www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 1 January 2015, Page No. 9994-9998

A Configurable Bus-Tracer For Errorreproduction In Post-Silicon Validation

E.Deepthi, Cha.Swamy, S.Prabhakar

ABSTRACT
In today's modern system-on-chips , there are several intellectual properties on the system to provide different functionality. However, more complex communications on SoCs the harder at which the programmer could discover all errors before first silicon during its verification. Therefore, it provides a reconfigurable unit for recording the transactions between IPs and adopt logical vector clock as a time stamp of each trace. The programmable trigger unit in debugging node could be configured by the validation to cache their interest sequences of transaction. Because traces of transactions would have their own timestamp, during the post-silicon validation, finally it could reproduce the errors in faulty transactions between IPs and get more information for by passing or fixing the problems. In future, due to several entries of traces finally shrink observation window very quickly, it also implement a compressor to compress traces before it store them into trace buffer. Finally,experiments demonstrate that the proposed debugging architecture is capable of recording the critical transactions, and the proposed reconfigurable debugging unit whole debugging execution time can be reduced more than 80%.

Keywords:AHBtracer,postsiliconvalidation,system onchiparchitectures,PTU(programmable trigger unit),cortex.

I.INTRODUCTION

Modernsystem-on-chips(SoCs) integrate multiple intellectual properties (IPs) including general purpose cores, digital signal processing cores (DSPs) and referred active peripherals

that are connected by standard interfaces such as buses on a single monolithic substrate. For example, Apple A4, a SoC design for the main processor of Apple iPhone, combines an ARM Cortex-A8 CPU, Power VR GPU and a memory controller.SoCsprovides us several indispensable benefits. First, integrating existing re-usable IP blocks simplifies the complexity of new SoC designs. Second, in custom designs, the modular architecture of SoCs provides the flexibility for specified applications. next, due to the characteristics of modular IPs in SoC designs, idle IP blocks can be powered down while production runs, thus lowering the power consumption of the system. Although SoCs have such advantages, they still have some chal- lenges need to be overcome for embedded system developers. The most hard-to-solve problem in SoCs is validation and debugging. SoC validation requires identifying errors in individual IP blocks, their interactions and whole system by running test programms . The test program would not stop until a system failure occurs. Sub- sequentially, developers will exploits auxiliary tools, such as debugging software on host machine, in order to localize the failure into a small region as well as to identify the root cause. Then, they could fix or bypass the failure by micro patching or circuit editing in the end. The main reasons for difficulty in SoC validation are described as ,Due to the limit length of an on-chip trace buffer, which is used to collect internal signal traces for a period of time, the root cause is hard-to-detect bugs may not be able to identify only one debugging session. As a result, the number of debugging sessions as wellhis research was supported by the Industrial could fix or bypass the failure by micro patching or circuit editing in the end. The main reasons for difficulty in SoC validation are described as,Due to the limit length of on-chip trace buffer, which is used to collect internal signal traces for a period of time, the root cause of hard-to-detect bugs may not be able to identify in only one debugging session. As a result, the number of debugging sessions as well This research was supported by the Industrial Technology Research Institute grant (100C551). As design time would increase in virtue of the hard-to-detect bugs. . It is hard to re-generate the faulty sequences because of non-deterministic execution . Basically, a methodology called "cyclic debugging" for removing a hardware bug is often adopted in today SoC development. To make the detected bug manifests itself, faulting sequence is re-executed and the faulty hardware state is returned by supplying the same stimulus. Unfortunately, board-level hardware does not guarantee deterministic

execution . Therefore, it is hard to expose the bug, caused by non-deterministic events, in the following debugging cycle. With the rapid progress of chip fabrication techniques, dozens of IPs and communication fabrics are integrated into a chip. The more IPs in a chip, the more complex communication behaviours they have, which raises the probability of interaction errors . Since individual IPs are typically well validated in isolation, the major cause of the interaction errors is due to unexpected communications between IPs in the design. Consequently, SoC validation and debugging solutions regarding interaction bugs are increasingly important for future SoC designs, especially for detecting and reproducing the interaction errors. Contributions In this work a hardware debugging solution is proposed to solve the above challenges in context of SoC validation. The key idea is caching transactions of interest as well as their timestamps of an IP via a hardware monitor piggybacked on the IP's interface. When a predefined transaction is detected, targeted information with compressed timestamp would be stored in the on-chip trace buffer. In summary, the contributions of this work are as following: We propose a debugging architecture, debugging node, with programmable unit, for monitoring the interactionsbetweenIPs.We propose a timestamp recording mechanism for ordering non-deterministic interactions between IPs and with compression technique in order to broad observation windows of traces. The rest of the paper is organized. Next introduces the related work, furthermore describes our debugging architecture with our debugging node and presents experimental evaluation of architecture and finally concludes the paper.

II.RELATED WORK

In order to catch the bugs when the first-silicon is available, post-silicon validation has four crucial steps: detecting a problem, localizing the problem, identifying the root cause of the problem, and fixing or bypassing the problem. During post-silicon stage, detecting errors and collecting internal signals for localizing problem are not intuitive; on-chip debugging units are apparently indispensable. proposed a scalable design for debug (DFD) architecture with distributed embedded logic analyzer (ELA) to better utilize the available on-chip storage for distributed trace buffers. In order to ideally solve diversified problems, a reconfigurable debugging instrument is also needed. The feature of reconfiguration is able to dynamically create new hardware structures in existing silicon for debugging purposes. In our work we adopt ELA to be the fundament of our debugging unit and apply a Programmable unit for validate to select specific transaction sequences to be recorded. To track the SoC runtime execution, hardware matcher provides an efficient way to monitor particular sequences of signals about potential errors for each IP. For the non-deterministic execution sequences, we adopt Lamports logical vector clock , which had been adopt in several works, to order the transaction sequences and identify relation between each transaction among IPs. Using logical vector clock also enhances the ability for validator to replay the error inter- actions and fetch crucial information in transactions. The statistic information proposed by points out that the interaction errors have predominated the escaped errors. Due to dozens of IPs and complex communication fabric in SoCs, interaction errors are expected to occupy a higher proportion of escaped errors in the future. Therefore, in this paper, we emphasize on debugging communications among IPs and propose architecture to monitor each transaction on the buses and record targeted transactions with the logical timestamps. There are some other works based on transaction debugging as well. Presented a specific trace buffer with Finite State Machine for post-analysis, and proposed a debug aware network interface which is compatible with AXI standard. However, our debug method is based on AHB buses , and the traces with timestamps could help us to identify the sequences of fault transactions and even replay the transactions with a checkpoint to reproduce the errors . In this paper, we focus on providing a debugging architecture with programmable trigger-unit and capacity efficient traces for validator to identify or reproduce errors.

III.DEBUGGING ARCHITECTURE

This work is based on the trace-based debugging used in most of today's SoC validation. Internal signals of SoCs are monitored while test programs running on the system at speed. Once desired trigger conditions have detected, the corresponded data are captured and stored into an internal trace buffer to obtain the system real-time observability. The content of trace buffer would be offloaded via serial interfaces for bug identification. In addition, in order to enable bug reproduction ability and to close the captured data from the observed system failure, a timestamp approach for SoC system failures is proposed in our debugging architecture. The address, data and control signals from the abstraction module are the inputs for the compression module. The signals are compressed based o n different compression techniques. In this compression module we instantiated three compression modules (Address, data, and control compression). The outputs of the compression module are given to the packing module.

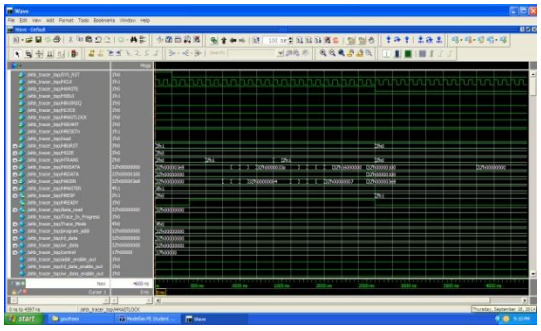


Figure 1 AHB tracer result for compressing nodes

3.1 Debugging Node

All components in DN show in Figure 2. Our debugging architecture consists of four major steps: configuring, monitoring, data recording, and post-processing. First, the trigger condition in the programmable trigger unit (PTU) of DN can be reconfigured by the control unit for multiple debugging experiments. After running a test program, each DN, piggybacked on a specified IP, would monitor the communication event/sequence of the IP and record it with timestamp. Furthermore, each DN has a timestamp unit, called Local Timing Vector (LTV), to record the timing information of the triggered communication events. Each time when PTU had been triggered, the LTV would request the latest global timing vector (GTV) and increase the number in field of vector, which represent the master itself, and update GTV. Subsequently, the alarm of the detected communication event and its new LTV would be com- pressed and stored into trace buffer. The high level description of LTV and GTV operation are depicted in Figure 3, respectively.

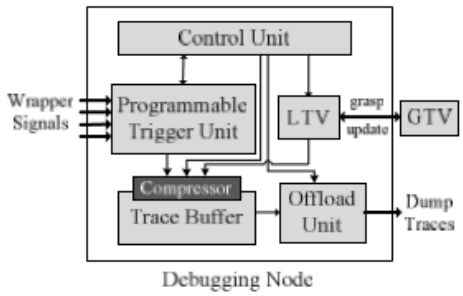


Figure 2: Debugging node

Variables

MWap1, MWap2, ...: the master wrapper internal signals
TrigCond: the configurable trigger condition, such as a single-word or a burst request
LTV: local timing vector
GTV: global timing vector
LTV[MasterId]: the specific element in vector LTV for recording #master on the system bus

LTV Operations

1. *TrigCond* = some configuration of *MWap1, MWap2, ...* ;
2. If(*TrigCond*)
3. Request latest GTV with *MasterId* from GTV block;
4. *LTV* = *GTV*;
5. *LTV[MasterId]* ++;
6. Send *LTV* to Compressor;

GTV Operations

1. If(get request from LTV with *MasterId*)
2. Send the latest *GTV* as response
3. *GTV[MasterId]* ++

Figure 3 LTV and GTV Operations:When the test program finished, the traces could be utilized for localizing or reproducing error sequences. If the clue for locating the bug is not sufficient, the system must re-executes for collecting more information about the bug. To avoid the bug disappears in the next debugging session, the bug reproduction mechanism must be enabled. Therefore, the debugging software not only processes data for locating bug but also analyzes the timing information for bug reproduction. However, in this work we emphasize on recording and bug reproduction with timestamps. One of the central aspects of this work is watching erroneous transaction sequences. Therefore, in order to know when a master's transfer begins, a DN is attached to IP's wrapper interface. For ex- ample, master wrapper provides DN sufficient information to watching any types of transfers the master requested, such as burst or locked transfers. The DN observes the wrapper signals and compares the signals with the triggered conditions in PTU.

3.2 LTV Compressing and Recovering

In order to generate a partial order of transfers between masters on a slave, each master and arbiter maintains their own timing vector table in the trace buffer. If we have four masters and an arbiter in the SoC system, to consider that each field in a timing vector is 32 bits and each vector has five fields, therefore each LTV will be 160 bits in total. Obviously, LTV is too large to the trace buffer. Therefore, large LTV will occupy a lot of space of the trace buffer and will shrink the trace buffer observation window. To overcome this problem, we put a compressor before the trace buffer to widen the observation window and only the difference of LTVs would be recorded in the trace buffer. Here we reuse the temporal vector in compressor, which records the last content of an LTV in the DN, to achieve this goal. As shown in Figure 4 , the progress of LTV compression is as follows:

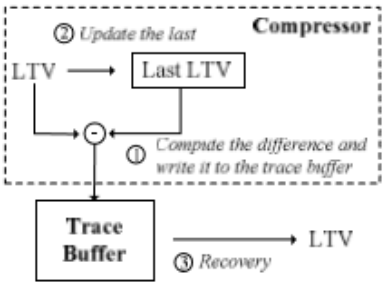


figure 4: The operation of LTV compressor

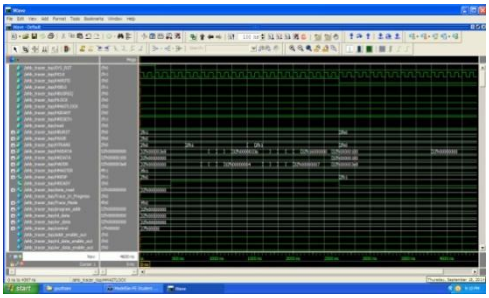
(1) When compressor gets the latest LTV, it computes the difference between the incoming LTV and the temporal vector, then writes the difference to the trace buffer.

(2) Compressor copies the incoming LTV to the temporal vector.

(3) In the post-analysis stage, the difference will be processed to recover the order info. The timestamp recovery process will be described in following

Each masters and arbiter would have their own LTV table in their trace buffer, which could be utilized for post-analysis. However, original timing info is lost in the form of the difference. As a result, the first process in post-analysis is to recover the original timing information. Since each time LTV was being store in trace buffer it subtracts the last LTV, the original value for any entry n can be restored from the differences by using the following formula, where

AHB TRACER TOP MODULE RESULT.



3.2.1ABSTRACTION MODULE

The Abstraction Module monitors the AHB bus and selects/filters signals based on the abstraction mode. The bus signals are classified into four groups as mentioned below

3.3CONCLUSION

We have designed the AHB (Advanced High Performance Bus) Tracer module for error reproduction. The trace can be triggered on a time basis or on the basis of occurrence of an event like “error response”, etc. This tracer is capable of compressing the data so that we can save valuable FIFO space and let the trace continue for a longer period of time without FIFO overflow. This work presents a debugging system for recording and repro- ducing system interaction errors. The debugging system is consists of multiple distributed programmable monitors, which can be config- ured to monitor the execution order for specified interactions between individual IP blocks. The execution history is recorded into trace buffers on each DN. After the execution of test program, the post-analysis could find out the transaction errors between IPs. Once an error is detected, the special post-analysis algorithms would be utilized for reproducing errors. As the results of our experiments demonstrate that the debugging system is capable of dealing with avariety of system-level errors, and improves the debugging execution time more than 80% and storage overhead with compression technique.

3.4FUTURE SCOPE

The tracer designed is compatible with AHB bus. The design may be modified in future to support more advanced buses like AXI, etc**.**

3.5REFERENCES

[1] L. Lamport, "Time, clocks, and the ordering of eventsinadistributed system," Communications of the ACM, vol. 21(7), pp. 558-565, 1978.

[2] "Apple A4 - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Apple_A4.

[3] A. Deshpande, "Verification of IP-Core Based SoC's," in ISQED, 2008.

[4] S. R. Sarangi, B. Greskamp and a. J. Torrellas,"CADRE:Cycle-Accurate Deterministic Replay for Hardware Debugging," in DSN, 2006.

[5] I. Wagner and C.-L. Lu, "Distrubuted Hardware Matcher Framework for SoC Survivability," in DATE, 2011.

E.Deepthi working as Asst.professor in Hyderabad institute of technology and management and her areas of interest in VLSI design, Embedded systems.

CH.A.Swamy working as Assoc.professor in Marri Laxman Reddy institute of technology and management.And his areas of interest in vlsi design,embedded system design, cmos technologies.

S.Prabhakar , a student of Hyderabad institute of technology and management and his areas of interest in embedded systems,digital designs.