Advanced SOC Tracing Technique

A.Sireesha(M.Tech)
Quba College of Engg& Technology
Nellore

MD.Hayath Razvee,M.Tech(VLSI)
Quba College of Engg& Technology
Nellore

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.

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

This paper presents a real-time multi-resolution AHB on-chip bus tracer, named SYS-HMRBT (aHb multiresolution bus tracer)[1]. 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. In addition, the bus tracer is capable of tracing signals before/after the event triggering, named pre-T/post-T tracing, respectively. This feature provides a more flexible tracing to focus on the interesting points.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.

DESIGN AND IMPLENTATION

Figure4.1 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.

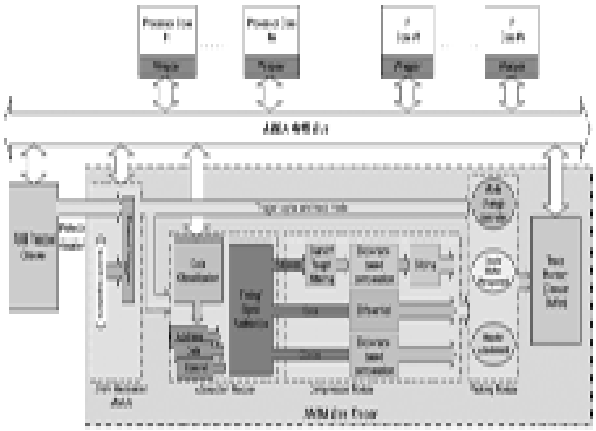


Figure4.1 Multiresolution bus tracer block diagram.

4.1 AHB Protocol checker (HP checker)

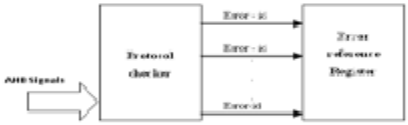


Figure4.2 Protocol Checker

Figure4.2 shows AHB Protocol Checker (HP Checker) architecture, which contains two main function blocks: Protocol Checker, ERROR Reference Table.

HPChecker is a rule-based protocol checker, thus how to establish a set of well-defined rules is very important. 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[2].

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. Table4.1 is the format of an event register.

32 bits →					
Address					
Address Mask					
Data					
Data Mask					
Control					
Control Mask					
Trace Depth					
Trace Mode (4bits)	**Direction**	**Enable**	**AHB Bus**	**Checker Event**	**Event Numbers (24 bits)**
Event Numbers(21 bits)					**[10:0] zeros**

Table4.1 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 level and transaction level, as shown in Table2.1. The cycle level captures the signals at every cycle. The transaction level

records the signals only when their value changes (event triggering).

Figure4.3 BSM for encoding bus master behaviors.

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.

The BSM is designed based on the AMBA AHB 2.0 protocol to represent the key bus handshaking activities within a transaction. The transitions between BSM states follow the AMBA protocol control signals. For example, in the beginning (state 0), if the master is granted the bus (HGRANT = true), it enters start state (state 2). After, the master begins to transfer by first acknowledging the transfer type, which is a sequential transfer (HTRANS = SEQ) or a nonsequential transfer (HTRANS = NONSEQ). If it is a successful transfer, the BSM goes to the normal state (state 3). After it is in state 3, if the slave is busy, the BSM enters to the wait states (HREADY = false and HRESP is OK). Later on, if the slave can finish the transfer, the BSM changes from state 4 to state 5. (HREADY = true and HRESP indicates OK).

Compression Module

The 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 Table4.2 the program address is compressed by a combination of the branch/target filtering, the dictionary-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. Details will be discussed below Compression Mechanism[3].

Program Address Compression

We divide the program address compression into three phases for the spatial locality and the temporal locality. Figure4.5 shows the compression flow. There are three approaches: branch/target filter, dictionary-based compression, and slicing.

Here we have three parts in address compression:

--Branch/Target Filtering
--Dictionary based Compression
--Slicing

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.

Dictionary-B*a*sed 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.

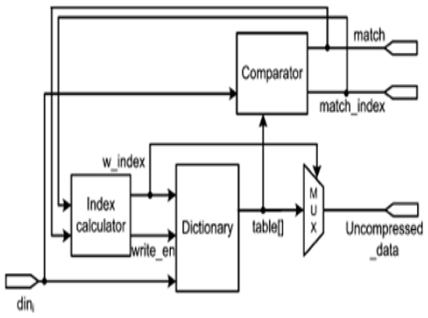


Figure4.7 Block diagram of the dictionary-based compression circuit.

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. Figure4.8 shows the hardware architecture. It has the register REG storing the previous data (din_{i-1}).

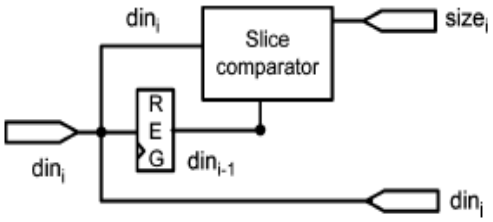


Figure4.8 Block diagram of slicing circuit

Data Address/Value Compression

Data address and data value tends 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. Figur4.9 shows the hardware compressor. The register REG saves the current datum din_i and outputs the previous datum din_{i-1}. By comparing the current datum with the previous data value, the three modules comp, differential, and sizeof output the encoded results. The comp module computes the sign bit (signed_bit) of the difference value. The differential module calculates the absolute difference value (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. Therefore, the sizeof module calculates the nonzero digit number ($size_i$) of the difference. Finally, the encoded datum is sent to the packing module along with $size_i$.

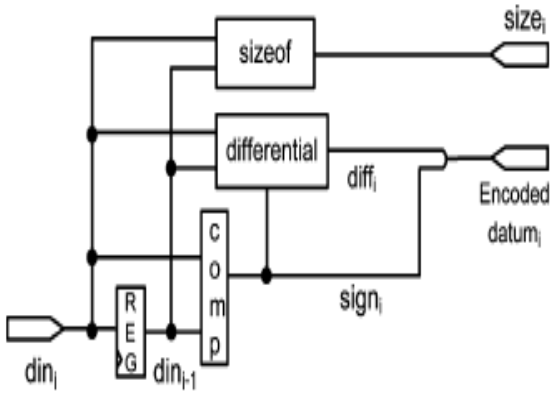


Figure4.9 Block diagram of differential compression circuit

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[4s]. 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.

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[5]. In this paper, we call a compressed data with a header as a packet. 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.

Circular Buffer Management

To decompress those traces (segments) in a circular buffer, we must know where the traces are in the circular buffer and which one is the oldest trace.Therefore, a header position table is used to keep track the location of each trace, as shown in Figue4.12.

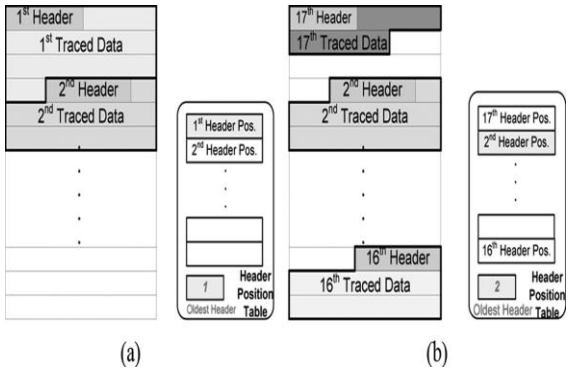


Figure4.12 Trace buffer and assistant header position table.

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.

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. As Figur4.13 shows, the trace mode changes seamlessly during execution.

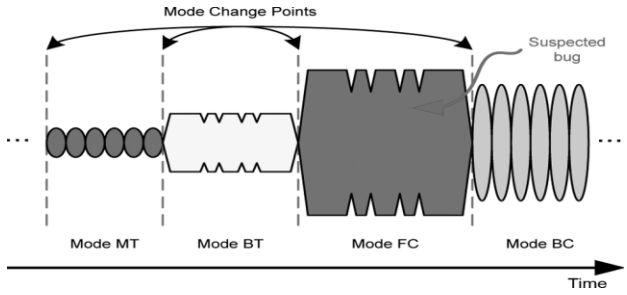


Figure4.13 Debugging/monitoring process with dynamic mode change.

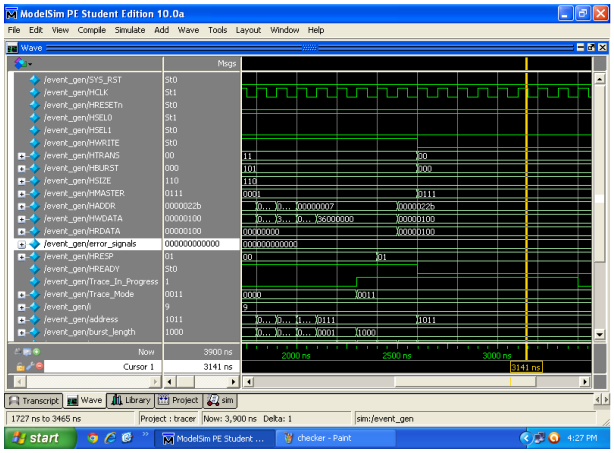
Dynamic mode change has two benefits. One is that it provides customized traces according to the debugging purpose. The other is that designers can make tradeoffs between the trace granularity and the trace depth. Thus, the trace memory utilization is more efficient.The dynamic mode change is achieved by setting up the event registers[6]. The event registers define the start/stop time of a trace and the trace mode.

Thus, when the trigger condition meets and a new trace begins, the new trace starts in the trace mode specified in the event registers.

RESULT

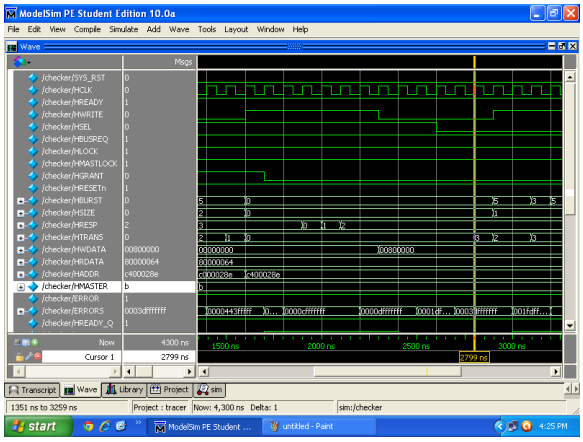
Event Generator Result

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.



Event Generation Simulation Result

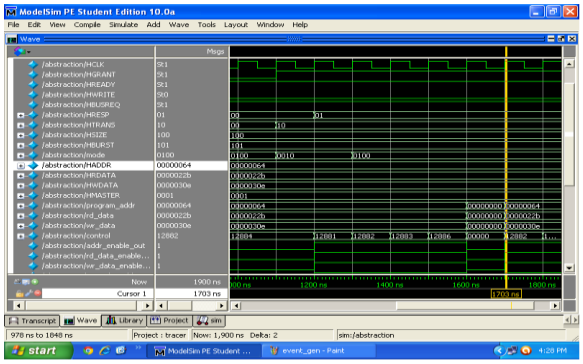
Checker Result



Checker Simulation Result

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.

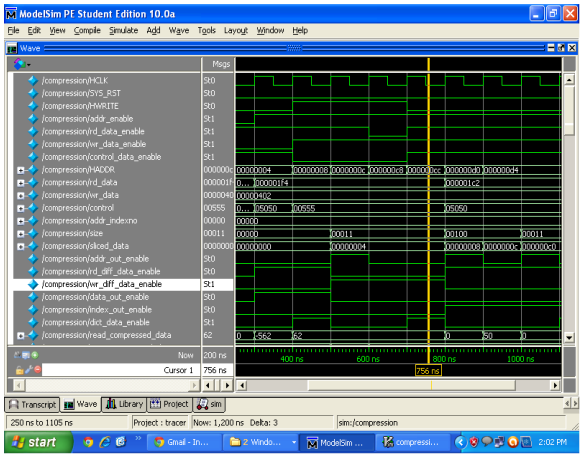
Abstraction Result



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.

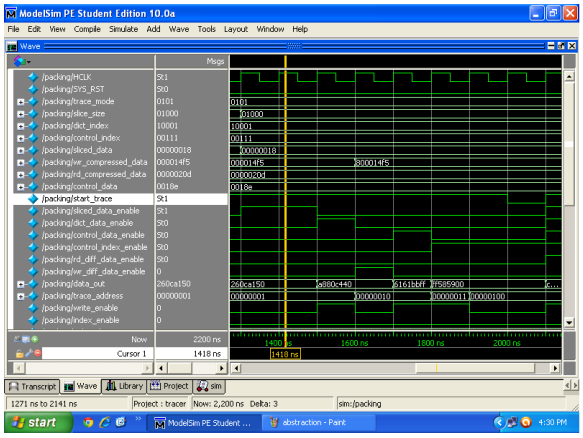
Compression Result



Compression Simulation Result

The address, data and control signals from the abstraction module are the inputs for the compression module. The signals are compressed based on different compression techniques.

Packing Result



Packing Simulation Result

The compressed data is in the form of bits only. If we transmit it directly to the memory, then there will be a great problem at the decoder to differentiate the data. So we have to attach the header for each data. According to the header only decoder can find out the different packets. . Each buffer is of 32bits. Whenever the data in one buffer is full, then that buffer gives the data to the memory.

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 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.In the future, we would extend this work to more advanced buses/connects such as AXI or OCP. In addition, with its real time abstraction capability, we would like to explore the possibility of bridging our bus tracer with ESL design methodology for advanced hardware/software co development/debugging/ monitoring/analysis, etc.

REFFERENCES

[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] http://www.arm.com/products/system-ip/amba/index.php

[9] www.asic-world.com/verilog/veritut.html