International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 1, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Multi Resolution AHB Bus Tracer with Real Time Compression for SOC

J. Jagadish Reddy
M.Tech Student
CVSR College of Engineering
jagadish.javajji@gmail.com

P. Ramakrishna
Associate Professor, ECE Department
CVSR College of Engineering
prk.cvsr@gmail.com

***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 Multi-resolution 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, we have to compress them depending on AHB protocols. A multi-resolution AHB on-chip bus tracer is named as SYS_HMRBT (AHB Multi-resolution Bus Tracer) and is used for monitoring. The goal is to provide better compression quality and multiple resolution traces to meet the complex SoC debugging needs. Compressing all signals at 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. By using this SYS_HMRBT, we can achieve 79%-96% of compression depending on selected resolution mode.**

Tools Used for this Project are Modelsim for Simulation, and Xilinx ISE II for Synthesis.

***Keywords* - AMBA, AHB, Tracer.**

I. INTRODUCTION

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 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. 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.

AHB Trace Macro-Cell (htm)

The HTM provides address and data trace information about AHB buses. The information from an HTM can be used with the debugger to enable easy, accurate debugging on AHB-based embedded systems. The HTM provides extensive resources for event recognition to generate trigger events. The HTM generates trace data for output through the *AMBA Trace Bus* (ATB). The trace debug function is non-intrusive and HTM can be controlled using an APB (AMBA v3) interface.The trace operation indicates the data transfers that have taken place to defined memory locations or regions. Other AMBA control information can also be included. However, other operations like IDLE cycles and BUSY cycles in AHB are not traced..The HTM is an *Advanced Microcontroller Bus Architecture* (AMBA) compliant *System-on-Chip* (SoC) debug component

HTM Features

Other features of the HTM are described in the following sections:

- *Register access lock.*
- *Security pins.*
- *Security and software protection* on page.
- *ASIC control output* on page.
- *Bus select output* on page.
- *ARM11 AHB extensions* on page.
- *Unsupported signals* on page.
- *Power saving and clamping logic* on page.

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 real-time on-chip bus tracing.

Existing on-chip bus tracers mostly adopt lossless compression approaches. ARM provides the AMBA AHB trace macro cell (HTM) 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 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 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). 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 encodes them with the slice compression approach: the control signal is recorded only when the value changes.

	Cycle Level	Transaction Level
Time Granularity	Cycle Accurate	Event Triggering

Table 1 : 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.

II. Advanced Microcontroller Bus Architecture

System on Chip:

A system on a chip or system-on-chip (SOC) is an integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip. A typical application is in the area of embedded systems.

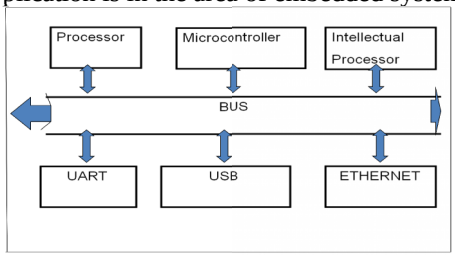

Fig.1. System on chip

These blocks are connected by either a proprietary or industry-standard bus such as the AMBA bus from ARM Holdings. DMA controllers route data directly between external interfaces and memory, bypassing the processor core and thereby increasing the data throughput of the SoC.

International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 1, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Introduction to AMBA

An AMBA-based microcontroller typically consists of a high-performance system backbone bus, able to sustain the external memory bandwidth, on which the CPU and other Direct Memory Access (DMA) devices reside, plus a bridge to a narrower APB bus on which the lower bandwidth peripheral devices are located. Figure 2 shows both AHB and APB in a typical AMBA system.

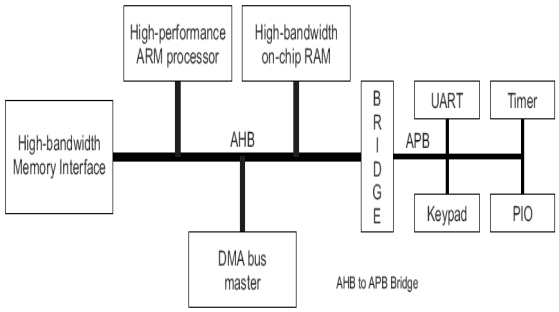

Fig.2. A typical AMBA AHB-based system

The Advanced Microcontroller Bus Architecture (AMBA) is used as the on-chip bus in system-on-a-chip (SOC) designs. Since its inception, the scope of AMBA has gone far beyond microcontroller devices, and is now widely used on a range of ASIC and SOC parts including applications processors used in modern portable mobile devices like smart-phones.

The AMBA protocol is an open standard, on-chip interconnect specification for the connection and management of functional blocks in a System-on-Chip (SOC). It facilitates right-first-time development of multi-processor designs with large numbers of controllers and peripherals.

AMBA was introduced by ARM Ltd in 1996. The first AMBA buses were Advanced System Bus (ASB) and Advanced Peripheral Bus (APB). In its 2nd version, AMBA 2, ARM added AMBA High-performance Bus (AHB) that is a single clock-edge protocol. In 2003, ARM introduced the 3rd generation, AMBA 3, including AXI to reach even higher performance interconnect and the Advanced Trace Bus (ATB) as part of the Core Sight on-chip debug and trace solution. These protocols are today the de-facto standard for 32-bit embedded processors because they are well documented and can be used without royalties.

AMBA-Advanced Microcontroller Bus Architecture Widely used as the on-chip bus for ARM processors. Four Distinct Buses of AMBA, The AMBA specification defines an on-chip communications standard for designing high-performance embedded microcontrollers.

- Advanced extensible Interface (AXI)
- Advanced High-performance Bus (AHB)
- Advanced System Bus (ASB)
- Advanced Peripheral Bus (APB)

AHB introduction

AHB mainly used for high-performance module (such as CPU, the DMA and DSP, etc.), as the connection between the SoC framework10 chip system bus, it includes the following some properties: a single clock edges operating; The three states realization ways; Support sudden transmission; Support subsection transmission; Support multiple main controller; Configurable 32-bit ~ 128-bit bus width; Support bytes, half bytes and word transmission. From the main module, AHB system from module and Infrastructure (Infrastructure) 3 parts, the whole AHB bus by the main module transfers from module issued, responsible for the response.

Basic structure is composed of arbitration device (arbiter), the main module to the many way from module, from the main module into how way, decoder (decoder), virtual from module (dummy Slave), virtual main module (dummy Master) together. Its interconnection layered architecture as shown in figure 3..AMBA AHB transfer can start with the bus master, by asserting a request signal to the arbiter. Then the arbiter when the master will be granted to use the bus. A granted Master bus starts the transfer with address and control signals.

The transfer should include the following details, they are:

- Burst Size
- Size,
- Write Data, Address
- Write/Read Operation
- Transaction Types

Bus Interconnection

The AMBA AHB bus protocol is designed to be used with a central multiplexor interconnection scheme. Using this scheme all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and response signal multiplexor, which selects the appropriate signals from the slave that is involved in the transfer. Figure3 illustrates the structure required to implement an AMBA AHB design with three masters and four slaves.

AHB is a new generation of AMBA bus which is intended to address the requirements of high-performance synthesizable designs. It is a high-performance system bus that supports multiple bus masters and provides high-bandwidth operation.

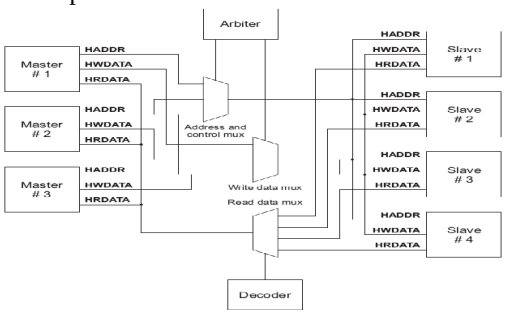

Fig.3. Multiplexor Interconnection

AMBA AHB implements the features required for high-performance, high clock frequency systems including:

- burst transfers
- split transactions
- single-cycle bus master handover
- single-clock edge operation
- non-tri state implementation
- wider data bus configurations (64/128 bits).

International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 1, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

III. Architecture Of AHB Tracer

Block diagram of bus tracer

Figure 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

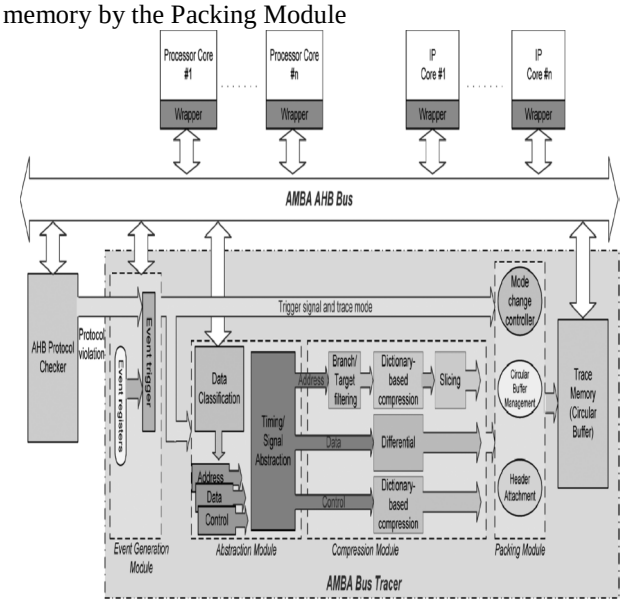

Fig.4. Multi-resolution bus tracer block diagram

In AHB Tracer we have the following modules

- Protocol checker
- Event Generator
- Abstraction
- Compression
- Packing

In addition to these modules we use Trace Memory to 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.

AHB Protocol Checker (hp Checker)

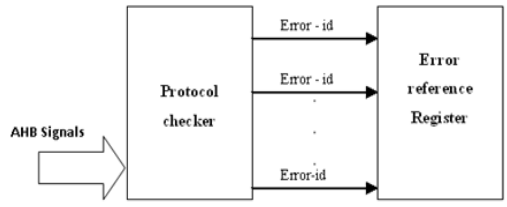

Fig.5. Protocol Checker

Figure 5 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.

HP Checker 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 HP Checker, 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 i^{th} bit of ERROR is set, which indicates current bus signals violate i^{th} rule. The Master/Slave ID is formed by the HMASTER signal. If an error occurs, the HP Checker 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 (HP Checker) to the Event Generation Module, to capture the bus protocol related trace.

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. The cycle level captures the signals at every cycle. The transaction level records the signals only when their value changes (event triggering).

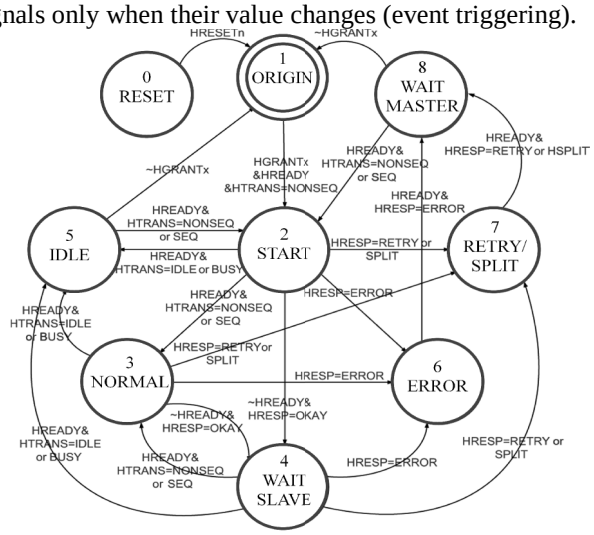

Fig.6. BSM for encoding bus master behaviors.

International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 1, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

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.

AHB Signals	Full Signal	Bus State	Master Operation
Program Address	All	All	Partial[1]
Data Address/value	All	All	Partial[1]
ACS[2]	All	All	Partial[1]
PCS[3]	All	Encoded	N/A

Table 2: Signal Abstraction

-1 Program address, data address/value, and ACS in IDLE, WAIT, and BUSY states are ignored.

-2 Access Control Signals (ACS) includes HWRITE. HSIZE, HBURST, HPROT, HMASTER.

-3 Protocol Control Signals (PCS) includes HBUSREQx, HTRANS, HREADY, and HRESP.

Combining the abstraction levels in the timing dimension and the signal dimension, we provide five modes in different granularities, as Figure 7 shows.

They are: a) Mode FC (full signal, cycle level), b) Mode FT (full signal, transaction level), c) Mode BC (bus state, cycle level), d) Mode BT (bus state, transaction level), and e) Mode MT (master state, transaction level).

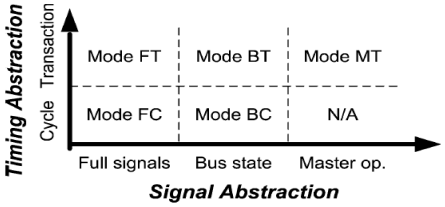

Fig.7. Multi-resolution trace modes

Applications of Abstraction Modes:

Mode FC (full signal, cycle level) 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.

Mode FT (full signal, transaction level) 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 designers want 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.

Mode BC (bus state, cycle level) 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.

Mode BT (bus state, transaction level) 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 and Mode BT. In this mode, designers can easily understand the bus transactions without analyzing the signals at cycle level.

Mode MT (master state, transaction level) 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 master's 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 is IDLE, 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 of observing master behaviors is to realize the whole picture. NOTE: Tracing master behaviors at cycle level is meaningless and can be replaced with Mode BC.

Abstraction Module Implementation

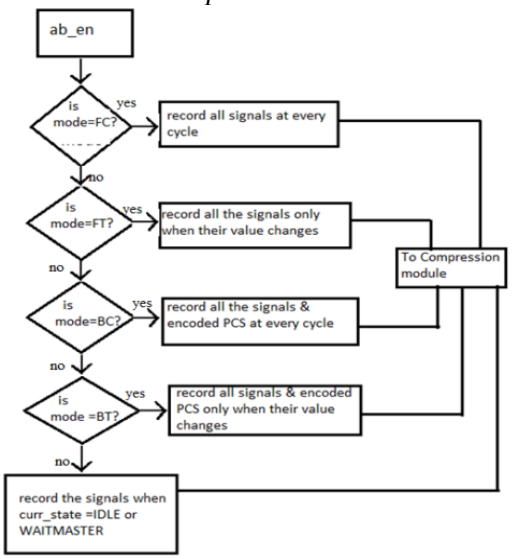

Fig.8. Abstraction module Implementation Flow

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

International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 1, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Compression Mechanism

Although 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.

Program Address Compression

We divide the program address compression into three phases for the spatial locality and the temporal locality. Figure 9 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

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. NOTE: Whenever enable goes to low the state machine runs for another two clock cycles to give the missed incoming address. So that state is given in the state diagram. It's implemented logically by using internal register, which is assigned to two whenever enable goes high. When enable goes low then internal register will be decreased by one. So it tale two cycles to complete the logic.

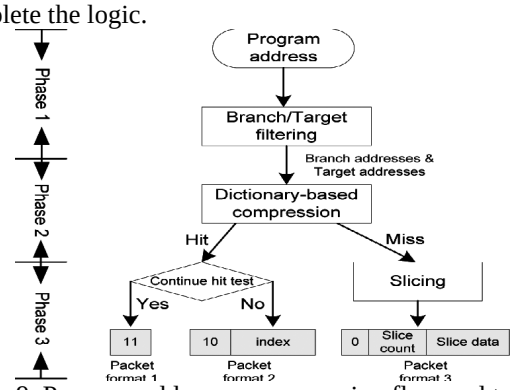

Fig.9. Program addresses compression flow and trace format.

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. Figure 3.7 shows the hardware architecture. The dictionary keeps the frequently appeared branch/target addresses. To keep the hardware cost reasonable, the proposed dictionary is implemented with a CAM-based FIFO. When it is full, the new address will replace the address at the first entry of FIFO. For each input datum (din_i), the comparator compares the datum with the data in the dictionary. If the datum is not in the table (match = Miss), the datum (uncompressed data) is written into the table and also recorded in a trace. Otherwise (match = Hit), the index (match index) of the hit table entry is recorded instead of the datum.

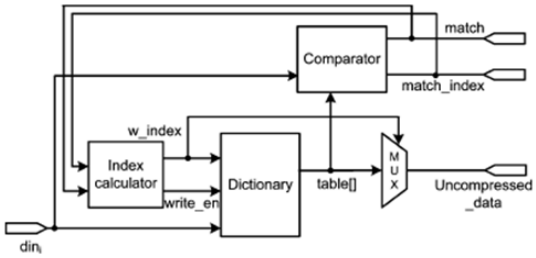

Fig.10. Block diagram of the dictionary-based compression circuit.

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}).

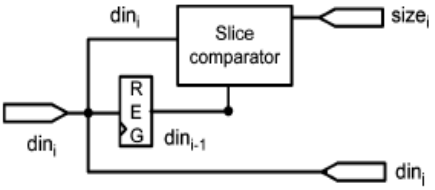

Fig.11. Block diagram of slicing circuit

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 size of 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 size of module calculates the nonzero digit number ($size_i$) of the difference. Finally, the encoded datum is sent to the packing module along with $size_i$.

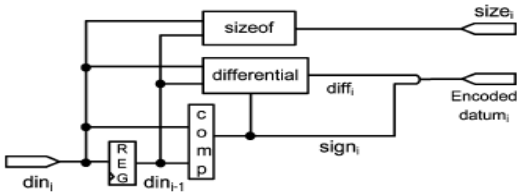

Fig.12. Block diagram of differential compression circuit

For simple hardware implementation, the digit number of an absolute difference is limited to four types, as Figure 13 shows. The header indicates the data trace format. If the difference is larger than 65535 (2^{16}-1), the bus tracer record the uncompressed full 32-bit data value. Otherwise, the bus tracer uses 4-, 8-, or 16-bit length to record the absolute differences, whichever is appropriate.

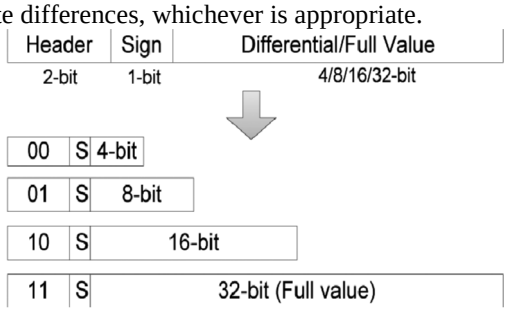

Fig.13. Data address/value trace compression format. S means the sign magnitude.

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. This step generates a proper header and attaches it to each compressed datum. 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. When the tracer is in the pre-T trace mode, this module also maintains the header position. 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.

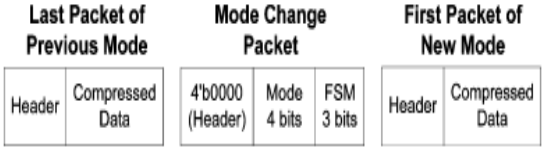

Fig.14. Concatenation of mode-change packet for abstraction mode switch. Header 4'b0000 indicates it is a mode change packet.

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. This task is not straightforward 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 Figure 15.

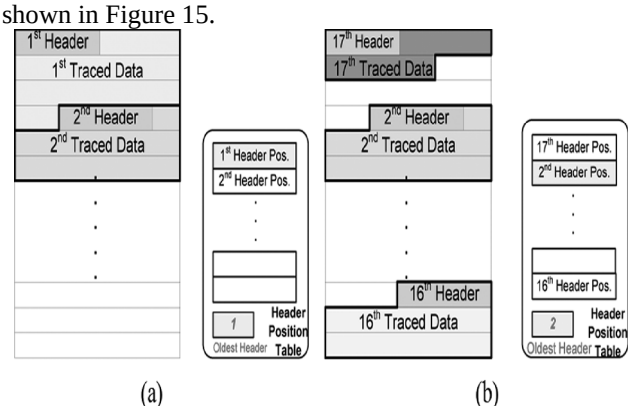

Fig.15. Trace buffer and assistant header position table. (a) The wrapping around does not occur. (b) The wrapping around occurs: the first trace is overwritten by the 17th trace.

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 Figure 15(a), when the circular buffer is not full, this register points to the 1st trace. After, in Figure 15 (b), the buffer is full and trace 17th wraps around the circular buffer, the first trace is damaged because the initial state is overwritten. 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.

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 Figure 16 shows, the trace mode changes seamlessly during execution.

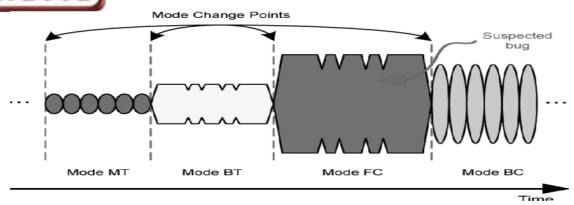

Fig.16. Debugging/monitoring process with dynamic mode change.

Features

This project presents a real-time multi-resolution AHB on-chip bus tracer, named SYS-HMRBT (AHB multi-resolution bus tracer). The bus tracer adopts three trace compression mechanisms to achieve high trace compression ratio. It supports multi-resolution 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.

Feature	Configuration
Trace Mode	FC, FT, BC, BT, MT
Trace Direction	Pre-T, Post-T
Input AHB signals	91 bits (HADDR, HRDATA, HWDATA, ACS's, PCS's)
Output trace word	32 bits
Pipeline stage	5
Max. # of masters	16
FIFO buffer	512 bits

Table 3 : Specification of the implemented SYS-HMRBT Bus Tracers

Debugging Your Results

If you don't get the results you expect, you can use Model Sim's robust debugging environment to track down the cause of the problem.

IV. THEORETICAL RESULTS

Checker Module Result

It 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 Module 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.

Abstraction Module 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 Module Result

It shows the compression module's 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. 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.

Packing Module 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. A simulation result for the Packing module is shown in the figure5. in packing module we used eight buffers which are used to save the data from the packing module. Each buffer is of 32bits. Whenever the data in one buffer is full, then that buffer gives the data to the memory.

Top Module Results

This is output for the AHB TRACER. Here we have Trace memory of 256 locations and each of 32bits. In addition to this memory we implemented INDEX TABLE to give the information where the trace packet is going to be saved in the trace memory. By using this index table we can easily find out the particular tracing information. Following is the trace memory result in different modes.

Mode FC:

```
# 2900 trace_memory[ 2] 8b882e00
# 3000 trace_memory[ 3] 27a8b8a0
# 3200 trace_memory[ 5] 00000101
# 3300 trace_memory[ 6] 44fd0000
# 3400 trace_memory[ 7] f0704a00
# 3500 trace_memory[ 8] 012c1c0a
# 3600 trace_memory[ 9] 17008150
# 3700 trace_memory[ 10] bc040940
# 3800 trace_memory[ 11] e0204a00
# 3900 trace_memory[ 12] 01025000
# 4000 trace_memory[ 13] 08100000
# 4100 trace_memory[ 14] 00812828
# 4200 trace_memory[ 15] 5c04a070
# 4300 trace_memory[ 16] 0a0e0250
# 4400 trace_memory[ 17] 08050700
# 4500 trace_memory[ 18] 41141c04
# 4600 trace_memory[ 19] 14a0e020
```

V. Conclusion and Future Scope

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, This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.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. SYS-HMRBT costs only 42 K gates, making it an valuable and economical investment in a typical SoC which satisfies the requirement of real-time tracing. Experiment results show it achieves high compression ratio ranging from 79% to 96% depending on the trace mode. and test chip levels.

Future Scope

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.

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 IP core user's manual, gaisler research," 2009.
[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. Tabaraand K.Hashmi, "Transaction-level modeling and debug of SoCs,"presented at the IPSoC Conf.,France,2004.
[11] B.Vermeulen, K.Goosen, R.vanSteeden, and M.Bennebroek ,"Communication-centric SoC debug using transactions," in Proc.12thIEEE Eur.TestSymp.,May20–24,2007,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.CircuitsSyst.,Seattle,WA,May18–21, 2008, pp.928–931.
[13] Y.-T.Lin, W.-C.Shiue, and I.-J.Huang, "A multi-resolution AHBbus tracer for read-time compression of forward/ backward traces in a curcular buffer," in Proc. Des. Autom. Conf.(DAC), Jul.2008, pp.862–865.

Author's Profile

J. Jagadish Reddy received the B.Tech degree in Electronics and Communication Engineering from Princeton College of Engineering and Technology, JNTU University, Hyderabad, in 2010, and currently pursuing the M.Tech degree in VLSI System Design from CVSR College of Engineering, JNTU University, Hyderabad respectively.

He is currently a Lab Assistant with the College. He was with the Department of Electronics and Communication Engineering, JNTU University. His research interests include microprocessor/ SoC debugging, SoC platform design, trace compression and hardware/ software co-verification. He has been actively involved in academic, educational and industrial activities

P. Ramakrishna received the B.Tech degree in Electronics and Communication Engineering from NIIT, Warangal in 2006, and the M.Tech degree in VLSI System Design from CVR College of Engineering, JNTU University, Hyderabad in 2009 respectively.

He is currently an Associate Professor with the CVSR College of Engineering with 3 years experience. He was with the Department of Electronics and Communication Engineering, JNTU University. His research interests include system on a chip design, hardware/ software co-design/ verification .