ON-CHIP: AMBA- AHB DESIGN USING DYNAMIC COMPRESSION AND MULTI-RESOLUTION BUS TRACER TECHNIQUE

Dr.Nikhil Raj [1], Dr M. Thamarai[2], Dr. Ganapathi sridhar[3], Dr.D. Kiran[4], Dr. Kishor Krishna kumar[5]
[1,2,3,4,]Professor, [5]Asst. professor, Dept. of ECE,MRCE, Hyderabad

Abstract— AMBA (Advanced Microcontroller based Bus Architecture) consists of AHB, APB, ASB and AXI. In this project we are Tracing AHB (Advanced High performance Bus) signals with Real time Compression and Multiresolution Techniques. A simple transaction on the AHB consists of an address phase and a subsequent data phase. Access to the target device is controlled through a MUX , thereby admitting bus-access to one bus-master at a time. In AHB Tracer we have to Trace Address signals, Data signals and Control signals the have to compress them depending on AHB protocols. A multiresolution AHB on-chip bus tracer is named as SYS_HMRBT (AHB Multiresolution Bus Tracer) and is used monitoring. By using this SYS_HMRBT, we can achieve 79%-96% of compression depending on selected resolution mode.

Key words- AHB, AMBA, compression, multiresolution, post- T trace, pre-T trace, real time trace.

INTRODUCTION

AHB Tracer

The ON-CHIP bus is an important system-on-chip (SoC) infrastructure that connects major hardware components. Monitoring the on-chip bus signals is crucial to the SoC debugging and performance analysis/optimization.

Unfortunately, such signals are difficult to observe since they are deeply embedded in a SoC and there are often no sufficient I/O pins to access these signals. Therefore, a straightforward approach is to embed a bus tracer in SoC to capture the bus signal trace and store the trace in an on-chip storage such as the trace memory which could then be off loaded to outside world (the trace analyzer software) for analysis. 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 granularities and trace length to make the most use of the trace memory. In addition, the bus tracer is capable of tracing signals before/after the event triggering, named

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-4, ISSUE-12, 2017
DOI:10.21276/ijcesr.2017.4.12.20

pre-T/post-T tracing, respectively. This feature provides a more flexible tracing to focus on the rest of this documentation is organized as follows. Chapter2.2 surveys the related work. Chapter3 illustrates the literature survey of AHB Tracer. Chapter4 presents the hardware architecture of our bus tracer. Chapter5 provides experiments to analyze the compression ratio, trace depth, and cost of our bus tracer. A case study is also conducted to integrate the bus tracer with a 3-D graphics SoC. Finally, Chapter7 concludes this project and gives directions for future research.

.

II VARIOUS CODE TECHNIQUES:

DUPLEX SYSTEM:

A duplex framework is an illustration of a traditional excess plan that might be utilized for simultaneous lapse location demonstrates the fundamental structure of a duplex framework. Duplication has been utilized for simultaneous mistake location as a part of various frameworks including the Bell Switching System, from organizations like Stratus and Sequoia. In any duplex framework there are two modules (indicated in Fig. 2.1 as Module 1 and Module 2) that actualize the same rationale capacity. The two executions are not so much the same. A comparator is utilized to check whether the yields from the two modules concur. On the off chance that the yields deviate, the framework demonstrates a lapse. For a duplex framework, information uprightness is protected the length of both modules don't deliver indistinguishable blunders (expecting that the comparator is shortcoming free). Since the comparator is significant to the right operation of the duplex framework, extraordinary checking toward oneself comparator plans (e.g., two-rail checker) that ensure information respectability against single comparator flaws must be utilized

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 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 realtime on-chip bus tracing.

Existing on-chip bus tracers mostly adopt lossless compression approaches. ARM provides the AMBA AHB trace macrocell (HTM) [4] 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 (to be explained shortly). 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 [5] 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. The AHBTRACE in GRLIB IP library [2] 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 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. Other technique such as the slice compression approach [2] 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. For control signals, ARM HTM [4] encodes them with the slice compression approach: the control signal is recorded only when the value changes.

	Cycle Level	**Transactio n Level**
Time Granularit y	**Cycle Accurat e**	**Event Triggering**

Table1 TIMING ABSTRACTION

As mentioned, compressing all signals at the cycle- accurate-level does not always meet the 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 propose the 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.

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 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 IMPLEMENTATION

Figure1 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.

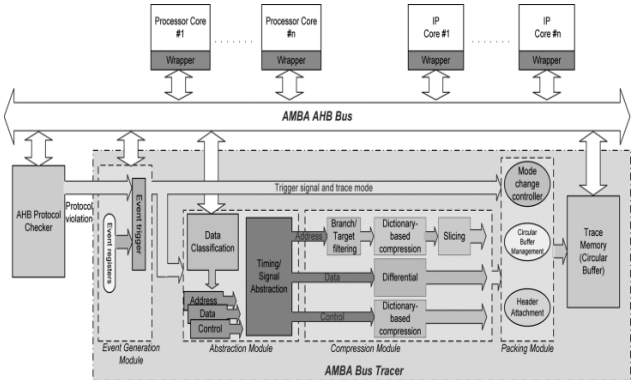


Figure1 Multiresolution bus tracer block diagram.

In AHB Tracer we have the following modules

--Event Generator

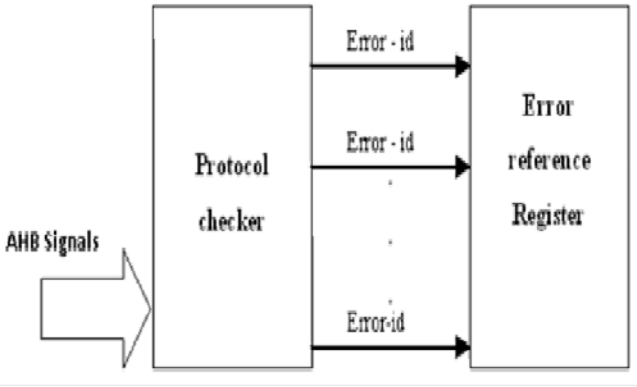
--Abstraction

--Compression

--Packing

In addition to these modules we use Trace Memory o save the data which is compressed by the Tracer. And we use CHECKER as a external module from where we can trace data other than AHB bus.

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-4, ISSUE-12, 2017
DOI:10.21276/ijcesr.2017.4.12.20



AHB Protocol checker (HP checker)

Figure2 Protocol Checker

Figure2 shows AHB Protocol Checker (HP Checker) architecture, which contains two main function blocks: Protocol Checker, ERROR Reference Table .Let us introduce these two blocks individually.

HPChecker is a rule-based protocol checker, thus how to establish a set of well-defined rules is very important. We reference Synopsys verification intellectual property (VIP) to establish rules. Besides, according to our design experiences, we add new rules to increase our error finding ability. In conclusion, our protocol checker has rules, including master-related rules, slave-related rules, reset-related rules, and bus components-related rules. Bus components include arbiter and decoder.

Protocol Checker is the main core of HPChecker, the inputs are all AHB bus signals, and the outputs are ERROR signals and corresponding master and slave IDs. Every rule has its own corresponded bit because every cycle maybe occur more than one error. If the ith bit of ERROR is set, which indicates current bus signals violate ith rule. The Master/Slave ID is formed by the HMASTER signal. If an error occurs, the HPChecker will output the corresponded master ID number or slave ID number to indicate which master or slave violates the AHB protocol. 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 registers. Optionally, this module can also accept events from external modules. For example, we can connect an AHB bus protocol checker (HPChecker) to the Event Generation Module, as shown in Figure4.1, to capture the bus protocol related trace.

Table4.1 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

32 bits					
Address					
Address Mask					
Data					
Data Mask					
Control					
Control Mask					
Trace Depth					
Trace Mode(4bits)	**Direc tion**	**Ena ble**		**Chec ker Event**	**Even t Num bers (24 bits)**
Event Numbers(21 bits)					**[10:0] zeros**

Table2 Event Register.

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 as mentioned below:

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

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-4, ISSUE-12, 2017
DOI:10.21276/ijcesr.2017.4.12.20

RESULTS

Checker Result

Figure Checker Simulation Result

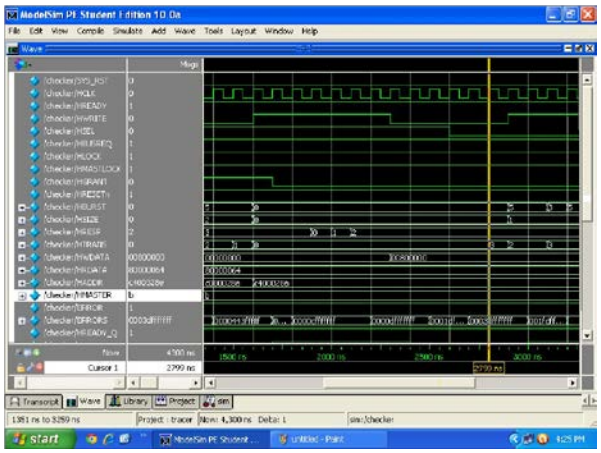
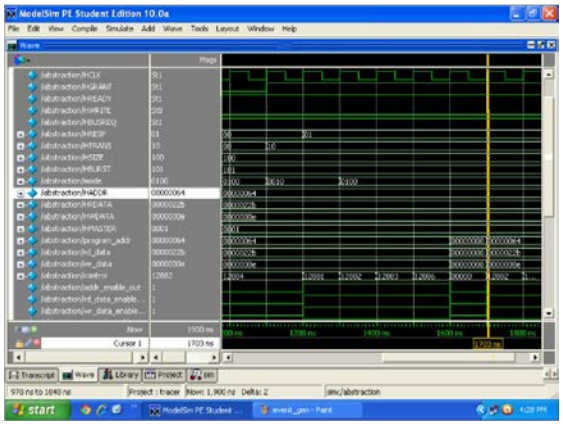


Figure5.1 shows the simulation result of checker module. The output for this module is ERROR register of 44 bit length, in which each bit represents various protocol errors of AHB. For example when reset signal is high (HRESETn) then all the control signals should be at initial state otherwise they will produce an error. The protocol list is given in table. Corresponding to the error, bit of the error register will respond.



Event Generator Result

Figure Event Generation Simulation Result

This module is responsible for producing the control signal for the tracer, which represents the start and stop point of the trace. Trace_In_Progress is the output signal for this module. And this module also produces mode of trace on which basis the tracer is working. Figure5.2 shows the simulation result for the Event Generation module.

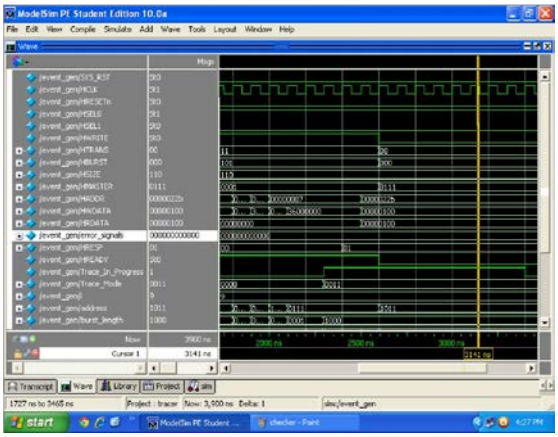


Figure Abstraction Simulation Result

Abstraction module takes the inputs from the AHB bus and the Event Generation module. If divides the AHB signals into ADDRESS signals, DATA signals and control signals. It is also responsible for producing the output depends on the mode of operation. For example if the trace mode is in Full cycle signal (FC) then it produces the output for every clock cycle. If it is in Bus transaction mode first it encodes the PCS control signals and generates the output on transactions only. Figure5.3 shows the simulation result for ABSTRACTION module.

CONCLUSION

We have presented an on-chip bus tracer SYS-HMRBT 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 (trace before the triggering event) and post-T trace (trace after the triggering event). In addition, a graphical user interface, running on a

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-4, ISSUE-12, 2017
DOI:10.21276/ijcesr.2017.4.12.20

host PC, has been developed to configure the bus tracer and analyze the captured traces. With the aforementioned features, SYS-HMRBT supports a diverse range of design/debugging/ monitoring activities, including module development, chip integration, hardware/software integration and debugging, system behavior monitoring, system performance/power analysis and optimization, etc. The users are allowed to tradeoff between trace granularity and trace depth in order to make the most use of the on-chip trace memory or I/O pins.

REFERENCES

[1] YANG et al.: On-Chip AHB Bus Tracer with Real-Time Compression

[2] AMBA AHB Bus Potocol Checker with Efficient Debugging Mechanism Yi-Ting Lin, Chien-Chou Wang, and Ing-Jer Huang Department of Computer Science and Engineering National Sun Yat-sen University.

[3] ARM Ltd., San Jose, CA, "AMBA Specification (REV 2.0) ARM IHI0011A," 1999.

[4] ARM Ltd., San Jose, CA, "ARM. AMBA AHB Trace Macrocell (HTM) technical reference manual ARM DDI 0328D," 2007.

[5] AHB Example AMBA System Technical Reference Manual, DDI0170A 1999 ARM Limited.

[6] ARM IHI 0011A AMBA™ Specification (Rev 2.0)

[7]http://en.wikipedia.org/wiki/Advanced_Microcontroller_Bus _Architecture

[8]H.NaeimiandA.DeHon,"Faultsecureencoderanddecoderf or nanoMemory applications,"IEEETrans.VeryLargeScaleIntegr.(VLSI) Syst.,vol.17,no.4,pp.473–486,Apr.2009.

[9] S. Liu, P. Reviriego, andJ. A. Maestro, "Efficient majority logic fault detection with difference-set codes for memory applications," IEEE Trans. Very LargeScale Integr. (VLSI) Syst., vol. 20, no. 1, pp.148– 156,Jan.2012.

[10] M. Y. Hsiao, D. C. Bossen, andR. T. Chien, "Orthogonal latinsquare codes," IBM J. Res.Develop., vol. 14, no. 4, pp. 390–394, Jul.1970.

[11]S.E. Lee,Y.S.Yang,G.S.Choi,W.Wu, andR.Iyer,"Low- power, resilientinterconnectionwithOrthogonalLatinSquares,"IEEED esign TestComput.,vol.28,no.2,pp.30–39,Mar.–Apr.2011.

[12] R.DattaandN. A. Touba, "Generatingburst-errorcorrectingcodesfrom orthogonallatin squarecodes–agraph theoreticapproach,"inProc.IEEE Int.Symp.DefectFaultToleranceVLSINanotechnol. Syst.,Oct.201 1, pp.367–373.

[13]A.R.Alameldeen,Z.Chishti,C.Wilkerson,W.Wu, andS.- L.Lu, "Adaptivecachedesigntoenablereliablelow-voltage operation,"IEEE Trans.Comput.,vol.60,no.1,pp.50–63,Jan.2011.

[14] G. C. Cardarilli, S. Pontarelli, M.Re, and A.Salsano,"Concurrent errordetectioninReed-Solomonencodersanddecoders,"IEEETrans. TheoryAppl.,vol.27,no.2,pp.215–218,Apr.2011. Large Scale Integr. (VLSI) Syst., vol. 15, no. 7, pp. 842–846, Jul.2007.

[15] I. M. Boyarinov, "Self-checking circuits and decoding algorithms forbinaryhammingandBCHcodesandReed- Solomon codesoverGF(2),"Prob.Inf. Transmiss.,vol.44,no.2,pp.99– 111,2008.

[16] H. Jaber, F. Monteiro, S. J. Piestrak, andA. Dandache, "Design of parallel fault-secure encoders for systematic cyclic block trans- mission codes," Microelectron. J., vol. 40, no. 12, pp. 1686–1697, Dec.2009.

[17]S. J.Piestrak,A. Dandache,and F. Monteiro, "Designing fault-secure parallelencoders forsystematic linearerror correctingcodes," IEEE Trans.Reliab.,vol.52,no.4,pp.492–500,Apr.200

[18]J.A.Maestro,P.Reviriego,C.Argyrides, andD.K.Pradhan,"Fault tolerant singleerror correction encoders," J. Electron.Test., J.DénesandA.D.Keedwell,LatinSquaresandTheirApplications SanFrancisco,CA:Academic ,1974.

[20] P.K.Lala,Self-CheckingandFault-Tolerant DigitalDesign.SanMateo, CA:MorganKaufmann,2001.

[21] K. De, C. Natarajan, D. Nair, andP. Banerjee, "RSYN: A system for automated synthesis of reliable multilevel circuits," IEEETrans. Very Large Scale Integr. (VLSI) Syst., vol. 2, no. 2, pp. 186–195, Jun.1994.

[22] N. A. Toubaand E. J. McCluskey, "Logic synthesis techniques for reduced area implementation of multilevel circuits with concurrent errordetection,"inProc.IEEE/ACM Int.Conf.Comput.- AidedDesign, Nov.1994,pp.651–654.

[23] M. Y. Hsiao, "A class of optimal minimum odd-weight column SEC-DEDcodes,"IBMJ.Res.Develop.,vol.14,no.4,pp.395–301,