IP Core Development of AMBA-AHB Bus Tracer for SOC

Varapala David[1] R. Veera Bhadraiah[2]
[1]P.G. Scholar [2]Assistant Professor
[1,2]Nimra Institute of Science and Technology Vijayawada, A.P. India

Abstract— *In the system-on-chip (SoC) debugging and performance analysis/optimization, monitoring the on-chip bus signals are necessary. But, such signals are difficult to observe since they are deeply embedded in a SoC and no sufficient I/O pins to access those signals. Therefore, we embed a bus tracer in SoC to capture the bus signals and store them. The stored trace memory can be loaded to the trace analyzer software for analysis. In this paper a multiresolution AHB on- Chip bus tracer is proposed for system-on-chip (SoC) debugging and monitoring which is capable of capturing the bus trace with different resolutions and efficient built-in compression mechanisms. In addition, it allows the user to switch the trace resolution dynamically so that appropriate resolution levels can be applied to different segments of the trace. It also supports tracing after/before an event triggering, named post-triggering trace/pre-triggering trace, respectively. The SoC can be verified in field- programmable gate array. 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.*

Keywords— AHB, on-chip bus, compression, multi-resolution, tracing

I. INTRODUCTION

In the System-on-a-Chip (SoC) era, it is a challenge to verify and debug system chip efficiently and rapidly. For design verification and debugging at system level and chip level, not only external I/O signals observation, but also internal signals tracing can help designer to efficiently analyze and verify the design such as the software program, hardware protocol, and system performance. SoC bus signal tracing can be accomplished with either software or hardware approaches. The software approach trace only program address and the cost of hardware implementation of software approaches would be high. On the other hand, the hardware approach can trace signals at the target system directly. However, the main problem with hardware-based tracing techniques is that the cycle based traces size is usually extremely large.

To address the bus tracing issue, we propose an embedded multi-resolution signal tracing for AMBA AHB. The bus tracer consists of two major tracing approaches: (1) signal monitor/tracing approach, and (2) trace reduction approach. In the first approach, it provides the trade-off between the trace accuracy and the trace depth. Designers can decide to trace AHB signals in detail or in rough depend on debugging objectives; moreover, the trace granularities can be changed while tracing is processed. In other words, different trace strategy results can be stored in a single trace file. In the second approach, the bus tracer compresses the trace data according to AHB signal characteristics such as address, data, and control signals. The traces data will be decompressed on the host, translated into VCD (Value Change Dump) [1] format, and displayed on a waveform viewer. The bus tracer is integrated into an ARM EASY (Example AMBA System) [2] [3] environment with a 3D graphic hardware acceleration system to demonstrate.

This paper presents a real-time multiresolution AHB on-chip bus tracer, named SYSHMRBT (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 tradeoff 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. Chapter1.1 surveys the related work. Chapter2 illustrates the literature survey of abstraction level. Chapter3 presents the hardware architecture of our bus tracer. Chapter4 provides simulation results of our bus tracer so as to analyze the expected errors while processing the signal through bus tracer used in SoC. Finally, Chapter5 concludes this project and gives directions for future research.

A. Related Work

Existing on-chip bus tracers mostly adopt lossless compression approaches. ARM provides the AMBA AHB trace macro-cell (HTM) [3] that is capable of tracing AHB bus signals. We characterize the bus signals into three categories: program address, data address/data and control signals. For program addresses, a straight forward 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 theTC1775 trace module in Tri Core [5] and ARM's Embedded Trace Macrocell (ETM) [6],[4].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. For control signals, ARM HTM [3] encodes them with the slice compression approach: the control signal is recorded only when the value changes. The spirit of a hardware tracer is its data reduction or compression techniques. For program address tracing, an intuitive way is to discard the continuous instruction addresses and retain only the discontinuous ones, such as the addresses of branching and target instructions, with some hardware filters. This approach has been used in some commercial processors, such as TriCore [4] [5], and ARM's Embedded Trace Macrocell [6]. The

hardware overhead of these works is small since the filtering mechanism is simple to implement in hardware. However, the effectiveness of these techniques is mainly limited by the average basic block size, which is roughly around four or five instructions per basic block, as reported in [7] and [8]. For data address and value tracing, the most popular method is used the differential approach based on subtraction. Some researchers have shown that using the differential method can reduce the data address and data values traces by about 40 percent and 14 percent respectively [9] [10]. Besides the address and data bus, there are several control signals on system bus that need to be traced. Some FPGA boards have built-in signal trace tools, such as the Altera Signal Tap [11] and Xilinx Chip- Scope [12]. FS2 AMBA Navigator [13] supports bus clock mode and bus transfer mode to trace bus signals on every clock and bus transfer respectively. Trace buffer stores bus cycles or bus transfers based on local internal memory size. Although these approaches support multiple trace modes such as tracing at cycle by-cycle or at signal Transaction, only one mode can use during a tracing process. The transaction level. They point out that the traditional hardware and software debugging cannot work collaboratively, since the software debugging is at the functional level and the hardware debugging is at the signal level. As a solution, the transaction-level debugging can provide software and hardware designers a common abstraction level to diagnose bugs collaboratively, and thus, help focus problems quickly. Both works indicate that the transaction-level debugging is a must in SoC development.

II. ABSTRACTION LEVEL

This paper presents the multi-resolution Approach that can use different trace modes during a bus signal tracing process. The transaction-level debugging provides software and hardware designers a common abstraction level to diagnose bugs. The abstraction level is in two dimensions: timing abstraction and signal abstraction. The timing dimension 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. The signal dimension involves grouping of 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. They are full signal level, bus state level, and master operation level. The full signal level captures all bus signals. The bus state level further abstracts the PCS by encoding them as states according to the bus-state- machine (BSM).The states represent bus handshaking activities within a bus transaction. The master state level further abstracts the bus state level by only recording the transfer activities of bus masters and ignoring the handshaking activities within transactions. This level also ignores the signals when the bus state is IDLE, WAIT, and BUSY. 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. Combining the abstraction levels in the timing dimension and the signal dimension, we provide five modes in different granularities They are Mode FC (full signal, cycle level), Mode FT (full signal, transaction level), Mode BC (bus state, cycle level), Mode BT (bus state, transaction level), and Mode MT (master state, transaction level). At Mode FC, the tracer traces all bus signals cycle-by-cycle so the detailed bus activities can be observed. At Mode FT, the tracer traces all signals only when their values are changed. At Mode BC, the tracer uses the BSM, such as NORMAL, IDLE, ERROR, and so on, to represent bus transfer activities in cycle accurate level. At Mode BT, the tracer uses bus state to represent bus transfer activities in transaction level Our bus tracer also supports dynamic mode change (DMC) feature which allows designers to change the trace mode dynamically in real-time.

The post-T trace captures signals after a triggering event, while the pre-T trace captures signals before the triggering event. The post-T trace is usually used to observe signals after a known event. The pre-T trace is used to diagnose the cause for unexpected errors by capturing the signals before the errors. In order to overcome the problem, we adopt a periodical triggering technique.

We divide the entire trace into several independent small traces. Destroying the initial state of one trace does not affect other traces since every trace has its own initial state. This technique can be accomplished by periodically triggering a new trace. With minor modification to their control circuitry it can be easily accomplished by the existing trace compression engines.

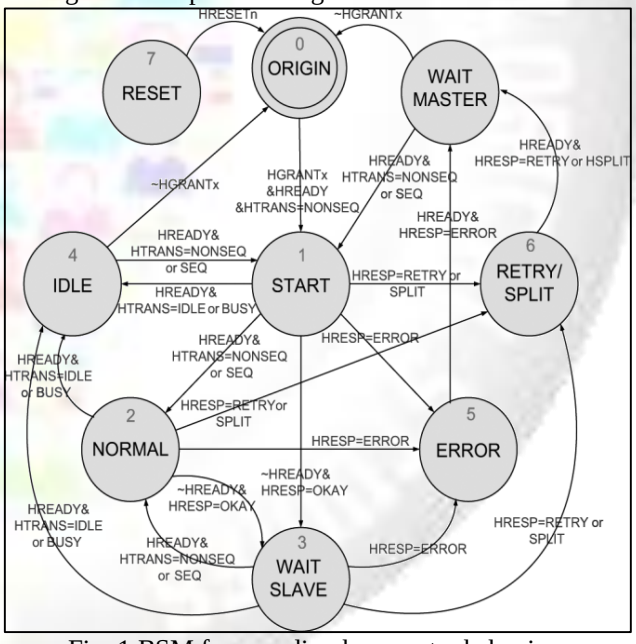


Fig. 1 BSM for encoding bus master behaviors

AHB Signals	Full Signal	Bus State	Master Operation
Program Address	All	All	Partial
Data Address	All	All	Partial
ACS	All	All	Partial
PCS	All	Encoded	N/A

Table I Signal Abstraction

III. HARDWARE ARCHITECTURE OF BUS TRACER

Fig. 2 is the bus tracer overview. It mainly contain 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.

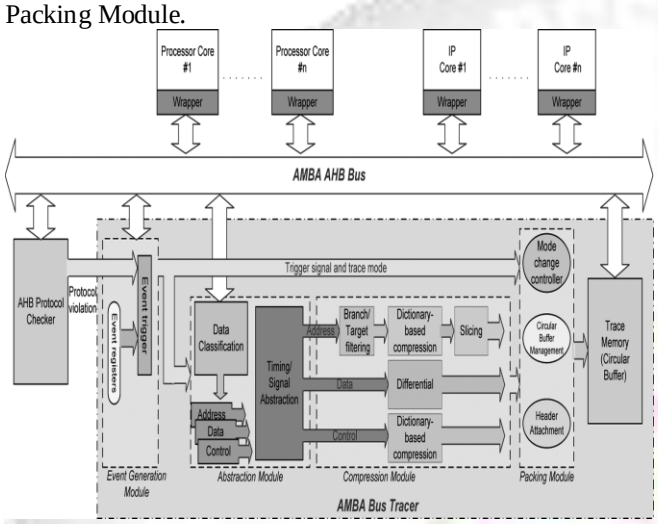


Fig. 2: SoC Tracer Architecture Using AHB Bus

A. *Event Generation Module:*

The trace and trace mode starting and stopping are decided by event generation module. The triggering events on the bus controlled by event registers. The matching circuit is used to compare bus activities with the events specified in the event registers. We can connect an AHB bus protocol checker (HPChecker) [10] to the Event Generation Module, as shown in Fig.2, to capture the bus protocol related trace.

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

Table II Event Register

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. Table II is the format of an event register.

B. *Abstraction Module*

This Module monitors the AMBA bus and selects/filters signals based on the abstraction mode. Depending on the abstraction mode, some signals are ignored, and some signals are reduced.

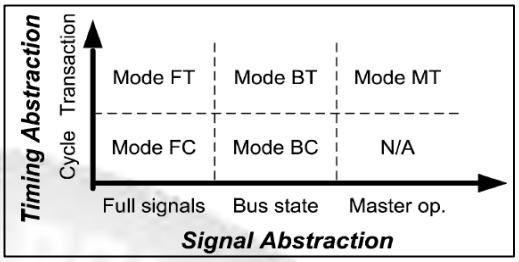


Fig.3 Multiresolution abstraction traces modes

C. *Compression Module*

This Module monitors the AMBA bus and selects/filters signals based on the abstraction mode. Depending on the abstraction mode, some signals are ignored, and some signals are reduced.

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 the Figure-1 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.

1) *Program Address Compression*

We divide the program address compression into three phases for the spatial locality and the temporal locality. Figure-1 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

2) *Branch/Target Filtering*

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

3) *Dictionary-Based Compression*

To further reduce the size, we take the advantage of the temporal locality. Temporal locality exists since the basic blocks repeat frequently (loop structure), which implies the branch and target addresses after Phase 1 repeat frequently. Therefore, we can use the dictionary based compression.

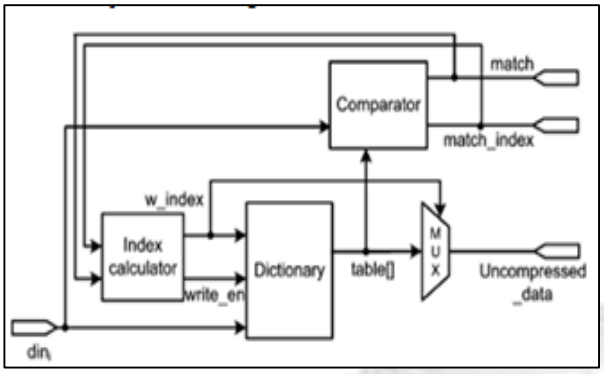


Fig. 4 Block diagram of the dictionary based compression circuit

4) Slicing

The miss address can also be compressed with the Slicing approach. Because of the spatial locality, the basic blocks are often near each other, which mean 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 an 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. Figure 5 shows the hardware architecture. It has the register REG storing the previous data (dini-1).

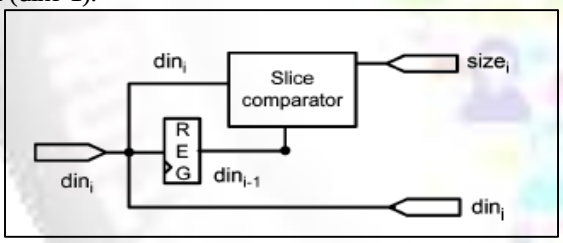


Fig. 5 Block diagram of slicing circuit

5) 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. Figure 5 shows the hardware compressor. The register REG saves the current datum dini and outputs the previous datum dini-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 (sizei) of the difference. Finally, the encoded datum is sent to the packing module along with size i

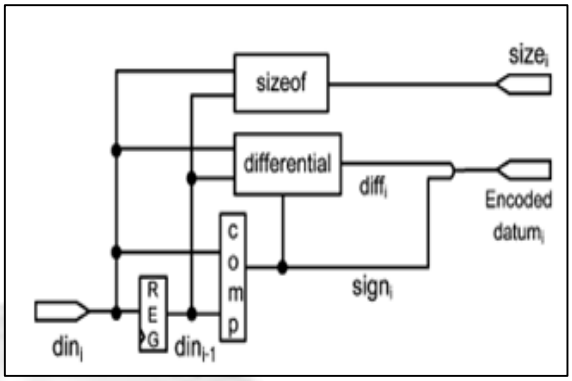


Fig.6 Block diagram of differential compression circuit

6) Control Signal Compression

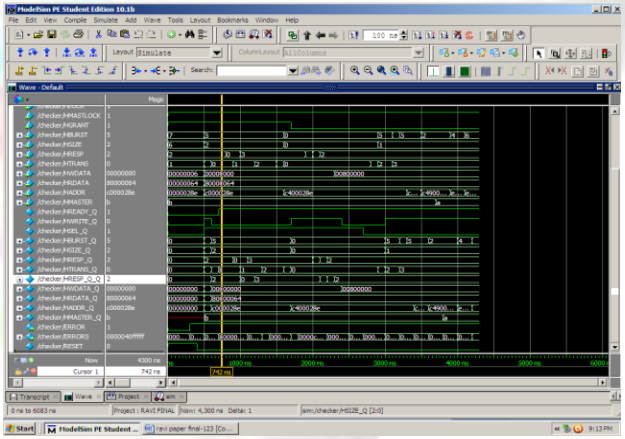
We classify the AHB control signals into two groups: access control signals (ACS) and protocol control signals (PCS). ACS is signals about the data access aspect, such as read/write, transfer size, and burst operations [4s]. PCS are signals controlling the transfer behavior, such as master request, transfer 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.

D. Packing Module

This Module receives the compressed data from the compression module. It processes them and writes them to the trace memory. Packet management, circular buffer management, and mode change control are managed by this module.

IV. COMPRESSION MECHANISM

A. Checker 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.

B. 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.

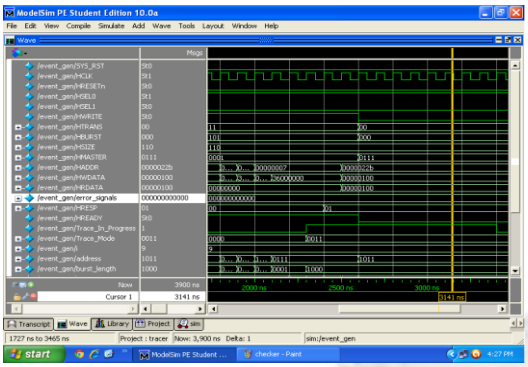


Fig. 8: Event Generation Simulation Result

C. *Abstraction Result*

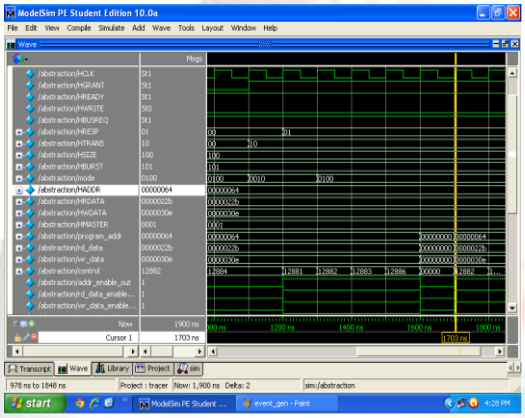


Fig. 9: Abstraction Simulation Result

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.

D. *Compression 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.

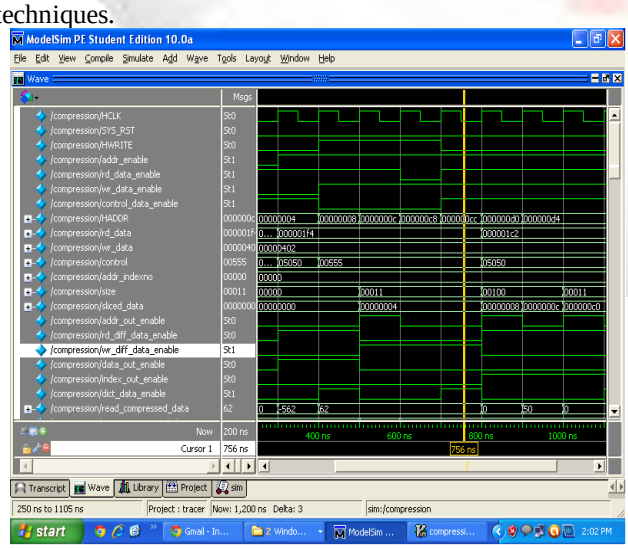


Fig.10 Compression Simulation Result

E. *Packing Result*

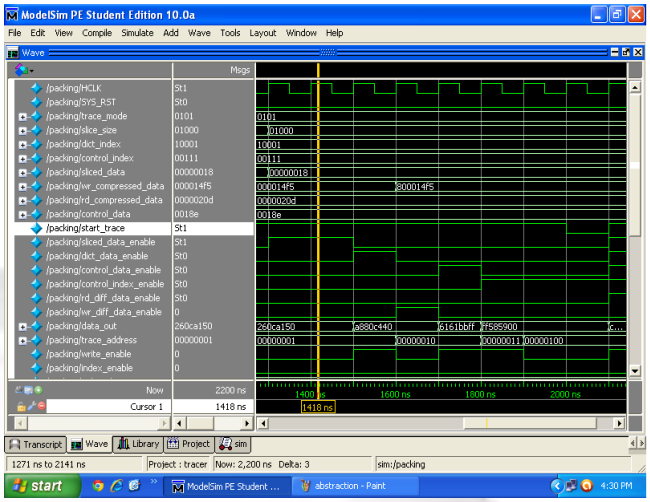


Fig. 11 Packing Simulation Result

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.

V. 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. This is the advantage of the Bus Tracer used in SoC. 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 granularities 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.

REFERENCES

[1] E. E. Johnson, J. Ha, and M. B. Zaidi, ―Lossless trace compression, ‖ IEEE Trans. Compute, vol. 50, pp. 158–173, Feb. 2001.

[2] CoreSight: V1.0 Architecture Specification, ARM.

[3] R. Leatherman and N. Stollon, ―An embedded debugging architecture for SoCs,‖ IEEE Potentials, vol. 24, no. 1, pp. 12–16, Feb-Mar 2005.
[4] ARM Ltd., San Jose, CA, ―ARM. AMBA AHB Trace Macrocell (HTM) technical reference manual ARM DDI 0328D,‖ 2007.
[5] T. A. Welch, ―A technique for high- performance data compression, ‖IEEE Trans. Comput., pp. 8–19, 1984.
[6] Embedded Trace Macrocell Architecture Specification, ARM.
[7] C. MacNamee and D. Heffernan, ―Emerging on-chip debugging techniques for real-time embedded systems,‖ IEE Comput. Contr. Eng. J., pp. 295–303, Dec. 2000.
[8] B. Tabara and K. Hashmi, ―Transaction-level modeling and debug of SoCs,‖ presented at the IP SoC Conf., France, 2004.
[9] ARM, AMBA Specification (Rev 2.0) ARM IHI0011A, May 1999.
[10] C. C. Wang, AHB On-Chip Bus Protocol Checker, 2007
[11] Fu-Ching Yang, Cheng-Lung Chiang and Ing-Jer Huang, ―A Reverse-Encoding-Based On- Chip Bus Tracer for Efficient Circular-Buffer Utilization‖, IEEE Transactions on. Very Large Scale Integration (VLSI) Systems, vol 18, Issue: 5, p732 – 741, May 2010.
[12] Y.-T. Lin, W.-C. Shine and I.-J. Huang, ―A multi-resolution AHB bus tracer for read-time compression of forward/backward traces in a curcular buffer,‖ in Proc. Des. Autom. Conf. (DAC), Jul. 2008, pp. 862–865.
[13] AMBA Navigator Spec Sheet, First Silicon Solutions (FS2) Inc.