and retain only the discontinuous ones, so called branch/ target filtering. This approach has been used in some commercial tracers, such as the TC1775 trace module in TriCore [6] and ARM's Embedded Trace Macrocell (ETM) [7]. The hardware overhead of these works is usually small since the filtering mechanism is simple to be implemented in hardware. The effectiveness of these techniques, however, is mainly limited by the average basic block size, which is roughly around four or five instructions per basic block [7], [8]. Other technique such as the slice compression approach [3] targets at the spatial locality of the program address. This approach partitions a binary data into several slices and then records all the slices of the first data and then only part of the slices of the succeeding data that are different from the corresponding slices of the previous one (usually the lower bit positions of the data). For **data address/value**, the most popular method is the differential approach which records the difference between consecutive data. Since the difference usually could be represented with less number of bits than the original value, the information size is reduced. Hopkins and Mc- Donald–Maier showed that the differential method can reduce the data address and the data value by about 40% and 14%, respectively [9]. For **control signals**, ARM HTM [3] encodes them with the slice compression approach: the control signal is recorded only when the value changes. As mentioned, all signals at the cycle-accurate- level does not always meet the compressing debugging needs. As SoCs become more complex, the transaction-level debugging becomes increasingly important, since it helps designers focus on the functional behaviors, instead of interpreting complex signals. Tabbara and Hashmi [10] propose the solution, the transaction-level debugging can provide software and hardware designers a common abstraction level to diagnose bugs collaboratively, and thus, help focus problems quickly. Both works indicate that the transaction-level debugging is a must in SoC development. Motivated by the related works, our bus tracer combines abstraction and compression techniques in a more aggressive way. The goal is to provide better compression quality and multiple resolution traces to transaction-level SoC modeling and debugging method. The proposed transactors, attaching to the on-chip bus, recognize/monitor signals and abstract the signals into transactions. The transactions, bridging the gap between algorithm-level and the signal-level, enable easy design exploration/debugging/monitoring. Vermeulen *et al.* [11] propose a communication-centric intrusive debugging method based on the transaction level. They point out that the traditional hardware and software debugging cannot work collaboratively, since the software debugging is at the functionallevel and the hardware debugging is at the signal level. As a meet the complex SoC debugging needs. For example, our bus tracer can provides traces at cycle-level and transaction-level to support versatile debugging needs. Besides, features such as the dynamic mode change and bidirectional traces are also introduced to enhance the debugging flexibility.

III. TRACE DIRECTION AND TRACE GRANULARITY

The multiresolution trace mode and the pre/post-T tracing are two important features for effective SoC debugging and monitoring. They are discussed in this section in terms of trace granularity and trace direction.

A. Trace Granularity—Multiresolution Trace

This section first introduces the definitions of the abstraction level. Then, it discusses the application for each abstraction mode.

1) Timing and Signal Abstraction Definition: The abstraction level is in two dimensions: timing abstraction and signal abstraction. At the timing dimension, it has two abstraction levels, which are the cycle level and transaction level, as shown in Table I. The cycle level captures the signals at every cycle. The transaction level records the signals only when their value change (event triggering). For example, since the bus read/write control signals do not change during a successful transfer, the tracer only records this signal at the first and last cycles of that transfer.
However, if the signal changes its value cycle-by-cycle, the transaction-level trace is similar to the cycle-level trace. At the signal dimension, first, we group the AHB bus signals into four categories: program address, data address/value, access control signals (ACS), and protocol control signals (PCS). Then, we define three abstraction levels for those signals. As shown in Table II, they are full signal level, bus state level, and master operation level. The full signal level captures all bus signals. The bus state level further abstracts the PCS by encoding they as states according to the bus-state-machine (BSM)[13].

TABLE I
TIMING ABSTRACTION

	Cycle level	Transaction level
Time granularity	Cycle accurate	Event triggering

Table II

COMPRESSION PHASES FOR DIFFERENT SIGNAL TYPES

	Action		
	Phase 1	Phase 2	Phase 3
P. addr.	Branch/Target filter	Dict. based	Slicing
D. addr./value	Differential	N/A	N/A
ACS	Dict. based	N/A	N/A
PCS	Dict. based	N/A	N/A

B. B.Trace Direction: Pre-T/Post-T Trace

Supporting both trace directions provides the flexible debugging strategies. As Fig.1 shows, the *post-T* trace captures signals after a triggering event, while the *pre-T* trace captures signals before the triggering event. The post-T trace is usually used to observe signals after a known event. The pre-T trace is useful for diagnosing the causes of unexpected errors by capturing the signals before the errors. The mechanisms of the pre-T trace and the post-T trace are different. The Post-T trace is simpler since the start time and the stop time are known. It is activated when the target event is matched and is turned off when the trace buffer is full. On the other hand, the stop time of the pre-T trace is unpredictable. The solution is to start tracing as soon as system reset (or some other turning-on event). When the trace buffer is full, the new trace data wrap around the trace buffer, which means the oldest data are sacrificed for the newest ones. Wrapping around the trace buffer causes a problem when the the trace needs to be compressed. Typical lossless compression algorithms work by storing some initial (previous) states of the trace first and then calculate the relationship between the current data and the previous states. Since the size of the relationship is smaller than the data size, e.g., the difference, it saves spaces.The initial state of the trace, which is stored at the head of the buffer, will be destroyed if the trace is wrapped around, and thus making the rest of the trace not decompressable. To solve the above problem, this paper presents a real time multi-resolution AHB on-chip bus tracer which is capable of capturing and compressing both forward and backward traces in a circular buffer.

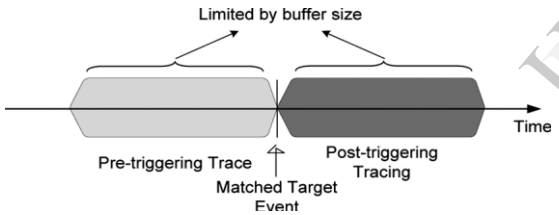

Fig.1. Pre-T trace and Post-trae with respect to a matched event.

1)Periodical Triggering Concept: : To solve the problem, we adopt a periodical triggering technique. The concept is to divide the entire trace into several independent small traces, as shown in Fig.2. Since every trace has its own initial state, destroying the initial state of one trace does not affect other traces. This technique can be accomplished by periodically triggering[14] a new trace. Therefore, it can be easily accomplished by the existing trace compression engines with minor modification to their control circuitry. To support periodical triggering, the circular buffer is partitioned into segments, with each segment storing one small trace. The damage due to wrapping around is thus limited to one segment. Therefore, if the total number of the segments is , after the first wrapping-around occurs (the buffer becomes full), there will be segments containing valid traces, and there will be one segment where the old trace is overwritten by the new trace. Assuming the trace size is random, half of this segment will contain valid new trace on the average. Therefore, the averaged total buffer utilization is (i.e.,). As a result, the more the segments, the better the buffer utilization ratio. On the other hand, the more segments, the less the effective trace depth since there are more initial data, which are not compressed, populating the buffer. Therefore, tradeoff has to be made among the segment number, the buffer utilization ratio and the effective trace depth.

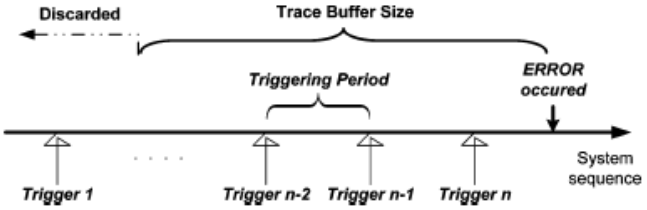

Fig.2. Periodic triggering concept.

*2) Circular Buffer Management/Management of Trace Headers:*To decompress those traces (segments) in a circular buffer, we must know where the tracesare in the circular buffer and which one is the oldest trace. This task is not straight forward because the trace lengths are variable due to the nature of compression. Therefore, a header position table is used to keep track the location of each trace, as shown in Fig.3 . This table consists of sixteen header position registers,which allows us to support up to 16 segments in the buffer. In addition, there is an oldest header register to point to the oldest trace. This helps the decompression software to identify the location of the oldest trace so that it can decompress the trace in time order. For example, in Fig. 3(a), when the circular buffer is not full, this register points to the 1st trace. After, in Fig. 3(b),the buffer is full and trace 17th wraps around the circular buffer the first trace is damaged because the initial state is over written. Then, the oldest header register is adjusted to point to the second trace. If necessary, more header position registers can be allocated to support more segments in larger buffer.

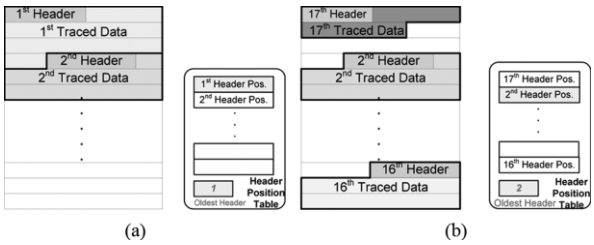

Fig.3. Trace buffer and assistant header position table.(a)the wrapping doesnot occur (b)the wrapping around occurs

3).Ping-pong Organization of Dual Forward: Compression Engines when the forward compression engine receives the *Trigger* signal, it takes at least one cycle (or may be more if the circuit cost is to be reduced) to reset all its internal data structure (e.g., the dictionary table) such that a new trace can be produced in the next cycle. Unfortunately, the bus signals will be lost during the reset cycle. To avoid this problem, we

could use two forward compression engines and operate them in a ping-pong fashion, as shown in Fig.4. When *Trigger* signal is asserted, the next engine begins to accept the bus signals for compression. On the other hand, the current engine stops receiving new bus signals but continues to operate a few cycles to clean up the remaining trace in its pipeline, and finally it takes a few cycles to reset its internal data structure. When the next *Trigger* signal comes, the roles of the two engines are exchanged There are two timings we must consider carefully. The *first* timing is the choice of active engines. When trigger signal is asserted, we will change the active compression hardware between engine 0 and engine 1 immediately. We use *CM0 Enable* and *CM1 Enable* to select the active compression engine. The *second* timing is when to clear the current engine and switch the multiplexor for the two engines. Our engine consists of four pipeline stages; there are active trace remaining in the pipeline when triggering occurs. Hence we assert *CM0 rst/CM1 rst* and *Data Select* signal four cycles after *Trigger* signal being asserted to reset the engine and switch the multiplexer to select the proper engine for output.

IV. BUS ARCHITECTURE

This section presents the architecture of our bus tracer. We provide an overview of the architecture for the post-T trace. Fig.4 is the bus tracer overview. It mainly contains four parts: Event Generation Module, Abstraction Module, Compression Modules, and Packing Module. The Event Generation Module controls the start/stop time, the trace mode, and the trace depth of traces. This information is sent to the following modules. Based on the trace mode, the Abstraction Module abstracts the signals in both timing dimension and signal dimension. The abstracted data are further compressed by the Compression Module to reduce the data size. Finally, the compressed results are packed with proper headers and written to the trace memory by the Packing Module.

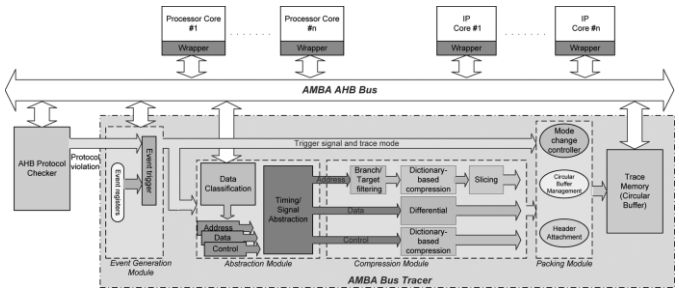

Fig.4 . Multiresolution bus tracer block diagram.

1)Event Generation Module: The Event Generation Module decides the starting and stopping of a trace and its trace mode. The module has configurable event registers which specify the triggering events on the bus and a corresponding matching circuit to compare the bus activity with the events specified in the event register. This module can also accept events from external modules. For example, we can connect an AHB bus protocol checker (HPChecker) [12] to the Event Generation Module, as shown in Fig. 4. to capture the bus protocol related trace. Fig. 5. is the format of an event register. It contains four parameters: the trigger conditions, the trace mode, the trace direction, and the trace depth. The trigger conditions can be any combination of the address value, the data value, and the control signal values. Each of the value has a mask field for enabling partial match. For each trigger condition, designers can assign a desired trace mode, e.g., Mode FC, Mode FT, etc., which allows the trace mode to be dynamically switched between events. The trace direction determines the pre-T/post-T trace. The trace depth field specifies the length of trace to be captured.

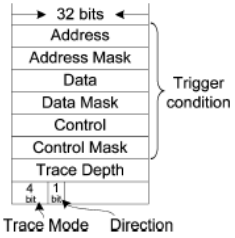

Fig.5. Event register.

2)Abstraction Module: The Abstraction Module monitors the AMBA bus and selects/filters signals based on the abstraction mode. The bus signals are classified into four groups. Then, depending on the abstraction mode, some signals are ignored, and some signals are reduced
to states.

Timing and Signal Abstraction : The abstraction level is in two dimensions: timing abstraction and signal abstraction. At the timing dimension, it has two abstraction levels, which are the cycle level and transaction level, as shown in Table I. The cycle level captures the signals at every cycle. The transaction level records the signals only when their value change (event triggering). For example, since the bus read/write control signals do not change during a successful transfer, the tracer only records this signal at the first and last cycles of that transfer. However, if the signal changes its value cycle-by-cycle, the transaction-level trace is similar to the cycle-level trace. At the signal dimension, first, we group the AHB bus signals into four categories: program address, data address/value, access control signals (ACS), and protocol control signals (PCS). Then, we define three abstraction levels for those signals. Combining the abstraction levels in the timing dimension and the signal dimension, we provide five modes in different granularities, as Fig. 6. shows. They are Mode FC (full signal, cycle level), Mode FT (full signal, transaction level), Mode BC (bus state, cycle level), Mode BT (bus state, transaction level), and Mode MT (master state, transaction level).

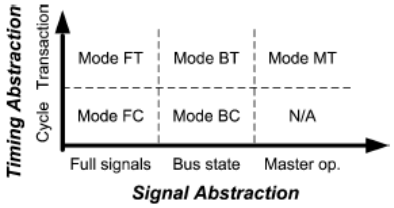

Fig.6. Multiresolution trace modes.

Applications of Abstraction Modes: At Mode FC, the tracer traces all bus signals cycle-by-cycle so that designers can observe the most detailed bus activities. This mode is very useful to diagnose the cause of error by looking at the detail signals. However, since the traced data size of this mode is huge, the trace depth is the shortest among the five modes. Fortunately, it is acceptable since designers using the cycle-level mode trace only focus on a short critical period.

At Mode FT, the tracer traces all signals only when their values are changed. In other words, this mode traces the untimed data transaction on the bus. Comparing to Mode FC, the timing granularity is abstracted. It is useful when designerswant to skim the behaviors of all signals instead of looking at them cycle-by-cycle. Another benefit of this mode is that the space can be saved without losing meaningful information. Thus, the trace depth increases.

At Mode BC, the tracer uses the BSM, such as NORMAL, IDLE, ERROR, and so on, to represent bus transfer activities in cycle accurate level. Comparing to Mode FC, although this mode still captures the signals cycle-by-cycle, the signal granularity is abstracted. Thus, designers can observe the bus handshaking states without analyzing the detail signals. The benefit is that designers can still observe bus states cycle-by-cycle to analyze the system performance.

At Mode BT, the tracer uses bus state to represent bus transfer activities in transaction level. The traced data is abstracted in both timing level and signal level; it is a combination of Mode BC of observing master behaviors is to realize the whole picture. Tracing master behaviors at cycle level is meaningless and can be replaced with Mode BC. Finally, the results are forwarded to the Compression Module for compression.

3)Compression Module: The main purpose of the Compression Module is to reduce the trace size. It accepts the signals from the abstraction module. To achieve real time compression, the Compression Module is pipelined to increase the performance. Every signal type has an appropriate compression method, as shown in Table II. The program address is compressed by a combination of the branch/target filtering, the dictionary- and Mode BT. In this mode, designers can easily understand the bus transactions without analyzing the signals at cycle level.

At Mode MT, the tracer only records the master behaviors, such as read, write, or burst transfer. It is the highest abstraction level. This feature is very suitable for analyzing the masters' transactions. The major difference compared with Mode BT is that this mode does not record the transfer handshaking activities and does not capture signals when the bus state[13] is IDLE, WAIT, and BUSY. Thus, designers can focus on WAIT, and BUSY. Thus, designers can focus on only the masters' transactions. Please note that there is no mode supporting master operation trace at cycle level, since the intension based compression, and the slicing. The data address and the data value are compressed by a combination of the differential and encoding methods. The ACS and PCS signals are compressed by the dictionary-based compression.

a)Compression Mechanism:

The Abstraction Module can reduce the trace size, the remaining trace volume is still very large. To reduce the size, the data compression approaches are necessary. Since the signal characteristics of the address value, the data value, and the control signals are quite different, we propose different compression approaches for them.

1) Program Address Compression: We divide the program address compression into three phases for the spatial locality and the temporal locality. Fig.7. shows the compression flow. There are three approaches: branch/target filter, dictionary based compression, and slicing.

2) Branch/Target Filtering: This technique aims at the spatial locality of the program address. Spatial locality exists since the program addresses are sequential mostly. Software programs (in assembly level) are composed by a number of basic blocks and the instructions in each basic block are sequential. Because of these characteristics, Branch/target filtering can records only the first instruction's address (Target) and the last instruction's address (Branch) of a basic block. The rest of the instructions are filtered since they are sequential and predictable.

3) Dictionary-Based Compression: To further reduce the size, we take the advantage of the temporal locality. Temporal locality exists since the basic blocks repeat frequently (loop structure), which implies the branch and target addresses after Phase 1 repeat frequently. Therefore, we can use the dictionary-based compression. The idea is to map the data to a table keeping frequently appeared data, and record the table index instead of the data to reduce size.

4) Slicing: The miss address can also be compressed with the Slicing approach. Because of the spatial locality, the basic blocks are often near each other, which means the high-order bits of branch/target addresses nearly have no change. Therefore, the concept of the Slicing is to reduce the data size by recording only the different digits of two consecutive miss addresses. To implement this concept in hardware, the address is partitioned into several slices of a equal size. The comparison between two consecutive miss addresses is at the slice level. For example, there are three address sequences: A(0001_0010_0000), B(0001_0010_0110), C(0001_0110_0110). At first, we record instruction A's full address. Next, since the upper two slices of address B are the same as that of the address A, only the least-significant slice is recorded. For address C, since the most significant slice is the same to that of the address B, only the lower two slices are recorded.

5) Data Address/Value Compression: Data address and data value tend to be irregular and random. Therefore, there is no effective compression approach for data address/value. Considering using minimal hardware resources to achieve a good compression ratio, we use a differential approach based on the subtraction.The differential module calculates the absolute difference value . Since the absolute difference between two data value may be small, we can neglect the leading zeros and use fewer digits to record it.Finally, the encoded datum is sent to the packing module .

6) Control Signal Compression: We classify the AHB control signals into two groups: access control signals (ACS) and

protocol control signals (PCS). ACS are signals about the data access aspect, such as read/write, transfer size, and burst operations. PCS are signals controlling the transfer behavior, such as master request, transfer type, arbitration, and transfer response. Control signals have two characteristics. First, the same combinations of the control signals repeat frequently, while other combinations happen rarely or never happen. The reason is that many combinations do not make sense in a SoC. It depends on the processor architecture, the cache architecture, and the memory type. Therefore, the IPs in a SoC tend to have only a few types of transfer despite the bus protocol allows for many transfer behaviors. Second, control signals change infrequently in a transaction. Because of these two characteristics, ACS/PCS are suitable for dictionary-based compression. The idea is to treat the signals in ACS/PCS as one group. Since the variations of transfer types are not much and transfer types repeat frequently, we can map them to the dictionary with frequently transfer types to reduce size. For example the original size of ACS is 15 bits.

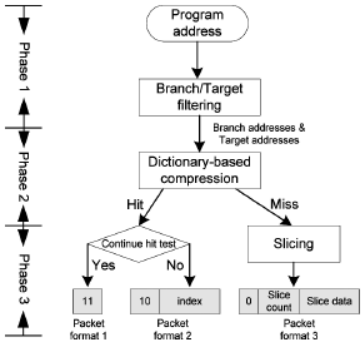

Fig.7. Program address compression flow and trace format.

4)Packing Module: The Packing Module is the last phase. It receives the compressed data from the compression module, processes them, and writes them to the trace memory. It is responsible for three jobs: packet management, circular buffer management, and mode change control. For packet management, since the compressed data length and type are variable, every compressed data needs a header for interpretation. Therefore, this step generates a proper header and attaches it to each compressed datum. In this paper, we call a compressed data with a header as a packet. Since the header generation takes time, to avoid long cycle time, the header generation is implemented in one pipeline stage. For circular buffer management, it manages the accesses to the trace memory. Since the size of a packet is variable but the data width of the trace memory is fixed, this module collects the trace data in a first-input, first-output (FIFO) buffer and outputs them to the trace memory until the data size in the FIFO buffer is equal/larger than the data width. If the tracing stops and the data size in the FIFO buffer is smaller than the data width, one additional cycle is required to output the remaining data to the trace memory. For mode change control, it manages the insertion of the special packet (called mode-change packet) that distinguishes the current mode from the previous mode. Dynamic mode change can be achieved by changing the mode in the abstraction module. Designers can achieve this by setting the desired trace mode in the event register. However, since the header of each packet does not include the mode information because of space reduction, the decompression software cannot tell how to decompress the packets. Therefore, there must be a mode-change packet, that indicates the trace mode, placing between two tracers belonging to two different modes. *Dynamic Mode Change:* Our bus tracer also supports dynamic mode change (DMC) feature. This feature allows designers to change the trace mode dynamically in real-time depicted in fig.8.

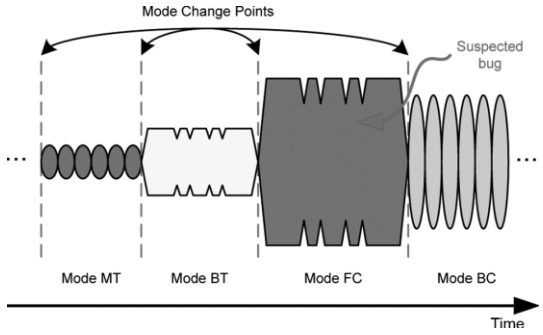

Fig.8. Monitoring process with dynamic mode change.

Multiresolution trace has two advantages for efficient SoC debugging.

First, it provides the customized trace for diverse debugging purposes. Depending on the debugging purpose, designers can select a preferred abstraction level to observe bus signal variation. For designers debugging at a higher abstraction level, it saves a lot of time analyzing the skeleton of system operations. The idea is to make the hardware debugging process similar to the software debugging process. Designers can use the higher abstraction level trace to obtain the top view and then switch to the lower abstraction level trace on-the-fly to check the detail signals.

Second, the multiresolution tracing saves trace sizes. Since higher-abstraction-level traces capture abstracted data, the required space is smaller. Therefore, given a fixed-size trace memory, the trace depth (cycle) in the higher abstraction level is larger than the traces in the lower abstraction level.

V. CONCLUSIONS

An on-chip bus tracer SYS-HMRBT provides the support for the development, integration, debugging, monitoring, and tuning of AHB based SOC's, it is attached to the on-chip AHB bus and is capable of capturing and compressing in real time the bus traces with five modes of resolution. These modes could be dynamically switched while tracing. The bus tracer also supports both directions of traces: pre-T trace and post-T trace. In addition, a graphical user interface, running on a host PC, has been developed to configure the bus tracer and analyse the captured traces. With the aforementioned features, SYS-HMRBT supports a diverse range of design/debugging/monitoring activates, including module development, chip integration, hardware/software integration and debugging, system behavior monitoring, system performance/power analysis and optimization.

ACKNOWLEDGMENT

I would like to thank the innumerable people throughout my life who taught me ,including my parents, my academic teachers, my guide for their generous support.

REFERENCES

[1] ARM Ltd., San Jose, CA, "AMBA Specification (REV 2.0) ARM IHI0011A," 1999.
[2] E. Anis and N. Nicolici, "Low cost debug architecture using lossy compression for silicon debug," in *Proc. IEEE Des., Autom. Test Eur. Conf.*, Apr. 16–20, 2007, pp. 1–6.
[3] ARM Ltd., San Jose, CA, "ARM. AMBA AHB Trace Macrocell (HTM) technical reference manual ARM DDI 0328D," 2007.
[4] First Silicon Solutions (FS2) Inc., Sunnyvale, CA, "AMBA navigator spec sheet," 2005.
[5] J. Gaisler, E. Catovic, M. Isomaki, K. Glembo, and S. Habinc, "GRLIB
[6] Infineon Technologies, Milipitas, CA, "TC1775 TriCore users manual system units," 2001.
[7] ARM Ltd., San Jose, CA, "Embedded trace macrocell architecture specification," 2006.
[8] E. Rotenberg, S. Bennett, and J. E. Smith, "A trace cache microarchitecture and evaluation," *IEEE Trans. Comput.*, vol. 48, no. 1, pp. 111–120, Feb. 1999.
[9] A. B. T. Hopkins and K. D. Mcdonald-Maier, "Debug support strategy for systems-on-chips with multiple processor cores," *IEEE Trans. Comput.*, vol. 55, no. 1, pp. 174–184, Feb. 2006.
[10] B. Tabara and K. Hashmi, "Transaction-level modeling and debug of SoCs," presented at the IP SoC Conf., France, 2004.
[11] B. Vermeulen, K. Goosen, R. van Steeden, and M. Bennebroek, "Communication-centric SoC debug using transactions," in *Proc. 12th IEEE Eur. Test Symp.*, May 20–24, 2007, pp. 69–76.
[12] Y.-T. Lin, C.-C. Wang, and I.-J. Huang, "AMBA AHB bus protocol checker with efficient debugging mechanism," in *Proc. IEEE Int. Symp. Circuits Syst.*, Seattle, WA, May 18–21, 2008, pp. 928–931.
[13] Fu-ching,Yi-Ting- Lin,Chung-Fu Kao and Ing-Jer Huang "An on chip bus tracer with real time compression and dynamic multiresolution support for soc". *IEEE trans .vlsi.*,vol 19,no.4 april 2011.
[14] Yi-Ting Lin, Wen-Chi Shiue, and Ing-Jer Huang "A Multi-resolution AHB Bus Tracer for Real-time Compression of Forward/Backward Traces in a Circular Buffer" DAC *2008*, June 8–13, 2008, Anaheim, California.