AUTOMATED TOOL TO MEASURE BANDWIDTH AND LATENCTY OF SoC DURING PRE AND POST DEBUGGING.

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

MASTER OF TECHNOLOGY

IN

VLSI DESIGN & EMBEDDED SYSTEMS

Submitted by:

MANISH KUMAR

2K17/VLS/12

Under the supervision of

Dr. A.K. SINGH

Associate Professor

ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

2017-2019

ELECTRONICS &COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, (MANISH KUMAR), Roll No. 2K17/VLS/12 student of M.Tech (VLSI & Embedded systems), hereby declare that the project Dissertation titled "**Automate tool to measure bandwidth and latency of SoC for pre and post debugging**" which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi **(Manish Kumar)**

Date:

ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "**Automated tool to measure bandwidth and latency of SoC for pre and post debugging**" which is submitted by Manish Kumar 2k17/vls/12, Electronics & Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date:

(Dr. A.K. SINGH)

Associate Professor

Deptt. of ECE

DTU

ACKNOWLDEMENTS

Every endeavor needs deft guidance and helping hands for its accomplishment, and I was fortunate to have both. I wish to express my indebtedness to my mentors, **Dr. A.K. Singh(Associate Professor)** and **Mr. Pradeep Venkatsva**, without whose guidance, this work would not be possible. I would like to thank my internal guide at DTU, **Dr. A.K. Singh(Associate Professor)**, for his kind guidance, motivation and for providing unconstrained freedom to explore. I wish to express my gratitude to my mentor and manager at Qualcomm(Bangalore), **Mr. Pradeep Venkatsva**, who was always lucid in his explanations, dexterous in his guidance, and generous with his care and support.

I am thankful to **Qualcomm(Bangalore)**, for providing the internship opportunity, which is the backbone of this work. I would like to thank the team members of System Performance Team at Qualcomm.

Finally, I would like to thank my parents, family, and friends, for bestowing their love, patience, understanding, and care, which kept me going.

Date: **Manish Kumar**

ABSTRACT

Due to the increasing design size and complexity of modern Integrated Circuits (IC) and the decreasing time-to-market, debugging is one of the major bottlenecks in the IC development cycle. This project gives a generalized approach to automate debugging which can be used in different scenarios from design debugging to post-silicon debugging. The approach is based on python based an automation tool. This GUI based SoC traces debugging are proposed as an enhancement reducing debugging time and increasing diagnosis accuracy. Here earlier debugging time is reduced by 6 times to the previous debugging method. Here python is used to do scripting for this tool, this tool is graphical user interface and for designing this GUI python is used. Here recent version of python is used i.e. Python 3.7.2 and here simulator tool TRACE32 is used. The experimental results show the effectiveness of the approach in post-silicon debugging.

CONTENTS

Candidate's Declaration..i

Certificate..ii

Acknowledgement...iii

Abstract..iv

Content..v

List of Figures...vii

List of Tables...ix

List of Symbols, abbreviations...x

CHAPTER 1 INTRODUCTION...1

CHAPTER 2 LITERATURE SURVEY...3

2.1 Overview..3

2.2 Debugging..4

2.2.1 Pre Silicon Debugging...6

2.2.2 Post Silicon Debugging..10

2.3 Debugging Techniques...11

2.3.1 Scan Chain Debugging Techniques...11

2.3.1.1 Scan Based DFT Methodology...12

2.3.1.2 Scan Based Debug Methodology...13

2.3.1.3 Scan Based Debug Architecture..13

2.3.2 Embedded Logic Analysis...16

2.3.2.1 EDM Overview..16

2.3.2.2 EDM Architecture..17

2.3.2.3 EDM Related Work..29

2.4 Concluding Remarks...20

CHAPTER 3 SoC ON CHIP DEBUGGING..22

3.1 Overview..22

3.2 System On Chip...22

3.3 On Chip Debugging..23

3.3.1 Components Of On Chip Debugging..25

3.3.1.1 JTAG...25

3.3.1.2 Debugging Board...26

3.3.1.3 TRACE 32(simulation tool)..26

3.3.1.3.1How To Use TRACE32(simulation tool)...27

3.3.1.3.2 Tracing Feature...30

3.3.1.4 Various Scripting Language..31

CHAPTER 4 AUTOMATED ON CHIP DEBUGGING TOOL................32

4.1 Overview..32

4.2 Proposed Work...33

4.2.1 Parser Automation Tool..33

4.2.1.1 Overview...33

4.2.1.2 Theory...34

4.2.2 Data Analysis Tool...44

4.2.2.1 Overview..44

4.2.2.2 Theory..45

4.2.2.3 Calculations..47

CHAPTER 5 CONCLUSION AND FUTURE SCOPE..............................51

REFRENCES...53

LIST OF FIGURES

Fig 2.1 Debug process in ASIC design flow

Fig 2.2 ASIC flow cycle

Fig. 2.3 Basic Principle of VLSI Circuits Debugging

Fig. 2.4 Basic Principle of BIST

Fig. 2.5 BIST with Multiple Scan Chains Architecture

Fig: 2.6 Single Scan Chain Architecture

Fig. 2.7 Scan-based Debug Architecture

Fig.2.8 Debug Flow during Post-silicon debugging

Fig. 2.9 Debug Framework of Embedded Logic Analysis

Fig.2.10 Circuit under Debug with Centralized Debug Module

Fig. 2.11 Embedded Debug Module Architecture

Fig. 2.12 Triggering Conditions Flow with Segmented Trace Buffer

Fig. 2.13 Multi-core Debug Architecture with Centralized Trace Buffer

Fig. 2.14 Multi-core Debug Architecture with Multiple Trace Buffers

Fig. 3.1 On chip debugging environment

Fig. 3.2 Main window of Trace32 application

Fig.3.3: Trace setup window

Fig. 4.1 Parser Automation Too

Fig. 4.2 How python connects database to system

Fig. 4.3 Master attached with ETM

Fig. 4.4 core clock enable script

Fig. 4.5 Debug Trace Subsystem

Fig. 4.6 ECT system

Fig. 4.7 Parsed data

Fig. 4.8 Data Analysis App

Fig. 4.9 Selection of file to be analysed

Fig. 4.9 Analysis of two column data simultaneously

Fig. 4.10 Latency Measurement

Fig. 4.11 FPS(frames per second)

Fig. 4.12 Timeline

Fig. 5.1 Comparison of debugging time for both manual and automation debugging

LIST OF TABLES

Table 5.1 Manual and Automated debugging comparison

LIST OF ABBREVIATIONS

GUI - Graphical User Interface

DSS - Debug sub System

SoC - System On Chip

IC - Integrated Circuit

LAT- Latency

BW - Bandwidth

RTL- Register Transfer Level

HDL- Hardware Description Language

LAL- Lower Abstraction Level

CAD- Computer Aided Design

CUT- Circuit Under Test

ATE- Automatic Test Equipment

ATPG- Automatic Test Pattern Generator

DFT- Design For Testability

SFF- Scanned Flip Flop

BIST- Bilt In Self Test

SR - Shift Register

TPG- Test Pattern Generator

SA - Signature Analyser

CHAPTER-1

INTRODUCTION

Now in the VLSI field role of debugging and validation are increasing, as day by day complexity of circuit is increasing whereas size of IC is decreasing so here debugging and validation plays an important role . Also we should be aware that it is very difficult to go for complete debugging and validation at the design level .So here automated debugging plays an important role in debugging, it is very needful to reduce time to market for the VLSI products. As design process is important but it will not be recommended until and unless its functionality is verified and for this validation and debugging plays an important role. If there will be any behavioural depended bug then we can find it through pre silicon validation and we forward it for the fabrication, before fabrication we must be assure about the its proper functionality because fabrication is the most costlier process of the IC designing, so it is very important, once IC get out from the foundry it then go for the post silicon validation, because we cannot go directly for the mass production until and unless we validate it after the fabrication, because there is the chances of bug during the fabrication process so we have to recover it before the mass production of the IC.

As in this competitive environment of VLSI industry the most important parameter of their basis of competition is the time to market, here every industry is planning to launch their product as soon as possible. In such pressure environment it is time taken for IC design flow plays an important role and this IC design flow validation is the most time consuming activity. So here we are decreasing this validation time with the help of "automated debugging tool" for validation.

Complete validation process is not a single step process it is a multi step process that requires lot of time for each step process so if we go for multiple step debugging process one by one then it will take lot of time to do complete debugging, so here automated debugging process is come in picture that reduced this multiple steps debugging in less no. of steps even here we use GUI for debugging process that reduced debugging time by 6 times of the older debugging process.

When chips are in the form of codes (netlist) then it is very simple to debug that, as it is simple to use test vectors on these codes (netlist), even here we can use clock in the form of codes but it is not so simple in post silicon, we know post-silicon debugging has become an important step in the design process of system on chip devices for finding and eliminating design errors which have left during pre-silicon validation/debugging. To address the fixed and limited observability constraints of the circuits in the process of post-silicon debugging, embedded logic analysis methods are implemented in order to examine the in circuit nodes at high speed and in the real-time. In this process, I propose novel on-chip debug process that is designed with the help of python language, here we designed two automated tools to get latency (lat.)and bandwidth (bw.) of SoC circuit.

CHAPTER-2

LITERATURE SURVEY

2.1 Overview

If we go through the ASIC flow design, we can see that how customer specified idea converts into a chip. Here is the flow chart of ASIC flow design

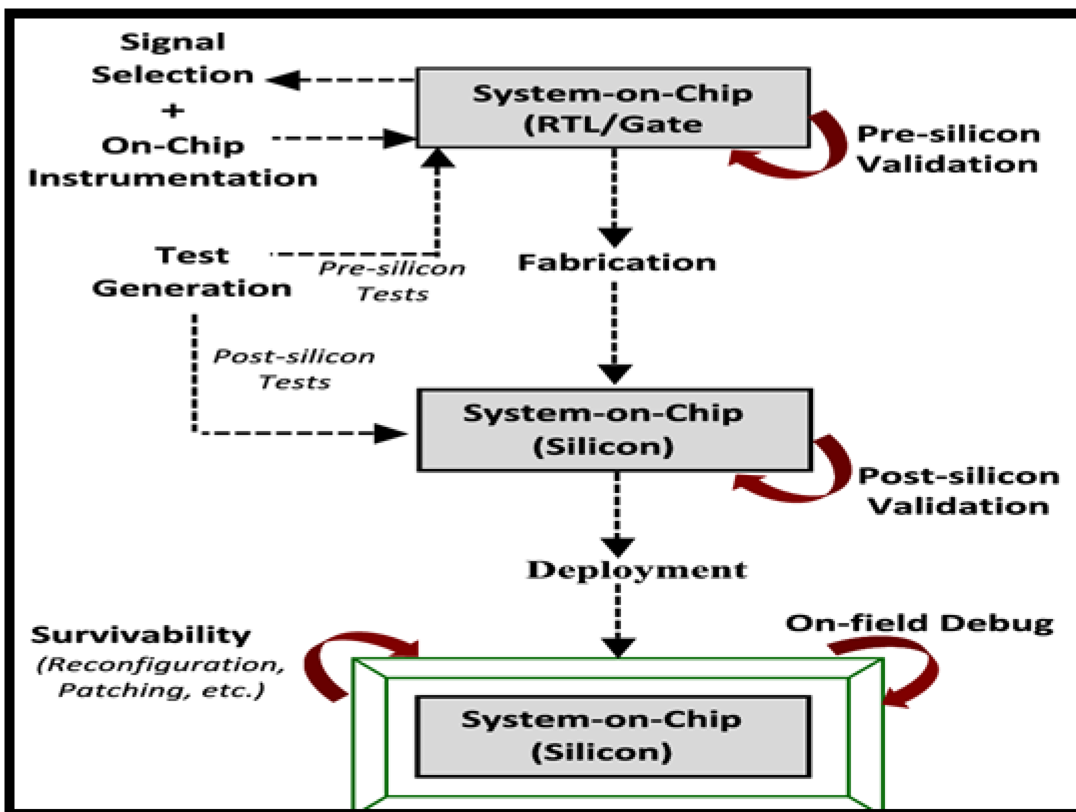


Fig 2.1 Debug process in ASIC design flow[1]

Debugging process plays an crucial role in IC designing process, in the ASIC flow designs we go two times through the debugging process, first one is before the chip fabrication that is called "pre silicon debugging" and second one is after the fabrication that is called "post silicon debugging".

2.2 Debugging

The design flow of VLSI circuits consists of three steps[2]:

a) Specification,

b) Implementation

c) Manufacturing

The specification step by vendor generally describes the expected behaviour and functionality of the VLSI circuits. Here design behaviours are described in a high-level language such as System C or in any hardware description language (HDL) such as Verilog, System Verilog or UVM. The register transfer level(RTL) abstraction are used in HDLs to describe the circuit's behaviour as a combination of transfer functions which shows the flow of signals between the different blocks of the circuit, and the different logical operations should performed on those signals. The operation of formation the circuit model from a higher abstraction level (HAL) to lower abstraction level (LAL) is known as synthesis. The logic synthesis performs the transformation of the HDL design into the generic design library of various components followed by adequate optimization and connecting into the gate level components. The next step in design flow is to design a circuit behaviour into a gate-level netlist that represents the logical behaviour and functionality of the circuit. The synthetization can be designed based on various design techniques either full custom design or semi-custom design. If we take the case of full custom design technique, designer makes the layout of the circuit and different interconnections between the different circuit components in order to enhance the performance of the circuit and optimize its area. On the other hand, if we take an example of semi-custom design technique, the synthetization is performed using different standard cell libraries or different gate arrays through different computer aided design (CAD) tools[3].

These tools implements the design from the RTL abstraction level (RAL) into the gate level netlist and perform the conversion from the gate level design into a physical layout design. The last step in the design flows of VLSI is the steps of creating the integrated circuit (i.e., fabrication). The fabrication method is a multiple-process sequence of photographic and chemical and different processing steps, in which the IC are sequentially created on a wafer made of semiconductor material based on the unique transistor technology (e.g., 90 nm) .We know that silicon is the most commonly used semiconductor material in VLSI circuits and CMOS are the main type of transistors.

The increasing intricacy of VLSI design in semiconductor industry makes pre-silicon debugging, manufacturing test and the validation of the first silicon an important step in the design flow.

During the process of debugging, test vectors are generated and it will check at different masters of chip and the values at different master's match with the preferred values, if it match with the preferred value then its ok otherwise there will be bug in the chip.

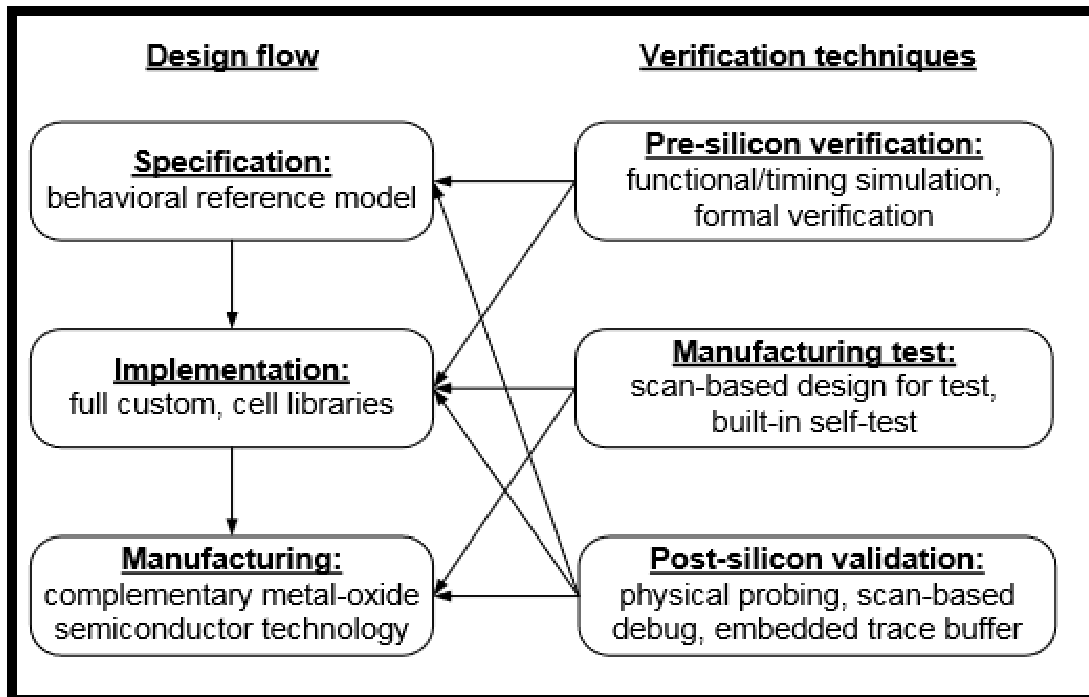


Fig 2.2 ASIC flow cycle[4]

As we know that in IC industry the step more valuable is "fabrication", before fabrication we get the chip in the form of GDS II form that we send to fabrication lab for silicon development that is the most costly part of chip development, so it is very necessary to check its functionality before it deliver to foundry, it is necessary to make the chip bug free before it deliver to foundry because once bug will be found after hard silicon then it will be big loss for the IC vendor.

In the same way as pre silicon debugging is important, post silicon debugging is also important, when foundry get GDS II it first develop as hard silicon in the form of CDP and MDP that dispatch to company for all post silicon testing ,after the all post silicon testing it further handover to IC vendor for the bulk production.

Post silicon debugging is also important because we know that there are enormous steps in fabrication and all these steps are micro level analysis so a minute mistake and bug in fabrication can cause major loss so it is need to debug IC after fabrication before it is handed over to IC vendor for the bulk production.

Here we had seen that the role of debugging is important in whole IC design process, so there are two debugging in whole IC design process

a) Pre silicon Debugging

b) Post silicon Debugging

2.2.1 Pre Silicon Debugging

As the SoC circuit complexity increases, the shrinking time to come to market it shows that the design validation is the biggest challenging activity in the IC design flow. If we talk about the result, pre-silicon debugging techniques like formal debugging and simulation tools are used largely to find out design or circuit errors before the circuit or design gets manufactured. The motto for these pre-silicon debugging methodology is to verify the completed design behaviour against its noted design constraints. Simulation techniques are used to analyse the circuit behaviour at various levels of abstraction. For example, a design can be simulated at a higher behavioural level where it does not contain complete timing information. The more complete information about the circuit level model has (e.g., logic-level netlist), the more time is required to verify its functional behaviour.

Thus, simulation for complex VLSI designs is practically not feasible. Formal debugging, is a complementary technique for simulation, it is the process of mathematically verify the correctness of the design with respect to different formal specification[5] or property. Basically there are two main approaches to formal debugging: first one is equivalence checking and second one is property (also known as model) checking. Purpose of equivalence checking is to verify that two given designs are functionally similar by comparing one design model called the reference design model with respect to the targeted design model. Property (or model) checking procedure is used to analyse a given design for the matching of its properties which are designed in a dedicated debugging language. Model checking is a common using approach in the case of formal debugging of concurrent systems. Formal debugging has proven its significance in finding the logic bugs during pre-silicon debugging of the processor.

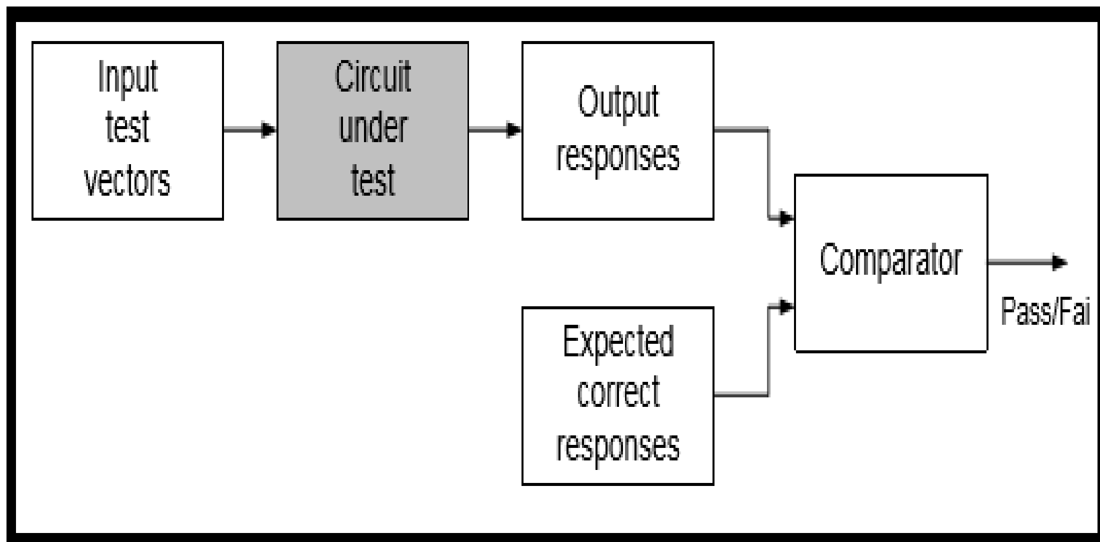


Fig. 2.3 Basic Principle of VLSI Circuits Debugging[6]

Debugging is not limited up to only analysis of different bugs only but we also assure about the functionality and manufacturing defect of the circuit. Like we go across different types of system performance tests and we analyse the outputs w.r.t our design constraints and see the deviation of debug data from the design constraint data and at last we find out the cause of this deviation. we remove the cause and again repeat this this task until we get the debug data value similar to our designed constraint value.

System performance test has become an important step in the implementation flow of the VLSI circuits to ensure that the functionality and behaviour of the designed circuits matches to its implementation. It is used to find out the physical defects that leads to faulty and undesired behaviours in the fabricated circuits. Figure 2.4 shows the basic principle of manufacturing test. The whole test process is commonly controlled by a test instrument called automatic test equipment (ATE)[7]. Circuit under test (CUT) is the complete chip or some random part of the chip to which the unique input test vectors are applied. The input test pattern vectors are random binary patterns applied to the input of the testing circuit. These random test pattern vectors are generated using ATPG(automatic test pattern generation) algorithms.

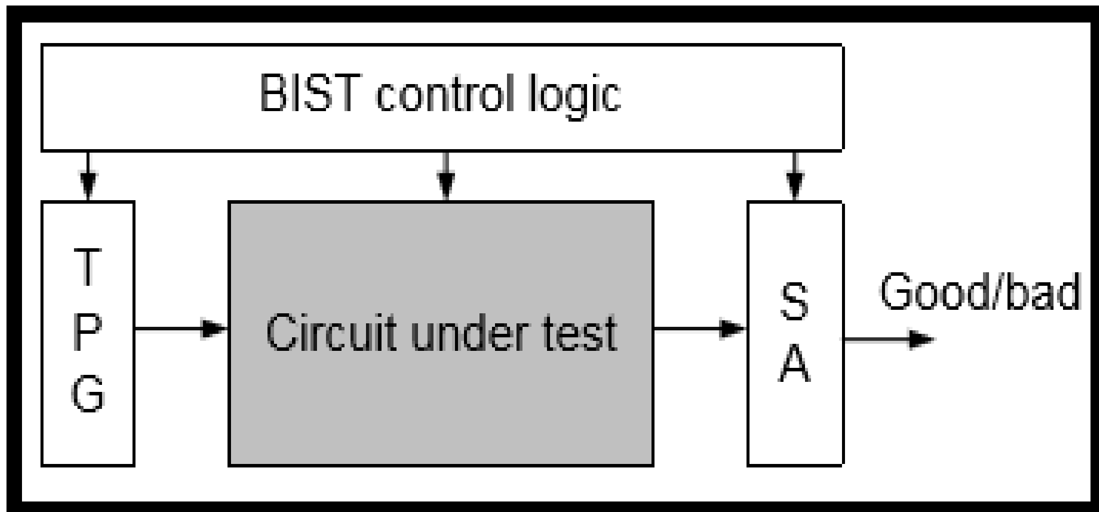


Fig. 2.4 Basic Principle of BIST

The circuit is declared as a fault-free circuit if the circuit observed output responses meet to the expected correct responses. If the circuit under test failed the test, the backtracking diagnose process started to diagnose and identify the basic root cause of the fault. The CUT can be tested using structural, behavioural or functional test.

Here functional test requires a full unique set of the test patterns to verify all entries of the truth table.

System performance is not limited to only frequency, bandwidth and latency but it is applicable for

a) Hardware system performance

b) Thermal system performance

c)Power system performance

d)Automation system performance

e) CPU system performance

f) Multimedia system performance

Along with these there are many system performance analysis we do during the debugging and we match the result of these analysis report to the design constraint and if we get any mismatch then we go for the reverse engineering and find out the cause of that. We repeat this activity until we clear this discrepancy.

Now we will see how achieving high fault coverage during manufacturing test we get through the use of design for testability (DFT) circuitry that increases the controllability and observability of the test circuit’s intermediate nodes.

We mostly use Scan based DFT technique, in which all or some part of the sequential elements (i.e., flip flops) of the circuit are changed with scanned flip-flops (SFFs). During the test mode, these SSFs form one or more long shift registers, these are called scan chains.

The unique input test patterns are applied to the input of these scan chains serially which are connected to the primary inputs of CUT. The states of these SFFs are noticed by shifting out the values of the shift registers (SR) whose outputs are linked to the primary responses of CUT(circuit under test). Other DFT method is BIST(built-in self-test). BIST can be preferred as a low cost testing process when it is compared to ATE-based testing. In ATE-based testing, the unique test stimuli are applied to the chip pins from the ATE and the test responses that we get are shifted out and compared with the desired responses stored in the ATE memory bank.

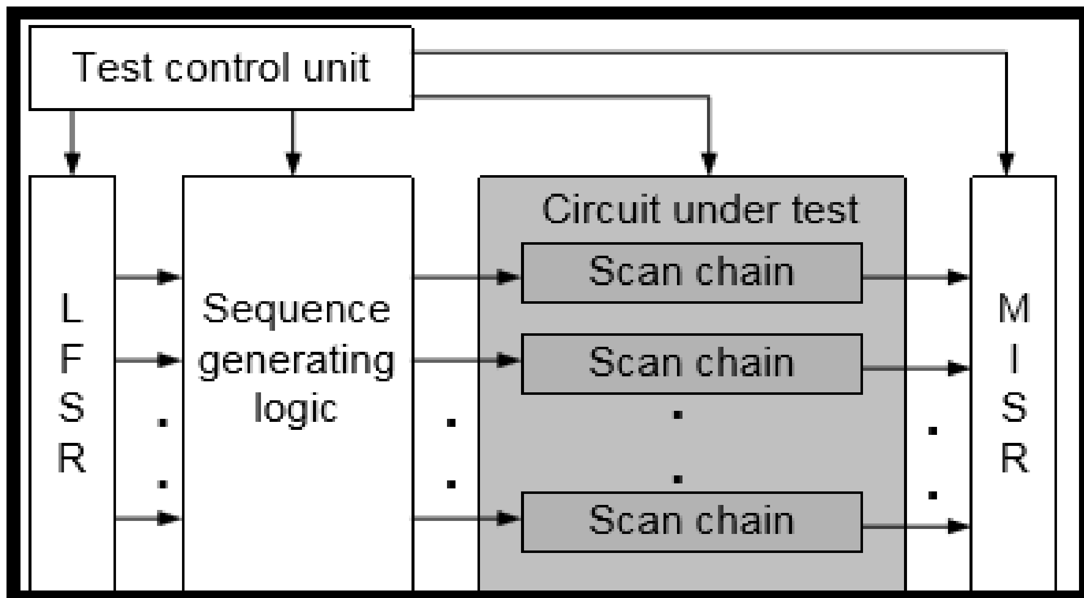


Fig. 2.5 BIST with Multiple Scan Chains Architecture

Due to the requirement of large memory and bandwidth for testing state-of-the-art SoCs, BIST can be used as another approach to external test for very complicated VLSI circuits. As shown in Figure 2.4, a test pattern generator (TPG)[8] generates an unique set of test stimuli and a signature analyser (SA) compares and analyse the test responses and makes the decision it is a good or bad chip. The control block of the BIST controls the test sequence. During BIST approach, a LFSR(linear feedback shift register) can be used as the TPG(test pattern generator), and for the SA we ca use a single input signature analyser (SISR) or a multiple input signature analyser (MISR). BIST methods can be divided into pseudo-random BIST and deterministic BIST.

Therefore, to make sure the product does not contain any design bugs (which are present in the netlist and hence affect every fabricated device), manufacturing test must be complemented by post-silicon validation/debugging(or silicon debug) in order to identify the design errors that have not been captured during pre-silicon debugging/validation.

2.2.2 Post Silicon Debugging

Here we had seen the overview of pre silicon debugging/validation, if we go through the post silicon debugging/validation we will notice that this post silicon debugging is more time consuming as compare to the pre silicon debugging. Here for the pre silicon debugging we get the RTL codes that we burn on the FPGA board and then after that FPGA board will act like a desired circuit that we wish to implement, Here we have to just only upload the different script on that FPGA board and we have to analyse the responses of different script.

But here in the case of the post silicon debugging we get the hardware kit from the foundry either in the form of CDP or MDP, then here it is in the form of hard silicon so here it has to be interfaced with the system then we have to setup the serial communication with the PC(system) through the JTAG and then we start our analysis through the same way we use to do during the pre silicon. There are different ways that we go for the post silicon debugging:

Scan based post silicon debugging: In this debugging technique this is the role of scan chain is there, in the scan chain based debugging methodology, here scan chain is connected all around the processor once the test pattern vector is applied to the input of the scan chain then it gets captured at the first scan chain shift register then after getting the clock that test pattern get shifted to the next shift register and this process gets continue. Here major drawback we face is that when we have to dump the output scan chain data at that time we have to stop the processing of the CUT(circuit under test).

It was the limitations for scan based debugging technique then we started for trace buffer based debugging. The target to design design-for-debug (DFD) architectures and methods that make possible the recognisance of the different functional bugs, when the debugging process is done at high speed and in the system.

Even though they have been debugged for different functional bugs, here the methods and trace buffer architectures can be taken as a front end to many physical probing methods to help in electrical bugs debugging process.

As explained, they are induced by the finite real time observability, as well as by the problem of handling with blocking bugs during post silicon debugging process

2.3 Debugging Techniques

The main problem in post silicon debugging is that its internal circuit's signal observability is limited. As the recent increments in the physical probing methods, the difficulty of condition of devices needs a localization step to validate the cause of the failure of IC working. we find out the step to recover this and we say this methodology as logic probing technique, relates the simulation data information to what we get in the silicon in order to recognize a subset of different circuit node points that is required to be probed physicaly. This we get by summing DFD hardware to improve the internal nodes observability and enhancing the debug process. The DFD methods are not only useful in getting the different electrical bugs but they are also important for locating the various functional bugs.

To identify the various design bugs during the process of post silicon debugging, DFD architecture is arranged to start capturing and tracing the various internal state of the CUD. Two complementary DFD methods have been used to keep eyes on the system's state: scan chain based debug method on run-controls different debug approaches, and embedded logic analysis technique that is based on real-time trace profiling approach. These different debug methods are essential to minimise the search work for design bugs by identifying both when (temporally) and where (spatially) a bug occurs.

2.3.1 Scan Chain Debugging Technique

The scan technique is the most frequently used method in manufacturing test for allowing the examination of intermediate circuit's full state. Here scan chain methodology is used for debugging highly complex or bulkier ICs. To differentiate between post-silicon debugging and the usage of scan chains in manufacturing test, we first demonstrate their working during the various manufacturing test procedure.

2.3.1.1 Scan Based DFT Methodology

When doing scan chains in scan-based DFT technique, the different sequential elements (i.e., flip flops, latches etc.) are connected to each other in such a way that permits two types of operation. During the test mode, the scanned flip-flops are configured again as one or more shift registers (known as scan chains registers)[9]. The unique input test vectors are serially put at the input of the CUT. Once a vector is loaded successfully, the CUT is allowed to work as in its normal mode where the circuit output response is stored in the flip-flops. Then after, the stored response is shifted towards out and these values are compared to the expected response values. Figure 2.6 illustrates the architecture block diagram of a single scan chain. Each scanned flip-flop (SFF) is made up of a 2-to-1 multiplexer and a D flip-flop. A SE (scan enable) signal selects one between two data inputs: the scan-data input SD and the original data input D. The primary inputs (PI)/primary outputs (PO), they are directly interfaced to the digital logic blocks like combinational circuit. The construction of scan chain is build up by connecting the data output Qout of one SFF to the SD signal of the next SFF, as illustrated in the Figure 2.6. Here is the explanation of the scan based DFT procedure during the test process as follows: (i) put scan enable signal (i.e., SE is set to 1), and upload the unique test vector value through the scan input (SI); (ii) put scan enable signal (i.e., SE is set to 0), and apply a clock cycle in order to store the circuit response from the combinational logic outputs value in SFFs; and (iii) put scan enable signal (i.e., SE is set to 1), and scan out the test responses value. While the test response value is shifted out, then simultaneously a new test pattern vector is moved in. If we talk about the testing time then it depends on the test pattern counts and the scan chain length. Multiple scan chains technique can be used to reduce the time for test application. Generally each scan chain has its own fixed SI and SO pins and hence the test response or data can be shifted in/out in parallel direction. It is necessary to remember that most of the blocks of a SoC are designed with full chip scan capability, where it’s all inside sequential elements are replaced by SFFs, in order to get high and best fault coverage.

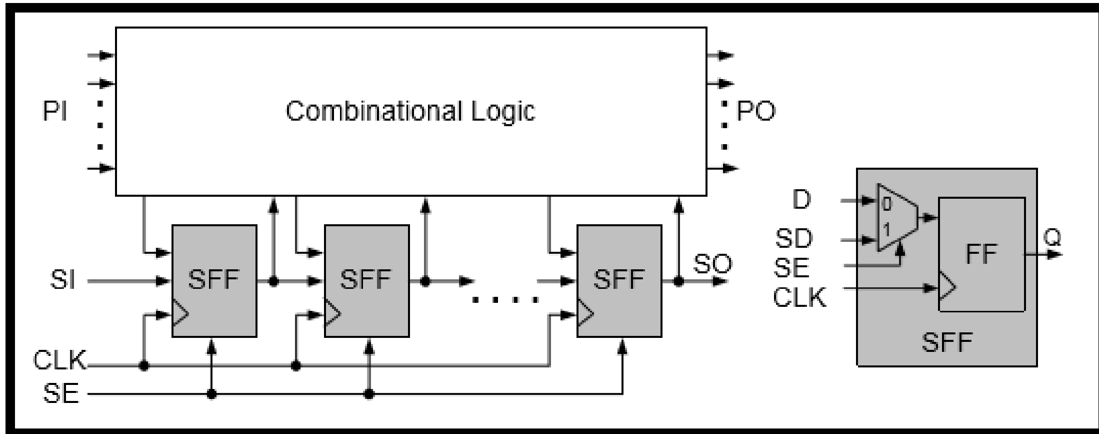


Fig: 2.6 Single Scan Chain Architecture

2.3.1.2 Scan Based Debug Methodology

Here in the case of scan based debugging methodology, the output data gets out from the scan chain that are out from the scan chain are stored and moved to the debugging software, this activity happens at the specific triggering of the event. The process of sending data to the debugging software is called as the scan dump. This debugging software will share scan chain output data to the CUT(circuit under test) that is get connected to the ATE(automatic test equipment). When the debugging board gets trigger for the debugging then the process going on is keep on hold and the debugging will take place here.

2.3.1.3 Scan Based Debug Architecture

Figure 2.7 shows the scan chain based architecture for the post silicon debugging technique. Note that the combinatorial logic surrounded, around the each scan chain, is not shown in this figure. There are two modes of operation: the first one is the test (or debug) mode on ATE and the second one is the debug mode on prototype target application board. During the test mode, the concat signal is put as 0 (i.e., Concat = 0) and the scan is set for manufacturing test on ATE, where the scan chains are traced in parallel through the different functional pins. As we had seen earlier, some part of electrical debugging is done on the ATE because of the ability to adequately control the frequency and voltage. During debug mode where the CUD is debugged inside the system, once the breakpoint in debug module finds the trigger event, the clock controller stops the all on chip various clocks.

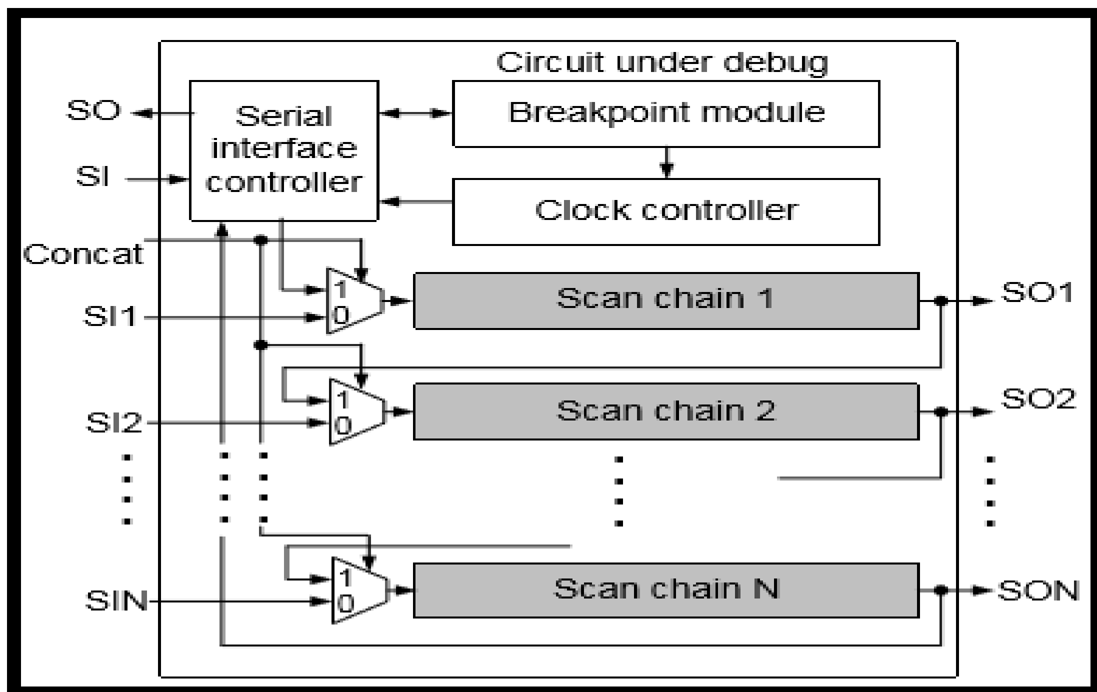


Fig. 2.7 Scan-based Debug Architecture[10]

Thereafter, the concat signal is assigned to 1 (i.e., Concat = 1) and hence the multiplexers concatenate with all the scan chains along one long shift register. The data of the dedicated scan chain is then moved out through the SO pin through the serial data interface (e.g., JTAG). The data that received can be loaded to the chip through the SI pin when moving out of scan chains data done. This process is repeated until all complete intermediate present scan chains have been scanned out. After analysing the complete circuit's state against the expected one state data, it is the option for the debug engineer that can again program the all breakpoint debug module to direct another trigger condition to get and analyse the chip's state at some other particular event. In any case of mismatch, the engineer attempts continuously to focus on the time and location of the occurrence of error by initializing another trigger situation and analysing the information we stored/captured. This process is will be in progress until the main cause of bugs is we get. Then, the bug that we get is removed or fixed in the original previous design and new silicon is manufactured. Figure 2.8 shows the repetitive debug flow during post silicon validation process as described above.

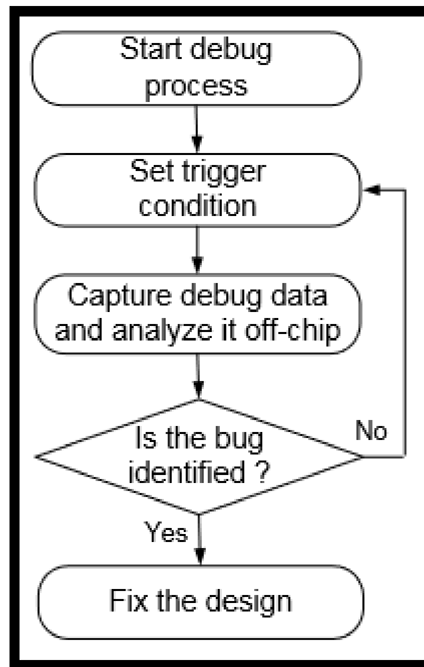


Fig.2.8 Debug Flow during Post-silicon debugging

It is an essential to note that pausing a system with various multiple clock domains may lead to invalidation of data where some clock domains may use data from other clock domains that have been already stopped because the on chip clocks cannot stop at the same time. A data invalidation detector has been designed in to find out the scan elements (i.e., flip-flops) that capture and store the data that is not valid, and therefore the data of these elements cannot be used for comparing with the simulation data model. The clock controller circuit can allow other control functions circuit such as multiple operations and single step. The single step operation is done as an extension to the debug scan out operation, where the design circuit is enabled to work in the normal mode and the scan enable on the scan able flip-flops is not started. The continue operation is done by removing the various breakpoint mechanisms that releases the clock gating, so that chip operation can start again. It is necessary that the frequencies and phases of the on chip clocks are the similar as at the time the chip was paused working; otherwise, the operation started again will not be the similar as in the case of a chip that has not been stopped.

Because pausing the execution process during scan dump will end the system's state, getting debug data in continuous clock cycles cannot be get by using only the all the available scan chains.

2.3.2 Embedded Logic Analysis

Embedded logic technique is a DFD method used for enhancing the observability of intermediate signals of SoC design by getting debug data on chip into embedded trace buffer. This is done through a DFD event detection technique that can be done to a unique trigger event at which the acquisition process stops or start.

2.3.2.1 EDM Overview

Figure 2.9 illustrates the principle of an embedded logic technique debug framework. In this framework, the on chip debug software link with the embedded debug module (EDM) through a serial data interface bus such as the Boundary Scan interface (i.e., JTAG). The embedded debug module contains an embedded trace buffer to provide re observability on the real time for a subset of intermediate signal. As shown in Figure2.10, we can see that an embedded debug module that work as centralized monitors, a limited set of intermediate signals come from various embedded cores and the different interconnect bus of the CUD.

The theme of a centralized debug module has been used for debugging and validation of multi-core processor design. The debug flow during embedded logic analysis technique is explained as follows.

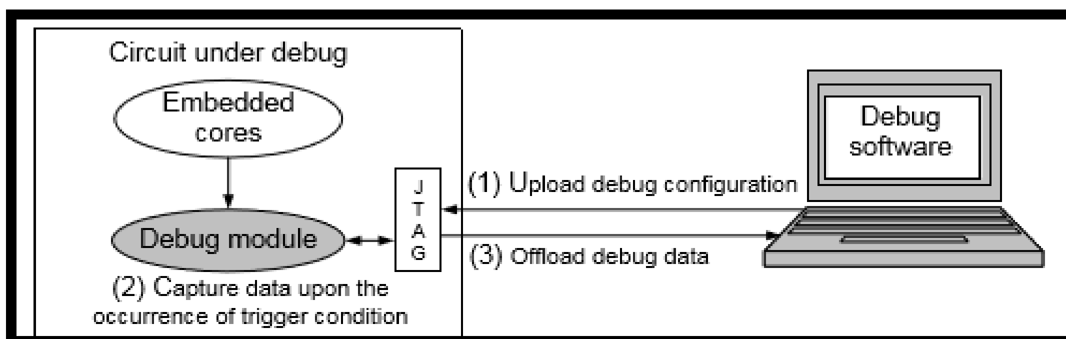


Fig. 2.9 Debug Framework of Embedded Logic Analysis[11]

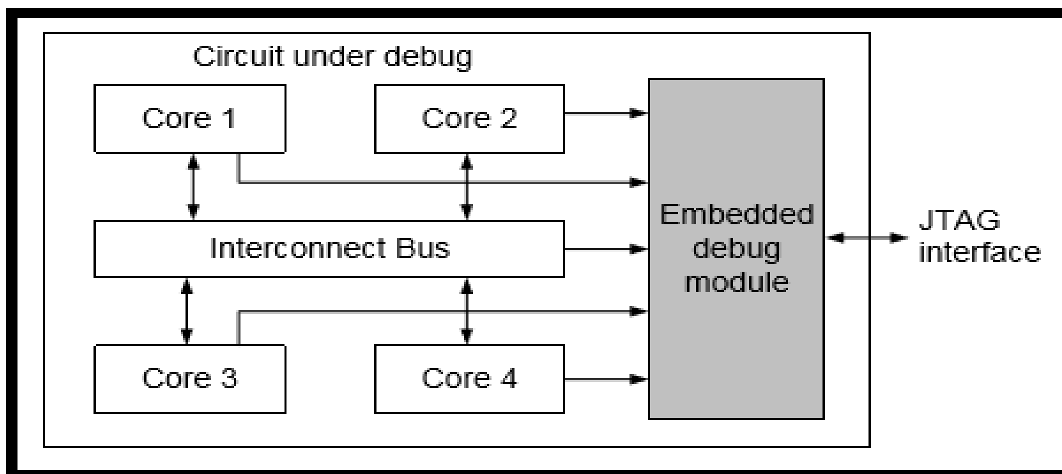


Fig. 2.10 Circuit under Debug with Centralized Debug Module[11]

A debug activity (or session) initiate by dumping the embedded debug module with the various debug configuration module (Step (1)). This configuration is having information about the debug configuration such as the various trigger condition that says the point in time when the capturing of debug data begins, and the control data that says the selection of node points that is needed to be probed. Once the trigger condition happens during the implementation of the on chip application, the debug module will start collecting the desired signals behaviour into an on chip trace buffer (Step (2)). This debug experiment is ended by sending the trace buffer's data to dedicated debug software, where the data that captured is analysed (Step (3)). The debug experiment can be continuously repeated with different debug configuration set until the design bugs are found.

2.3.2.2 EDM Architecture

The main feature of an embedded debug module is to collect the behaviour and function of selected intermediate signals upon the happening of a certain triggering events. This we get using an event detecting that analyse a group of the trigger event signals to find when the data that is to be debug is collected in the trace buffer, we can see this in Figure 2.11. It is necessary to analyse that the event sequencer/detector functionality and behaviour is used in the embedded debug module techniqu[12]e is likely to the one used in that breakpoint module of scan based debug architecture.

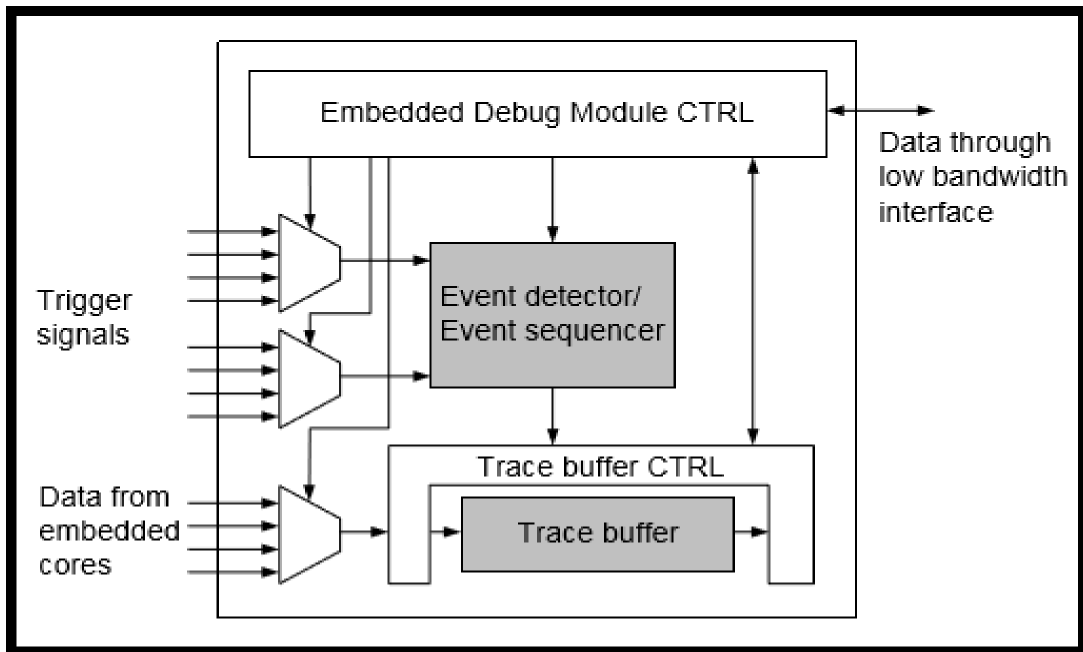


Fig. 2.11 Embedded Debug Module Architecture[12]

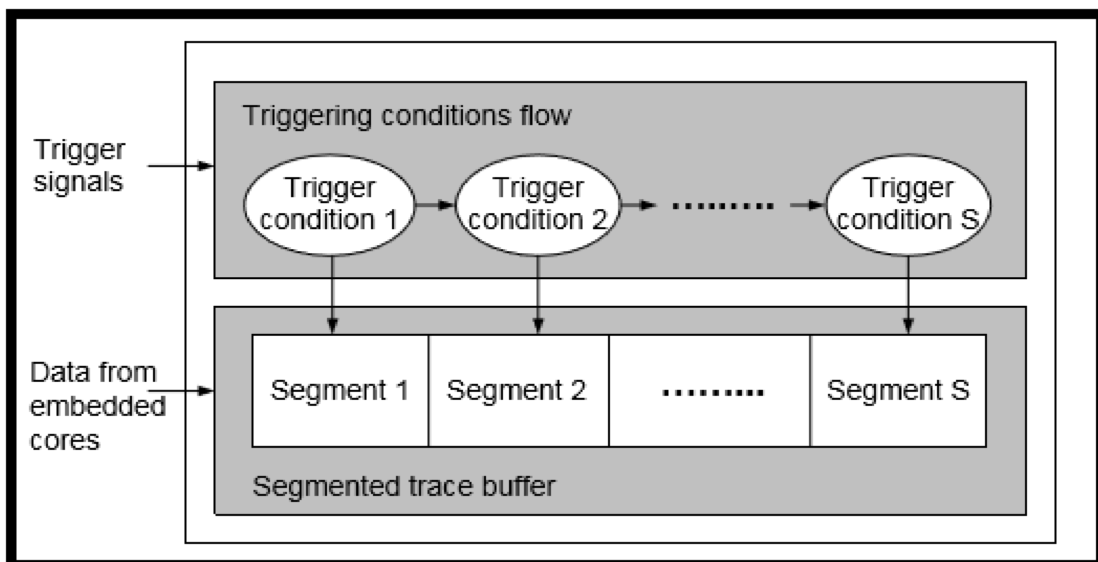


Fig. 2.12 Triggering Conditions Flow with Segmented Trace Buffer[12]

The embedded trace buffer can gives flexible acquisition by assigning it as a segmented buffer. Figure 2.7 shows the segmented trace buffer with S no. of segments.

Here each segments collects the information upon the happening of the corresponding triggering condition event, e.g., Segment 1 begins the acquisition upon the happening of trigger event condition 1. Once the first trigger event situation occurred, the event detector circuit is fixed to monitor the second trigger event condition. The event detection circuit is parallelly dumped with the control data that

shows the required triggering situation until the last triggering condition is done. Each of the segment size can be fixed to work as a circular buffer segment and thus in this case here, the triggering event condition can be used to end the data acquisition. The working of a segment trace buffer described above has been used in debugging FPGA based design.

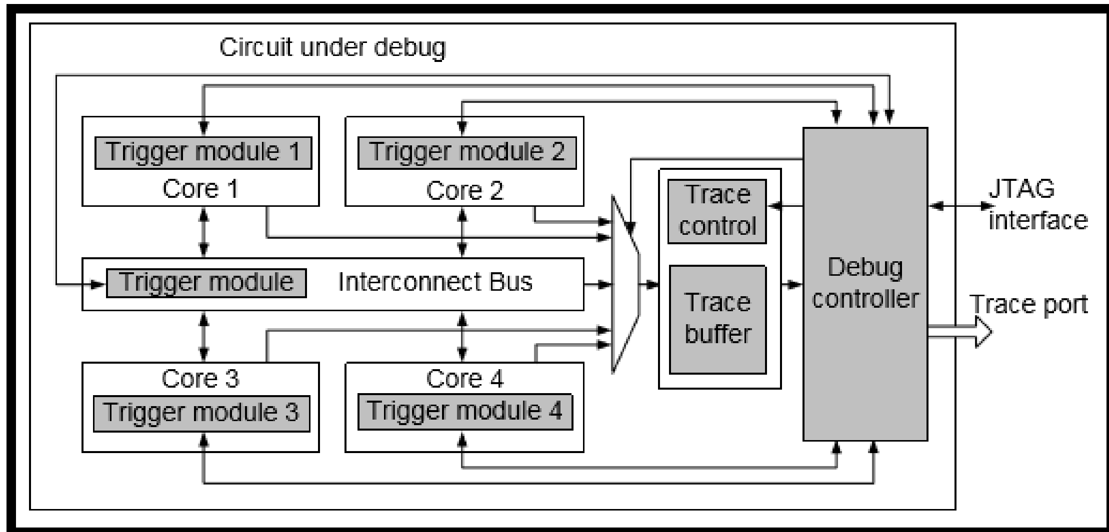


Fig. 2.13 Multi-core Debug Architecture with Centralized Trace Buffer[13]

2.3.2.3 Embedded Logic Analysis Related Work

In order to continuously analyse signals behaviour from various cores in complex and big SoCs, the distributed trigger technique modules are fixed in multi-core architecture. As, two ways can be used to support data tracing from different embedded cores either: distributed or centralized trace.

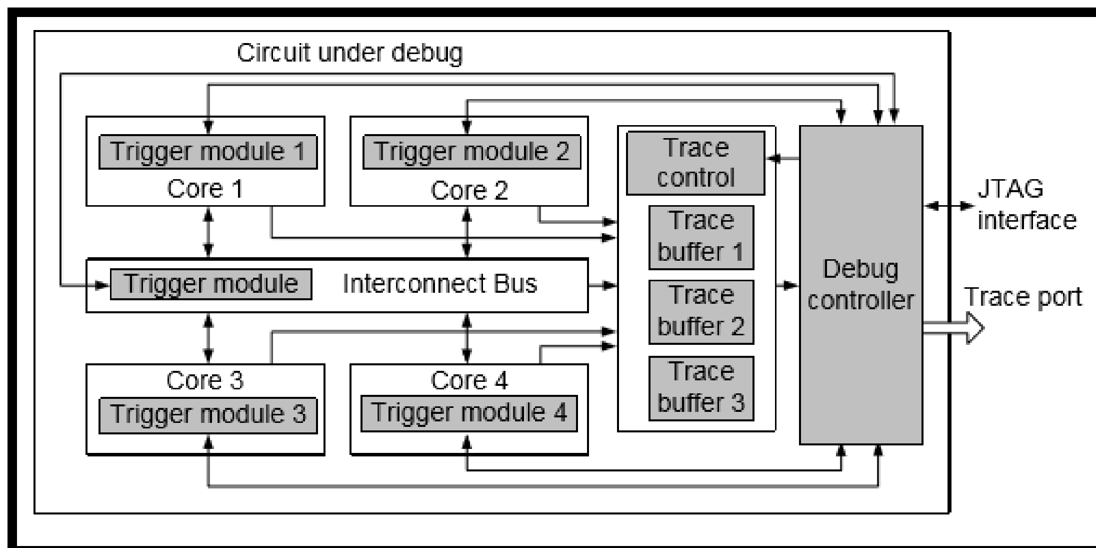


Fig. 2.14 Multi-core Debug Architecture with Multiple Trace Buffers[13]

During the centralized data trace method, one trace buffer is allotted for the complete SoC, where the trace data is captured through a multiplexer. Figure 2.14 shows multi-core debugging architecture model with complete trace buffer. The common trace buffer method has been taken for debugging the Cell processor which contains different nine processing core units. Due to the fixed speed and data bandwidth of the serial data interface, a high speed trace port can be utilised for analysing the data that is stored into the trace buffer register. Hence, the trace port can used to do the debug process by configuring the trace buffer data when running the on chip application and that's why allowing more data to be stored on chip.

This all about the embedded logic analysis, where we had seen how trace buffer based analysis is useful for real time fast on-chip debugging.

2.4 Concluding Remarks

In this chapter, we have discussed the background and the related work of the all existing methods for post silicon debugging. Here we presented two complementary approaches. Here the first approach that depends on scan-based debug technique gives full observability of the intermediate system's various state, but it is not able of handling real-time conditions in continuous clock cycles. It is also unpractical to put this method every clock cycle over a long operation time. Thus, so embedded logic analysis methods, which are based on real-time trace system, have been explained in order to aid scan based technique and enhance real time observability.

Overall, we had gone through all the past techniques used to enhance the observability of the intermediate circuit's node points and improves the post -silicon debugging process.

CHAPTER-3

SoC ON CHIP DEBUGGING
DEBUG SUB SYSTEM (DSS)

3.1 Overview

In the last chapter we had seen about the different debugging system like scan based debugging and embedded logic based debugging, here we will see about system on chip(soc) debugging, before we go for this soc debugging we must understand about the term SoC.

3.2 System on chip

A SoC brings together the all desired electronic circuits of different electronics parts onto a single, IC. SoC is a completely electronic substrate system that including digital, analog, mixed analog and digital signal or various radio frequency functions. Its components generally takes a CPU that it can be a multi core, a GPU, and system memory.

Because SOC is combination of both the software as well as hardware, it takes less power, has the better performance, needs less area and is better reliable than multi chip systems. Most of the system on chips now a days come inside the mobile devices like tablets and smart phones.

A SoC is designed specially to meet the standards of different electronic circuits of numerous electronic components onto a single compact integrated chip. In place of a system that combines various chips and their components onto a circuit board, the SoC designs and fabricates all necessary circuits into one unit part.

The challenges of a SoC is that it is having higher prototyping designing and architecture designing costs, lower IC yields and more complex debugging.

IC is not economical and it takes time to get manufacture. Yet, this is to change as such technology that continues to be employed and developed.

An SoC basically contains different components like:

- An operating system
- Different utility software applications
- Power management circuits and voltage regulators.
- Different timing sources such as PLL control systems or oscillators
- A microprocessor, microcontroller or DSP.
- Various peripherals like counter timers, real time clocks and power for reset generators
- Various external interfaces like FireWire, USB, universal asynchronous receiver transmitter, Ethernet or different serial peripheral interface bus
- Analog interfaces such as DAC and ADC.
- ROM and RAM memory

3.3 On chip Debugging

Previously there was the debug analysis monitor. Economic but influential, it still serves handily alongside the most costly debugging tools. However analysis monitors have their own limitations and weaknesses. For instance, they need RAM, ROM, and a communications trace path from the target; they require to be mapped to the target hardware; and they don't permit to set breakpoints in the programs that running out of Random access memory.

After the debug analysis monitor that came the in-circuit emulator. By using some brilliant hardware methods (usually based on bond out versions of processors), an ICE provides capabilities far beyond those of a random access memory monitor.

The ICE's most different features that include various complex breakpoints, the real-time traces of different processor activity, and there is no use of target resources. But if we analyse this extra added function then we have to pay high cost.

Basically emulators can be costly. The integration levels, increasing speed and complexity, of recent processors bounds the presence of bond out versions, making emulators tougher and costly to configure and design. As a conclusion, some debugging technique features are unique to ICEs are not available for recent processors.

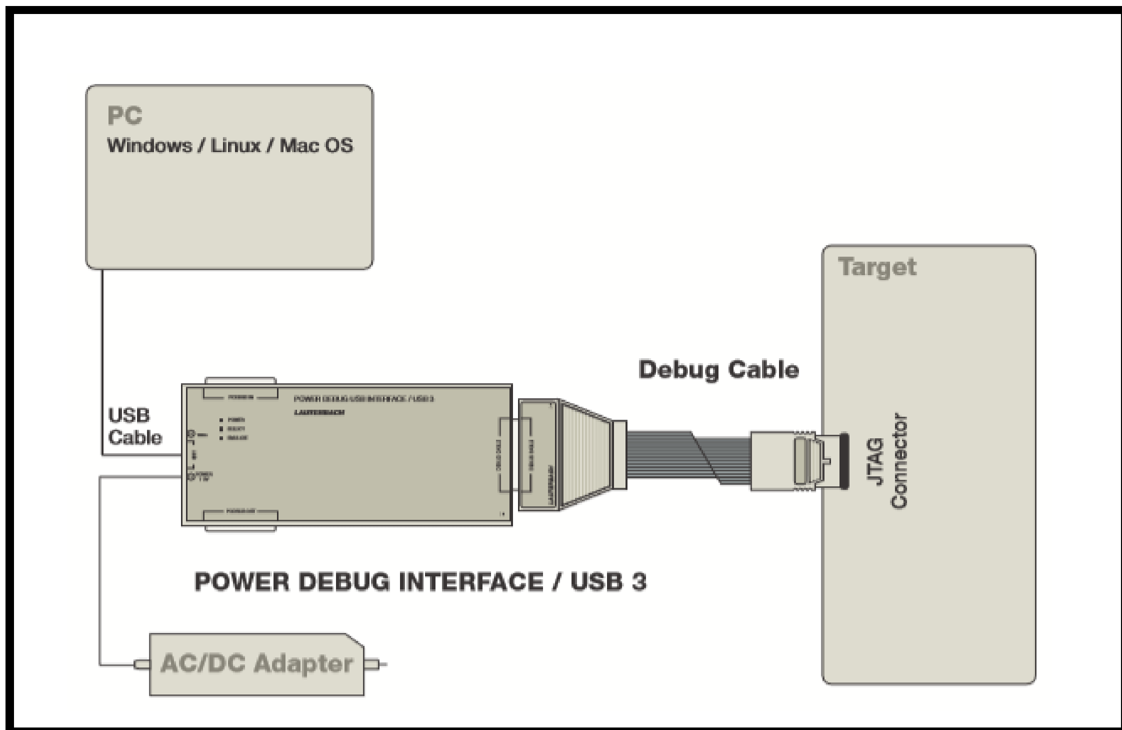


Fig. 3.1 On chip debugging environment[13]

To overcome these difficult and hard trend, many semiconductor chip vendors now design dedicated and fixed debug circuitry configuration into their chips. Each chip vendors basically have their own proprietary configuration name for these technologies, such as MPSD, BDM and OnCE. Whereas other chip vendors simply go through software debug configuration capabilities to their already existing JTAG ports. Collectively, we will call this technologies as on chip debugging.

Such hardware based debugging capability replaces a software debug analysis monitor, yet this offer some additional enhanced features previously these associated with emulators only. And they do both of them at a common cost.

3.3.1 Components of on chip debugging

Along with debug target to debug and PC to operate debugging there are some hardware and software components called debugging environment required for on chip debugging.

- JTAG.
- Debugging board.
- TRACE 32(simulator tool)
- Power supply

Along with the above knowledge of below language required for debugging.

- CMM scripting
- TCL scripting
- PERL scripting
- ADB commands
- C and Python language

Now we will little bit about different components of debugging environment

3.3.1.1 JTAG

JTAG is commonly termed boundary scan and it is termed by the Institute of Electrical and Electronic Engineers (IEEE) 1149.1, which basically began as a integrated method for testing and validating various interconnections on printed circuit boards implemented at the integrated circuit level. As Printed Circuit Board’s grew in density and complexity—it is a trend that continues today—drawbacks in the previous most using test methods of in the circuit testers.

Basically Ball Grid Array and Packaging of formats, some of the best fine pitch parts, synthesized to meet always increasing physical space values, it also led to the loss of physically accessing the desired signals.

Now a days, JTAG is used for everything from testing interconnects and functionality and working on ICs to programming the flash memory of the systems positioned in the area and all in between an area.

JTAG and their related standards have been and will run to be increased to address extra problems in the different electronic tests and making, having test of the 3D ICs and big complex, hierarchy systems.

Here we had gone through the how evolution of JTAG took place, now we will see how JTAG gets linked to our debug board and how it work for connecting debug board to PC. With a view to use boundary scan, JTAG architecture system, it is important to be capable to do data transfer correctly with any of the that is set up to use JTAG. A JTAG architecture interface has several of lines that are took in use and collectively these are called as the TAP. This JTAG port is used to control JTAG and also giving connections by which the serial data transfer take place bidirectionaly.

3.3.1.2 Debugging Board

There is no any direct point of contact on any IC that will be connected to JTAG, to connect IC core to the JTAG one medium is required and that is the debugging board. Debug board is connected in between the JTAG port and IC. It has two ports one for the get connected to system through JTAG and another for the get connected to the debug devices.

3.3.1.3 TRACE 32(simulation tool)

The lauterbach item TRACE32 ICD gives a huge extent of on chip debugging interface. The hardware for this debugger is widespread and permits interfacing various target processors by basically changing the software.

Support and debug cable and for a wide extend of on chip debug interfaces assembler debugging Interface and Simple high-level to all compilers quick download RTOS awareness Interface to all hosts Display of inner and outside peripherals at a consistent level breakpoints of Flash programming Hardware and trigger (in the event that backed by chip).Multiprocessor/multicore debugging Software follow Virtual analyzer USB 3 Interface.

3.3.1.3.1 How to useTRACE32 (simulation tool)

Procedure for debugging will be like that we have to connect the device to debug to the debug board through DAP (debug access port) then that debug board will be connected to our system through JTAG. we access the debug target through one simulator tool that is TRACE 32.we can see any at present register value and we can see their changes, even we can change that register value with the help of this simulation tool. Here we will see how we can operate the debug target with the help of this simulation tool.

- First of all, install the software TRACE 32, there are different sources to download this software, this software designed in python language, so in industry for different projects their TRACE 32 is differently designed according to their project specification.
- Secondly, we will connect our debug target to the system through JTAG
- Then we will flash the Meta file of that project on the devices, basically we get two types of hardware for debugging first one is CDP and the second one is MDP, this MDP is the compact form of CDP.

These all the basic steps that we will do before starting any project, after this we plan according to our debugging target and for any debugging activity script is written in various languages like .cmm, tcl, perl and python.

Besides, it must be expressed that the Trace32 program does not collect, compile, or connect your program.

This must be done by a few other application specific for this reason.What trace32 does arrange to the user is download the executable to the target run the program debug it and the rest of the features recorded underneath the software.

The software provided with the Trace32 system, gives support for all the capacities that the Trace32 system[14] orders:

• Debugger

• Emulator

– ICE

– FIRE

•Trace

– Bus trace, Flow trace, NEXUS trace

– CTS (Context Tracking System)

– Cache analysis (PowerTrace)

– Function statistics – Performance analysis – Code coverage (PowerTrace)

• Timing analyzer

• Port analyzer

• RTOS Debugger

• Flash programming

During our debugging experiments, we concentrate on the program flow trace w.r.t timestamps, which uses the tracing features given by the PowerTrace module.

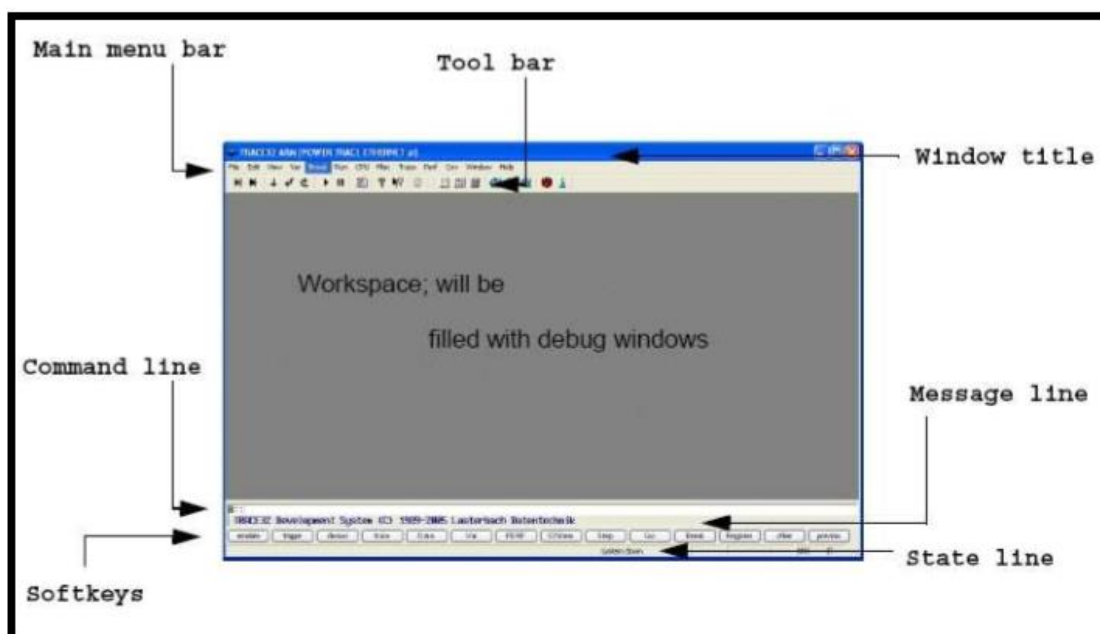


Fig. 3.2 Main window of Trace32 application

In the case of Windows operating system, the user interface can work in two modes:

1. MDI (it stands for "Multiple Document Interface"), this keeps all windows inside the surrounding window.

2. MWI (it stands for "Multiple Window Interface"), this take use of command line window and it spreads other windows inside the whole screen.

Every command or operation can be done in three different ways[14]:

1. By using the pulldown menus and menu bar.

2. By using the tools bar and the tools bars with some special functions that appear in some specific windows.

3. By using the command line (command prompt) and some specific PRACTICE script language.

Among the all way of operation we focus more on the operation using command line.

In the GUI for simulation, commands can be loaded in the command line at the bottom command bar of the main window of the simulation tool. Commands can be shortened using the significant characters for each command entry. Upper case letters are used to indicate significant characters in this document as well as in the Trace32 documentation. However, it is no need of using upper case letters when executing the commands, since the language is not cases sensitive. The soft keys provide a command entry by showing all possible parameters and commands, so that all command can be combined by clicking the soft keys instead of the wasting time by writing the complete command line.After the proper connections are made and there is no power supply provided to the debugger or target, the following steps should take place in the order they are mentioned:

1. Turn on debugger.

2. Start Trace32 ICD application.

3. Apply power to the target.

4. Run batchfile.

A batchfile, mentioned in step 4, is a Trace32 startup script, a text file with a “.cmm”.Here to make batchfile we have to focus on .cmm scripting, we will discuss about this later, now we just see how we do tracing with TRACE 32

3.3.1.3.2Tracing Features

Trace32 tools order various different trace methods, like Analyzer, ART, Logger, SNOOPer etc. as shown in Figure 3.3. For this set of experiments, we take help of the the trace method Analyzer, which means that there is availability of physical trace memory, provided by the development tool, in our case, it is by the Power Trace module. Apart from the trace methods, there are also different trace techniques supported, like flow trace, bus trace and NEXUS trace. Flow trace, and program flow trace are what we focused on. Program flow is having the tracking of Power Trace, that allows to track the program instruction flow with almost no degradation in performance. To upgrade the trace analysis setup and control the storing to the trace buffer, use command “Trace.state” or “Trace”. The analysis to the trace buffer can also be managed by trigger and filter features provided either by the Trace32, or by the on chip trace and the trigger support of the CPU[15]. The command options will open a window like in Figure 3.3. There, you can see various options to control, some of them can vary depending on the target. For example you can check or change:

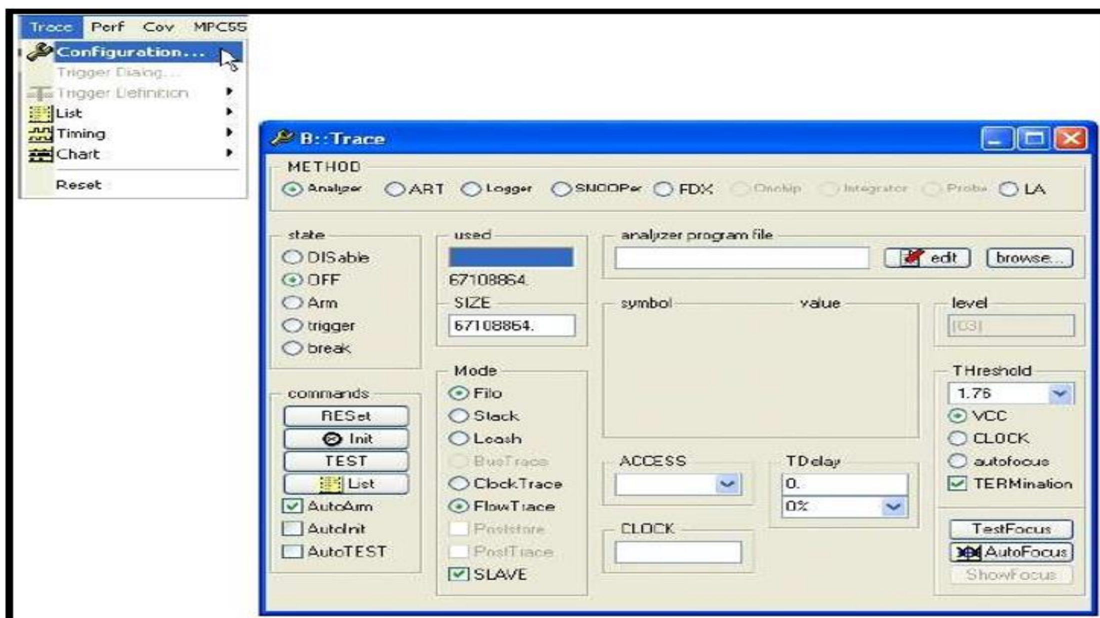


Fig.3.3: Trace setup window

This all about the simulator tool TRACE 32 now we will see about scripting.

3.3.1.4 Various scripting language

We had seen how to use the simulation tool, now we will see about different scripting language used for the debugging.

- **Cmm script:** In the scripting each script line contains only one command . We basically write scripts in the notepad or save with the cmm as extension.There are so many direct commands in the cmm commands like: data.dump, data.set, data.store etc.

  In this scripting language we use to write different use cases like for display, modem, camera, camcorder etc. we use different cmm script for different programming and run these all cases on the TRACE 32.

- **Perl script :** first of all we initialize this TRACE32 with command "sys.m.a" this makes the TRACE32 active for the device attached to it. Then we keep the device in fastboot mode, after flashing the device we use ADB commands to operate the device. Then we reboot the device and be sure that your TRACE32 status bar should be green in colour and it will show in running status. If device status will show "no power" mode then we run the low power mode batchfile to change the mode in running mode.
  After this there is need to start with defining source and sink to the tracing profile so that we can measure data at different instances in between the source and sink we defined. Let's take an example if we had taken MNOC (multimedia NOC) as source and ETB (embedded trace buffer) as sink then we first of all in TRACE32 we run the script of source and sink defining script. After that we run the specific use case script and analyse the data at different registers.

Similarly we use tcl script, c, python and ADB commands during our debugging process for different process.

CHAPTER-4

AUTOMATED ON CHIP DEBUGGING TOOL

4.1 Overview

If we see on-chip pre and post silicon debugging then we will see that for tracing data in debug sub system there is a relation among

1) Source

2) Sink

3) Timestamp

4) Cross trigger Interface

When we go for the trace analysis we can see that the scenario will be like that we will dump any value in register of any source and trigger that register and analyse that value at the sink register, we will do all analysis w.r.t to timestamp interval. In between the source and sink there are 3-4 checkpoints that is called the ATID (advanced trace bus ID).When we run the script then we get the value in the form of bin file then we have to parse that bin file and we get excel sheet as outcome, in this excel sheet leftmost row is the time stamp row that show different timestamp value at different timestamp interval. At different nodes we get the value at different timestamps and by this we can analyse that how much time is required to travel data from one node to another node and by this we can measure the "DATA LATENCY" and this also gives frequency between two nodes, parser value gives different values at different timestamp on nodes so if we plot these values w.r.t timestamp then we can also get the "BANDWIDTH" at that node, so this is the way to get latency and bandwidth of SoC and this is also a way to debug, let's suppose we get any latency and bandwidth of any node, we match these to design constraints.

If it will in safe zone then its ok but if it will be in danger zone the its matter to go for backtracking and detect the place for bug.

Till now all these was in the form of multiple steps:

1) Attaching devices through JTAG, and flashing META file

2) Start of TRACE32

3) Setup source (any master) or sink (DDR buffer) through Perl script.

4) Setup all clock enable through running cmm script

5) Run the desired use case script

6) Pull DDR value, it gives ddr.bin file

7) Extract ATID, we get all bin files

8) Parse all bin files with designed parser

9) We get data in form of excel, we can analyse bandwidth and latency from it.

Prerequisites of these steps are keep the different scripts ready for the analysis. so we have seen that to debug any of the use case all the following 9 steps have to follow.

4.2 Proposed work

I had divide all the above 9 steps into 2 steps with the help of python automation tool, here two automation tool is designed

1) Parser automation tool

2) Data analysis tool

4.2.1 Parser Automation Tool

4.2.1.1 Overview

As we have seen that if we have to go for validation, the debug is the way, and for that we have to go through above 9 steps one by one but it is the very time consuming process and to go through the best validation we have to go through these 9 steps 100 times for one use case analysis. So, here i converted the above all 9 steps in a graphical user interface(GUI), so that you can do all these steps on the single window.

Here we can see the parser window in figure no. 4.1

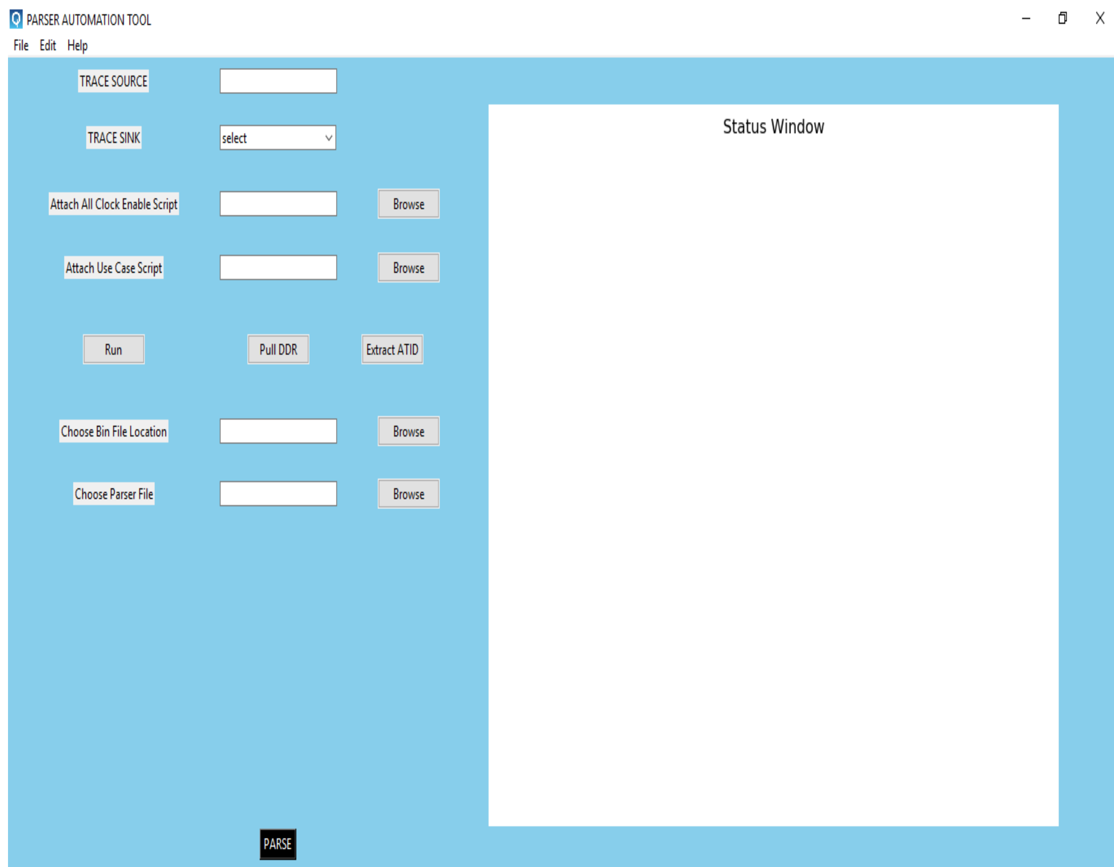


Fig. 4.1 Parser Automation Too

4.2.1.2 Theory

Here we had seen that this whole validation/debugging is the matter of scripts, we have to write different scripts for different situations and use cases. So here i had divided my whole debugging activity in different steps.

Step: 1

I have to make the list of all scripts that is required from start and end of the debugging process. Then after i have to write the script according to our situation, here like the first script will be for source and sink definition, so as we enter the source and sink of trace profiling activity on the tool GUI then our python script will take source and sink name from the GUI and it will automatically create the script and store this script in the data base.

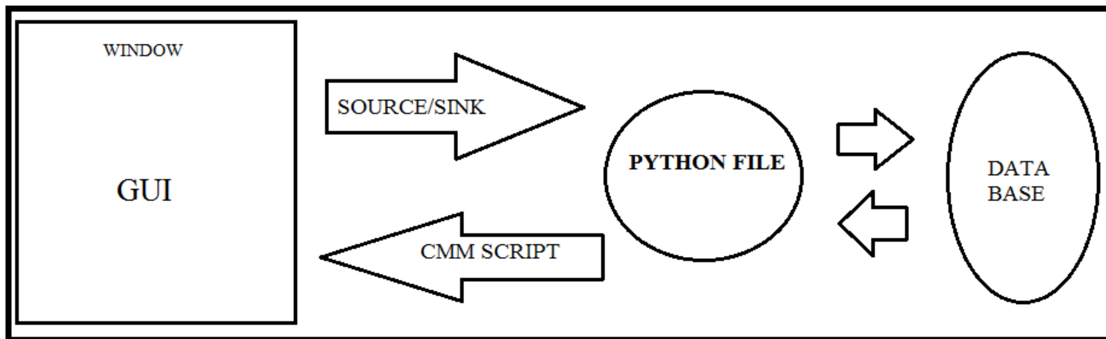


Fig. 4.2 How python connects database to system

Here for example i am taking MNOC (multimedia NOC) as source and ETB(embedded trace buffer) as sink, for this we have entered this value to the GUI after this entered name will be send to the python script then that python script will make adp script in which source and sink will be defined and that perl script will be stored in the data base, after that when we click on run button then python script will fetch that adb script from data base and it will forwarded to window to run.

source adb script:

echo 1 > /sys/bus/coresight/devices/coresight-mnoc-etm0/enable ....(i)

sink adb script:

echo 1 > /sys/bus/coresight/devices/coresight-tmc-etb/curr_sink ....(ii)

each master on soc are attached with a register that is called as the ETM(embedded trace maceocell) so here in (i) we had seen that here we had chosen MNOC as our source master then we had written "mnoc_etm0/enable" it makes MNOC as source enable.

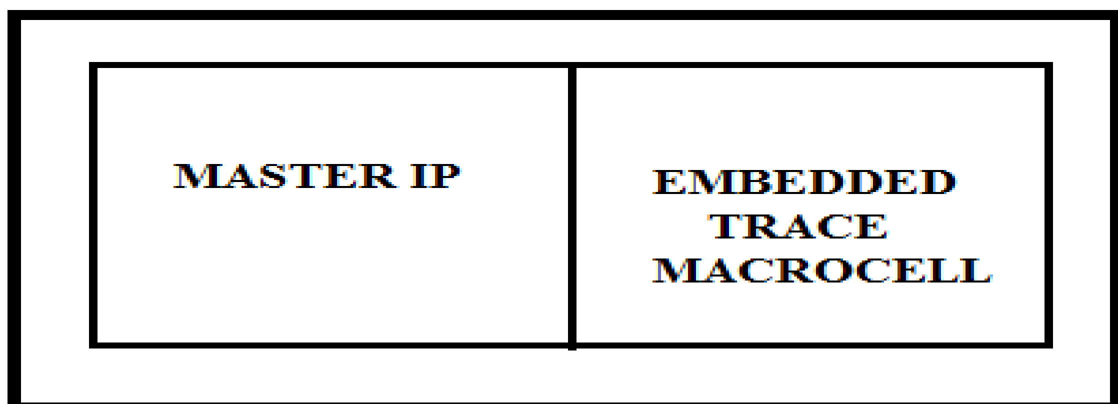


Fig. 4.3 Master attached with ETM

After getting the source and sink name from the GUI that will send to the python file that have script that enable the source and sink , here enabling of source and sink stands for the keep the the ETM of the source ready for accepting the trace data whereas at the sink end there is an alternate buffer that is attached with the DDR/ETB that holds the data that will move from source to sink during the tracing, here in the script along with enabling the source and sink we also define the sink buffer size, for example we can see that.

echo 1 > /sys/bus/coresight/devices/coresight-tmc-etb_buff-size 0x200000 ....(iii)

Here we had assigned the sink buffer size as 0x200000,that creates one buffer attached to sink for dumping the data that send by the source.

Step: 2

Here after selecting the source and sink and defining memory to the sink, as we move to the next tab then here we have to upload the all clock enable script, here we have to browse that script and upload it ,by running this all clocks get enable here so we can analyse data at different points in between the source and sink w.r.t the clock.

we write the all clock enable script in .cmm script , here if we talk about the pre silicon debugging then there is no need to worry about clocks too much because all clocks value are written in system verilog or UVM so here easily clock details can be mentioned but when we go for the pos silicon debugging the we got all in form of silicon so its difficult to give clock values, so here we have to provide clock virtually, like we have to right PLL script in cmm that will generate clock according to our required frequency. So that is the difference between pre silicon and post silicon clock for debugging.

```
39 ;==========================================================================
40 ;arrange debugger windows
41 &core0 framepos      -183.   0. 181.  26.
42 &core1 framepos      -183.  39.3 181.  26.
43
44 ;==========================================================================
45 ;print message to distinguish between cores
46 &core0 print "*** hello core 0 ***"
47 &core1 print "*** hello core 1 ***"
48
49 ;==========================================================================
50 ;Set path of core 1 debugger to the path of core 0 debugger
51 &path=OS.PWD()
52 &core1 cd &path
53
54 ;==========================================================================
55 ;System settings
56 &core0 SYStem.BdmClock 4.MHz
57 &core1 SYStem.BdmClock 4.MHz
58
59 ;detect used processor
60 SYStem.CPU SPC56EL70
61 SYStem.DETECT CPU
62 &cpu=CPU()
63
64 ;==========================================================================
```

Fig. 4.4 core clock enable script

Now when we browse and upload the all clock enable script then it will run and it will enable all clocks. Here we can see in the script of core clock script in figure 4.4, but remember that for the post silicon debugging we have to first do the PLL scripting, here we have to set different registers that gives the clock frequency according to the register we set for PLL.

Now after running the clock script we have to go for different use case data tracing.

Step: 3

Now till now we had enabled all the script , now we have to go for the data tracing, here we trace the data along the complete path from source to sink. But here are there are two modes of use cases:

(i) Stand Alone Mode

There are many use cases during the debugging like

- Camera & Camcorder Use Case
- Modem Use Case
- Display Use Case etc.

Here like this there are so many use cases are there, basically debugging process is all about the validate the different trace paths in different use cases. During debugging we

- analyse the following thing
- Latency between two different points.
- Bandwidth at different NOCs. etc.

So we measure all above values and match up with the specified constraints and later on we compare the pre silicon data with the post silicon data and if there is no deviation in result then it means there is no any bug during the fabrication. So, for the tracing we have to run any use case script that will move data from source and sink . To capture data at different points we define ATID(advanced trace bus ID) at different NOCs, and points in between the source and the sink and for this we define different ATIDs in the use case cmm script. By defining these ATIDs in script data gets dumped at these points and we can easily analyse data at these points in the form of excel sheet so it will be easy for us to analyse data at these points.

As above we had seen that there are two modes to run the use cases one of them is stand alone mode, in this mode only one use case is analyse at a time like i have to measure data in only one use case like i have to capture data for camera preview mode or camcorder 1060 HD mode or any individual mode then we run only one use case and select that use case script from browse button of the GUI and click run button, what all happen after pressing run button we will see in step 4.

(ii) Concurrency Mode

Above we had seen the stand alone mode now, here we will discuss about the concurrency mode, there will be some situations like that we have to analyse the system when two modules will work simultaneously like we have to analyse when we are using both wifi and camera simultaneously then what will be effect on the bandwidth , latency and other parameters.

So to validate or debug such type of situations we have to run both use cases scripts simultaneously then we have to capture and analyse data at different points, to analyse these two or more use cases simultaneously, this mode is called concurrency mode.

For the concurrency mode debugging/validation in the GUI we have to attach all the use cases script simultaneously and we have to press run button.

Step: 4

After uploading the use case script either of stand alone mode or concurrency mode ,now we have to go through the script that we had uploaded, so that we can analyse what is the way and sequence of writing the script. For any trace debugging method there is no any direct path between source and sink, in between that there are so many things like:

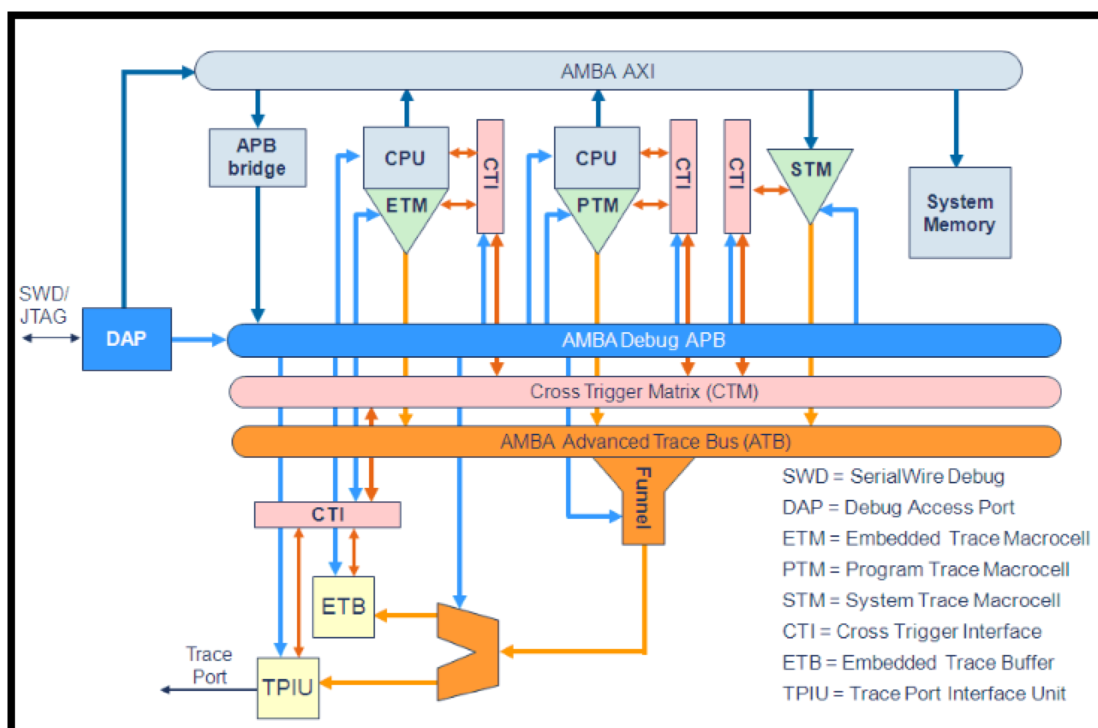


Fig. 4.5 Debug Trace Subsystem

- NOC(network on chip)
- Funnels
- Replicators
- Upsizer and Downsizer
- ATB(advanced trace bus)

So, these are the major things that will be appear in between the source and sink, there are different register that assigned with the above elements ,and different values are assigned to these registers that are responsible to give dedicate path between source and sink. So all the elements in between the source and sink are included in cmm script that makes an open path from source to sink.

Role of Timestamp:

Here timestamp plays an important role as reference for measuring latency in between source and sink and bandwidth at different NOCs. Here timestamp plays a role of reference timeline for measuring time taken to move data from source to sink, by this

we ca calulate how much time data takes to move at different sinks from different sources and we can get bandwidth at different nodes at different time stamp.

Role of ECT(embedded cross trigger):

It is having the ability to send triggers to HW blocks & processors around SoCs. ECT is having two modules:

(i) CTI(cross trigger interface)

It controls the trigger interfaces. It combines and map the different triggering requests and send it to other interfaces.

(ii) CTM(cross trigger module)

It controls the distribution of channel event

Here is the role of Timestamp and ECT in trace sub system, we had gone through the role of both two in the debugging. During writing the use case script we have to initialize all these registers according to their need, what all values you have to keep during debugging, and what all registers we have to use during debugging is provided by the DV(design verification) team.

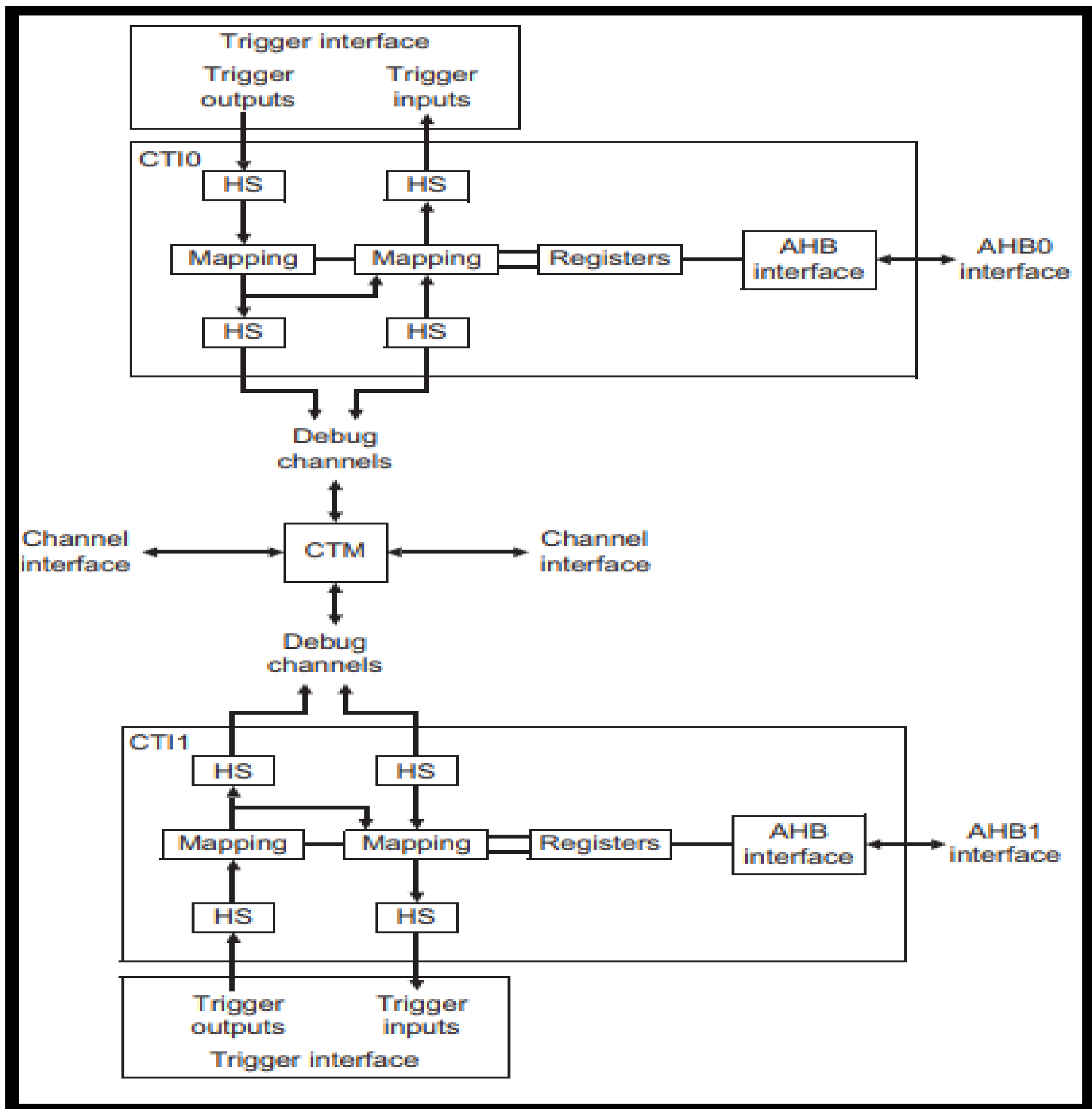


Fig. 4.6 ECT system

Step: 5

Here we are ready with script and keep the script in the data base and we will browse the specific use case according to the mode either stand alone or concurrency mode. Here we will see what will happen in this trace path ,when script will run.

As we load the use case script file address and press the run button it will directly send the script location address to the python script of TRACE32 simulator tool and it will start running after pressing run button on the GUI, as it start the tracing profiling timestamp will start its reading as data moves from source to sink.

In the trace sub system path then data will be recorded in the form of bin file and get stored at the location where use case script was kept. we will get . Suppose in between the source and sink we are having 3 points along with DDR where we have to analyse the data then we give some IDs to these locations like ATID 9, ATID 11, ATID 13 and we will define these in the use case script so that at different time interval we can capture the bin file data at these points.

Now we have to click on the "pull ddr" button by this we can get the data that captured at the DDR, here we get data in the form of bin file the file we get will be like"ddr.bin" here sink data gets captured in the format of binary codes.Now we had seen binary data data captured at DDR now we see what will happen at the different ATIDs.

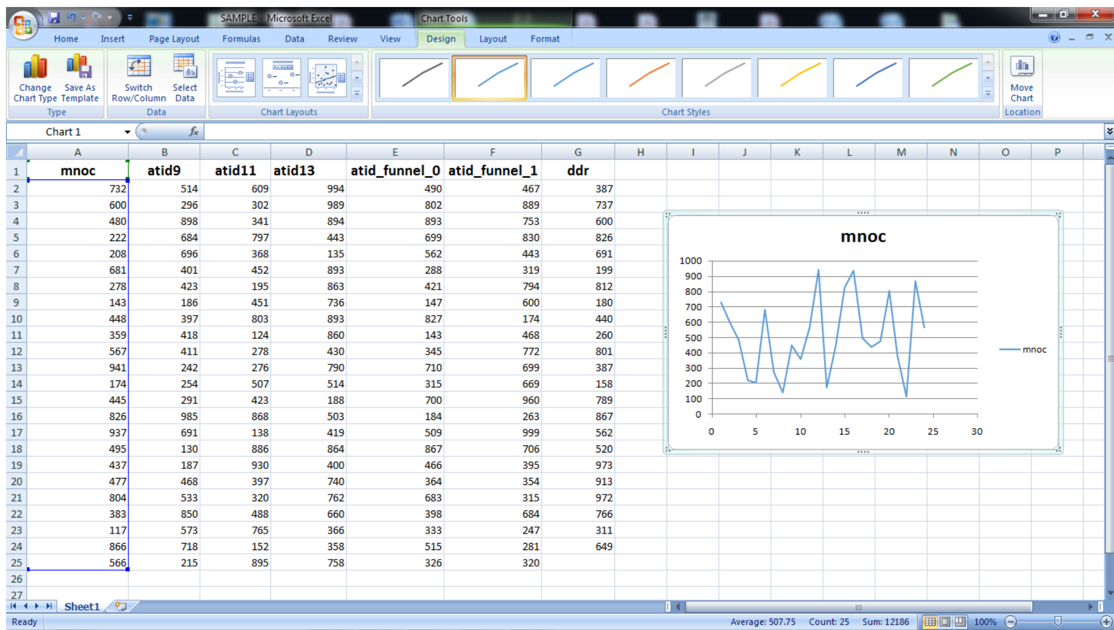
After clicking "pull ddr" button we got ddr captured data now we click on "generate ATOD" button by this we get bin file data at the different ATIDs like ATID 9 , ATID 11, ATID 13 so on . Here we got

- ddr.bin
- atid_9.bin
- atid_11.bin
- atid_13.bin

Now we have to go for parsing these all bin file to get the data in excel file. After this we go for choose file location column in GUI by selecting the file all, generated file will get move to that location. Now we go for parsing

Step: 6

Here we go for the parsing, basically parser is a file that is use extract data as single value or table from the instrument output. Here first we have to go for the parser scripting, where we have to write the parser batch file , here it is in c language.



	mnoc	atid9	atid11	atid13	atid_funnel_0	atid_funnel_1	ddr
2	732	514	609	994	490	467	387
3	600	296	302	989	802	889	737
4	480	898	341	894	893	753	600
5	222	684	797	443	699	830	826
6	208	696	368	135	562	443	691
7	681	401	452	893	288	319	199
8	278	423	195	863	421	794	812
9	143	186	451	736	147	600	180
10	448	397	803	893	827	174	440
11	359	418	124	860	143	468	260
12	567	411	278	430	345	772	801
13	941	242	276	790	710	699	387
14	174	254	507	514	315	669	158
15	445	291	423	188	700	960	789
16	826	985	868	503	184	263	867
17	937	691	138	419	509	999	562
18	495	130	886	864	867	706	520
19	437	187	930	400	466	395	973
20	477	468	397	740	364	354	913
21	804	533	320	762	683	315	972
22	383	850	488	660	398	684	766
23	117	573	765	366	333	247	311
24	866	718	152	358	515	281	649
25	566	215	895	758	326	320	

Fig. 4.7 Parsed data

We had mentioned here the table column heading sequence where we get the data in the same sequence after parsing. For example like our sequence is {"mnoc", "atid9", "atid11", "atid13", "atid_funnel_0", "atid_funnel_1", "ddr"}. If we keep the parser in such format then parsing all the bin files we get data in the same format.

Here we had got the parsed data in the tabular format(csv/excel), now we have to analyse this csv/excel file and for that we have to go for my second automated tool i.e "Data Analysis Tool".

4.2.2 Data Analysis Tool

4.2.2.1 Overview

Till now we had got the data in the format of csv/excel now we have to analyse the data because data is in form of tabular format then we have to go for the graphical representation of this tabular data so that it is easy for to analyse that. Here we go for four major thing analysis:

- Latency
- Bandwidth
- FPS(frames per second)
- Timeline

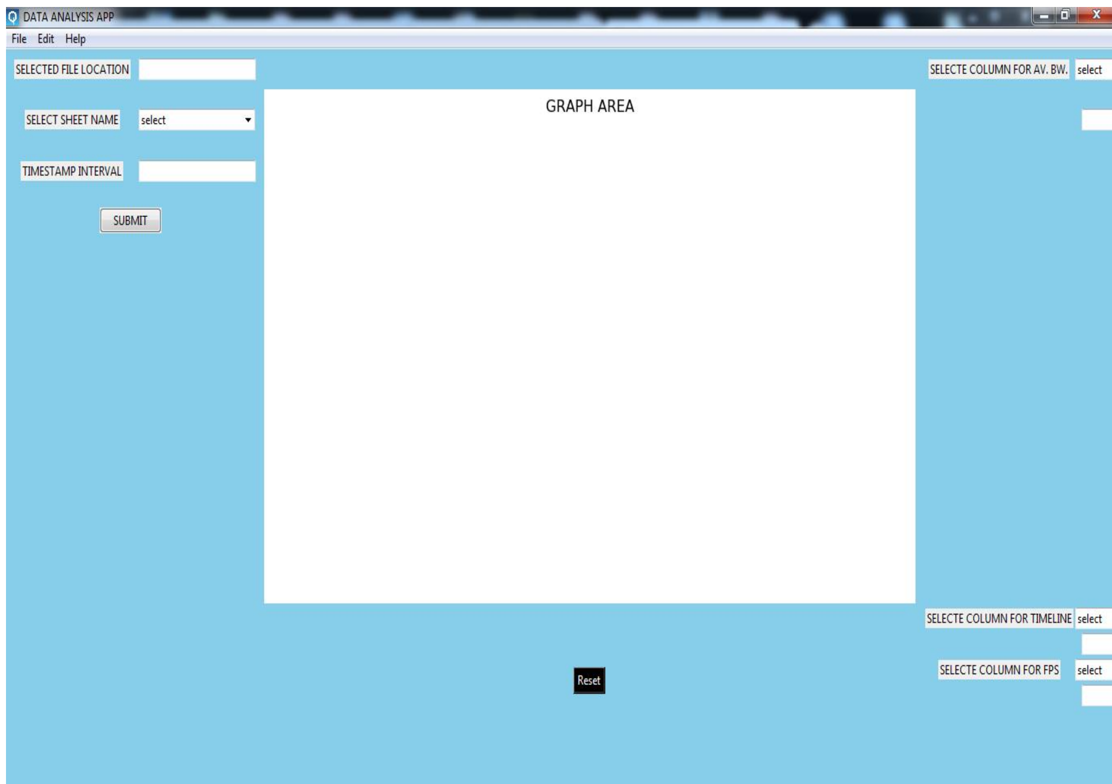


Fig. 4.8 Data Analysis App

So, these all are the major thing, for analysis of them graphical representation is required.

This is the GUI window for the "Data Analysis Tool", we can see the different labels and column box for the different functionality like graph area, timestamp interval, average bandwidth, latency, FPS, Timeline etc.

4.2.2.2 Theory

Here we had got the parsed data now we have to go for the analysing the parsed data.

Step: 1

first we have to select the parsed data file name to whom we have to analyse ,for that we have to go to file option in menu bar then we have to go for open option, and from there we can select our desired file to whom to analyse.

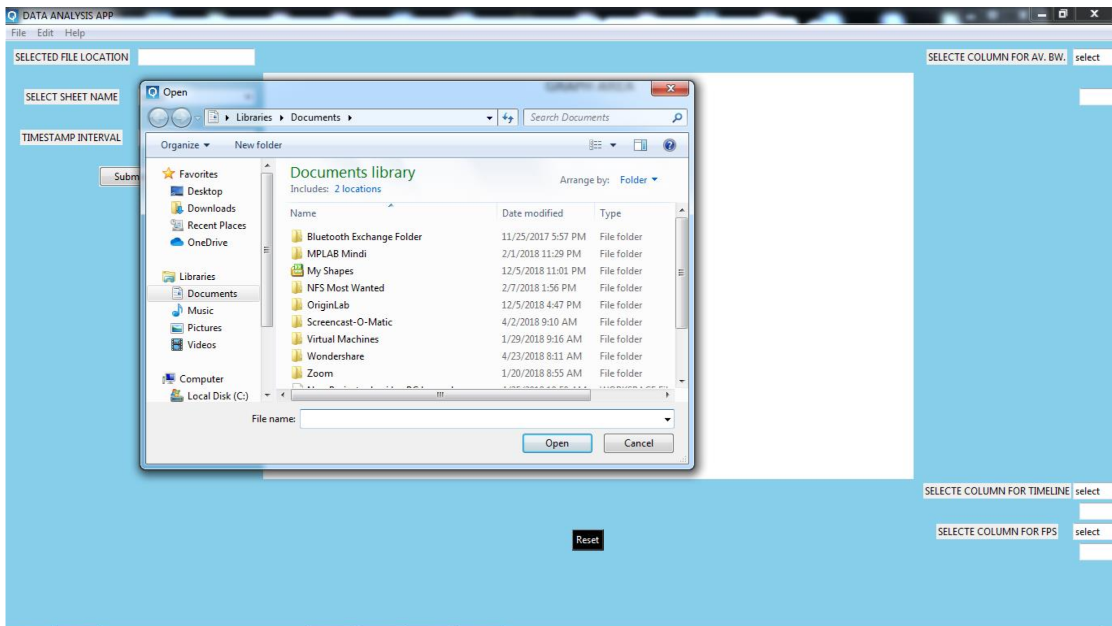


Fig. 4.9 Selection of file to be analysed

As we select the desired file then that file address goes to the python script that read the parsed excel file we selected and it saves that file into one variable. Now we know that in one excel file there are more than one sheets available .

Here we have to select the sheet name also because we can analyse one sheet at a time, so as we select the excel sheet then it automatically gives the list of all sheets available in this excel sheet, just we have to select that sheet, now we have to fill the timestamp value, basically this timestamp interval value is that time interval between the two consecutive row. Here for analysis we mostly take 1 micro second.

Now after filling this as we select the submit button then all present columns in that excel sheet will appear on the GUI and as we select one or more than column name then it will automatically get plotted in the graph area, Here we had different colours to the different column names to identify any particular graph on the multiple graph plot on the graph window. Let see that:

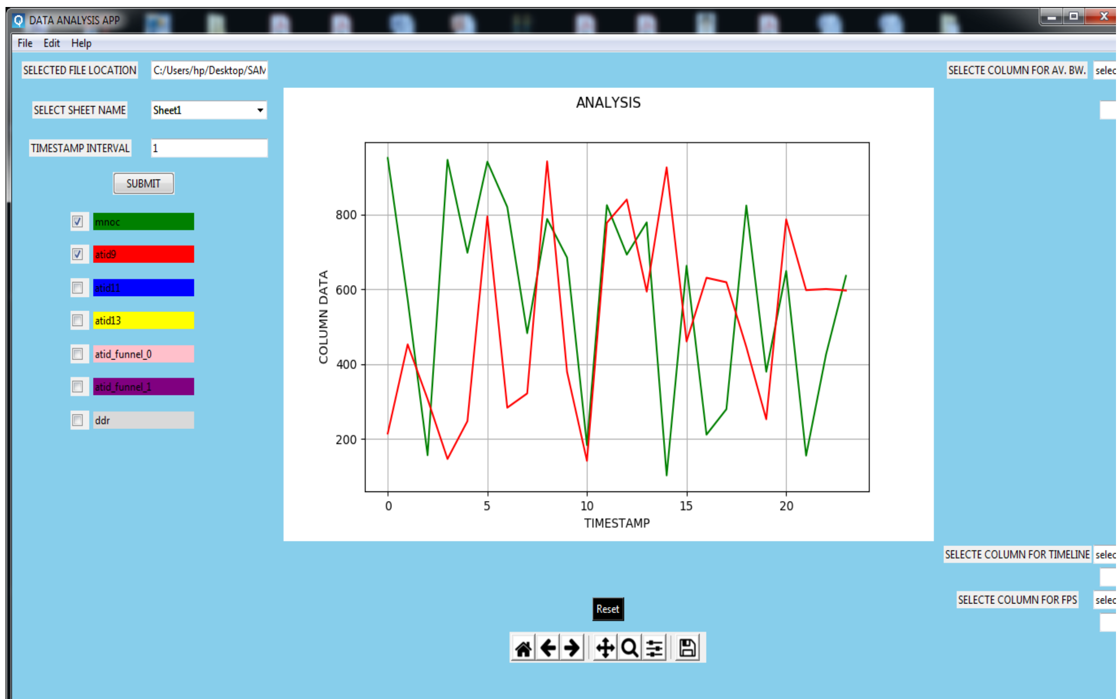


Fig. 4.9 Analysis of two column data simultaneously

Here we had seen how we can analyse two columns data at a same time, here navigation window is also designed that is responsible for the drag, zoom in, zoom out and we can save our plot as jpeg image.

Step: 2

Here by zooming and drag option we can calculate even latency, bandwidth, FPS also because as we move cursor at any point of graph window then navigation window shows the x and y value w.r.t cursor.

4.2.2.3 Calculations

Here we had discussed about the plotting of data, now we see about how to go for the different parameters calculations:

(i) Latency

Here latency is the minimal time required to move data from any point "a" to another point "b". this can be analysed with the help of plot let see an example:

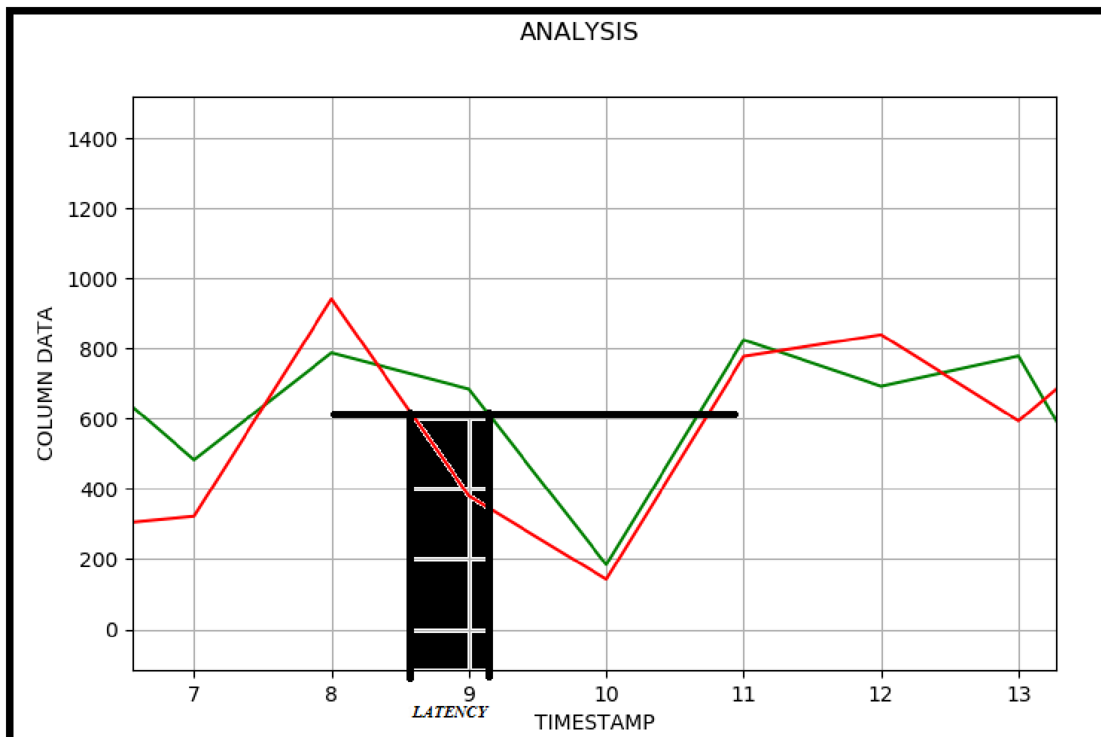


Fig. 4.10 Latency Measurement

So, here is the measurement of latency with the help of plot, here x axis difference gives the minimal time difference that is called latency.

(ii) Bandwidth

So here we had seen how to calculate the latency, now if we go for calculation of average bandwidth then we have to just sum the all values that collected on any point and we have to divide it with the total time taken, here we can see on GUI there is option to calculate the average bandwidth, here in the drop box all columns in the sheet we selected will appear\, now we have to do only to select the specific column whom bandwidth you wish to calculate, as we select the column there will be python scripting in the backend that is responsible for the average bandwidth calculation.So by this way we can calculate the average bandwidth of any column.

(iii) FPS(frames per second)

If we for the multimedia validation or debugging then it is a major role of this parameter, this count gives the no of frames per second captured at any node, earlier it was the way that we see the plot and then count the no of frames that is a very difficult process to analyse all these because sometimes we get 60fps and more so its difficult to analyse 60 frames in 1 second window, that's why here we had written python script that can calculate no of frames in 1 sec from the parsed excel sheet data only, here there is no need to go for the pictorial analysis of the data.

This is done with the help of python script, here on the GUI just we have to select that column whose FPS we wish to calculate, here after uploading the column name for the FPS calculation, automatically python script will take that column name from the GUI and it will start calculating.

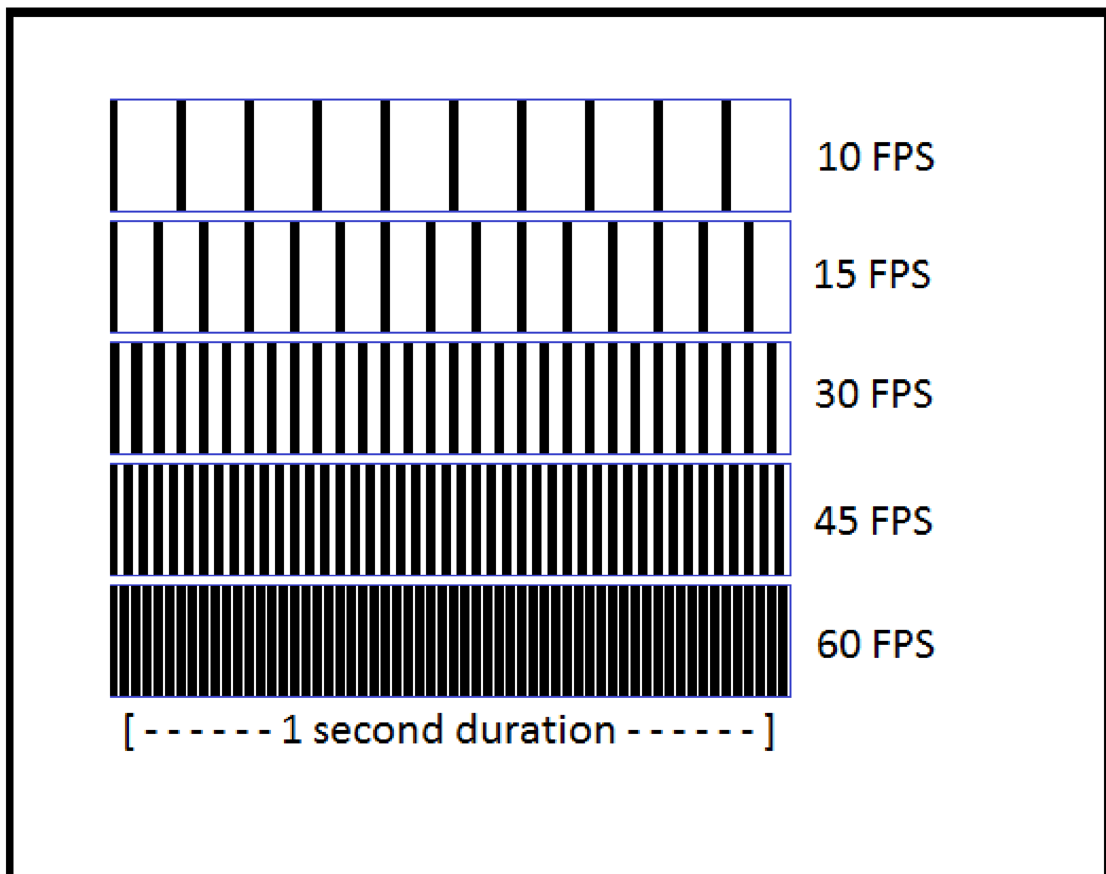


Fig. 4.11 FPS(frames per second)

Here we can see the different FPS values.

(iv) Timeline

Here timeline is that time duration for which our value is high.

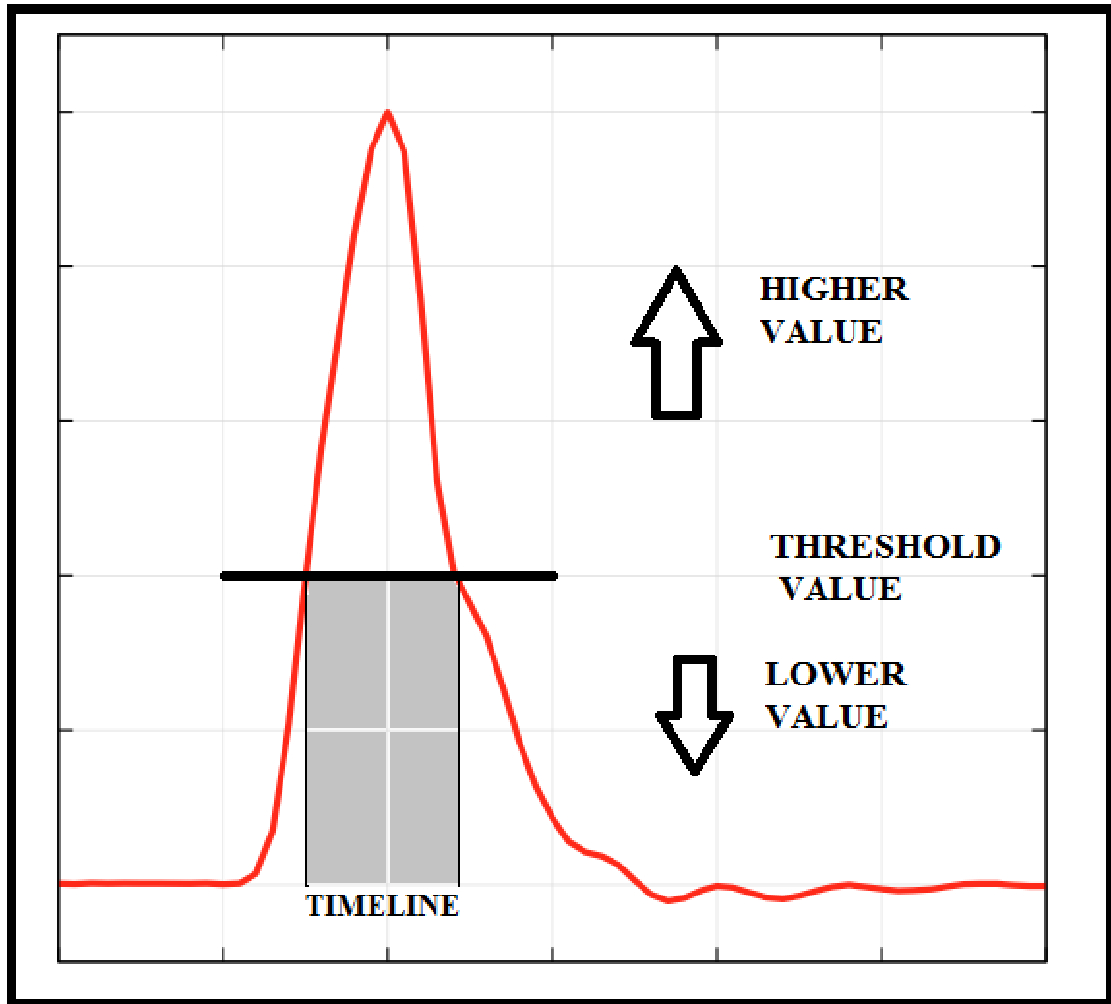


Fig. 4.12 Timeline

Earlier it is calculated with the help of plot like we use to see the plot and note down two values one where it goes from zero value to non zero value and other is when it goes from non zero value to the zero value and then we use to subtract the value(higher to lower) it gives the value of timeline, but here it is also not a perfect way to calculate the timeline, here also we calculate the timeline with the help of python scripting.

So these all are the parameters calculated during debugging.

CHAPTER-5

CONCLUSION AND FUTURE SCOPE

Now we are aware about the importance of validation and debugging in the IC industry, but it is a very much time consuming process that increases the time to the market and in this competitive era in the field of VLSI industry time to market plays an important role .

Here with the help of "AUTOMATED DEBUGGING" time to market is reduced and it gives a positive edge to an industry in this competitive era. Here we had applied both the way of debugging process on an industry level project and we analyse the following debugging/validation time difference.

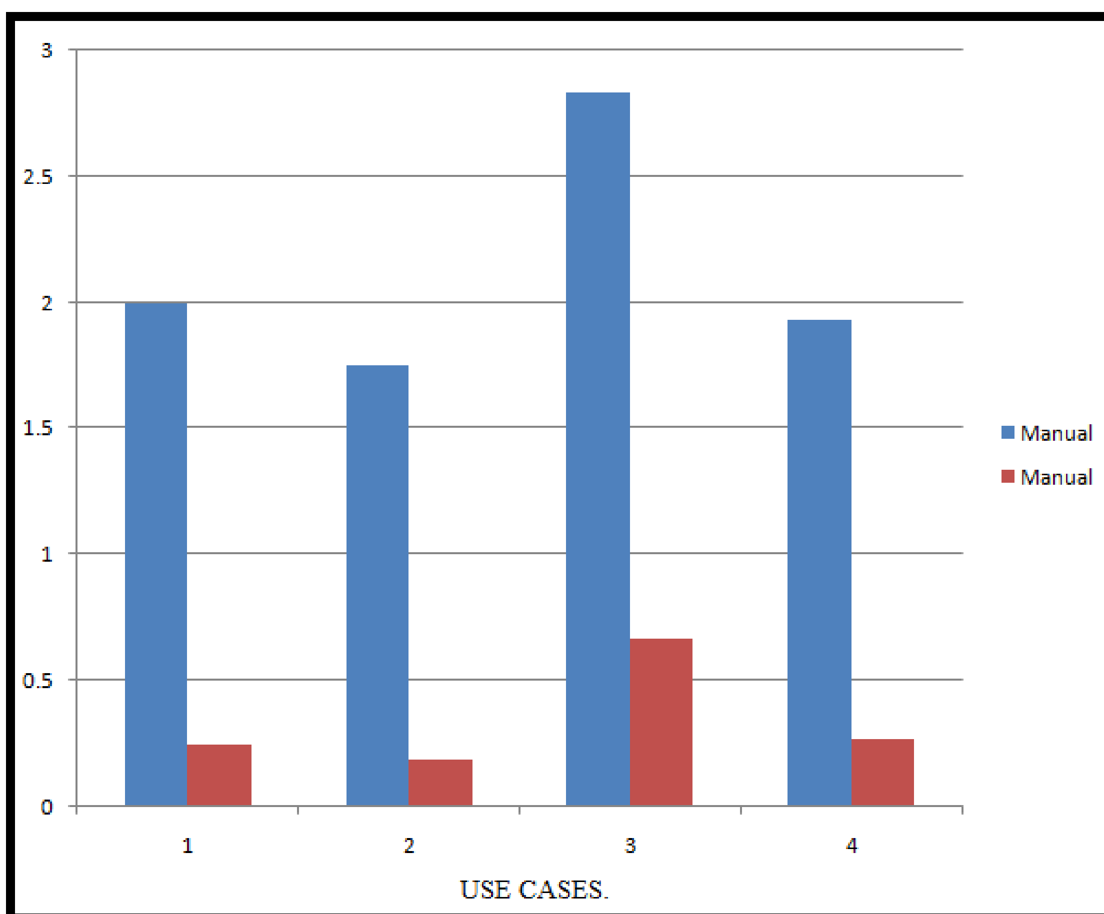


Fig. 5.1 Comparison of debugging time for both manual and automation debugging

Here we had not only crosschecked the debugging time but also we had compared the accuracy also, we had done the debugging for similar use case and matched the data as shown below:

Debugging	BANDWIDTH	LATENCY	FPS	TIMELINE	NO. OF STEPS	DEBUG TIME
Manual	1089.3	21.64	30	108.5	9	1.57
Automated	1089.2678	21.6378	30	107.89	2	.26

Table 5.1 Manual and Automated debugging comparison

So here we had seen that if we compare the manual and automated debugging then there is no variation in all parameters but we get 6 times less debugging time in automated debugging as compare to manual debugging.

Here this analysis is enough to focus on the future scope of this automated debugging tool, as day by day as complexity of the chip is increasing day by day where as the size of the chip is decreasing, so it is very difficult to go one by one for all 9 debugging steps, that time we have to prefer the automated debugging.

REFRENCES

[1] A. Veneris and I. N. Hajj, “Design error diagnosis and correction via test vector simulation,” IEEE Trans. on CAD, vol. 18, no. 12, 1999 , pp. 1803–1816

[2] P.-Y. Chung and I. N. Hajj, “Diagnosis and correction of multiple logic design errors in digital circuits,” IEEE Trans. on VLSI Systems, vol. 5, no. 2, 1997 , pp. 233–237.

[3] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, “Fault diagnosis and logic debugging using boolean satisfiability,” IEEE Trans. on CAD, vol. 24, no. 10, 2005, pp. 1606–1621

[4] A. Sülflow and R. Drechsler, “Automatic fault localization for programmable logic controllers,” in Formal Methods for Automation and Safety in Railway and Automotive Systems, 2010, pp. 247–256.

[5] A. Hopkins and K. McDonald-Maier, “Debug support for complex systems-on-chip: a review,” Proc. of Computers and Digital Techniques, vol. 153, no. 4, 2006, pp. 197–207.

[6] B. Vermeulen, T. Waayers, and S. Bakker, “IEEE 1149.1-compliant access architecture for multiple core debug on digital system chips,” in Int’l Test Conf., 2002, pp. 55–63.

[7] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, “A reconfigurable design-for-debug infrastructure for SoCs,” in Design Automation Conf., 2006, pp. 7–12.

[8] J.-S. Yang and N. A. Touba, “Expanding trace buffer observation window for in-system silicon debug through selective capture,” in VLSI Test Symp., 2008, pp. 345–35

[9] J. Melngailis, L. W. Swanson and W. Thompson, “Focused Ion Beams in Semiconductor Manufacturing”, Wiley Encyclopedia of Electrical and Electronics Engineering, Dec. 1999.

[10] S.-J. Pan, K.-T. Cheng, J. Moondanos, and Z. Hanna, “Generation of Shorter Sequences for High Resolution Error Diagnosis Using Sequential SAT”, ASPDAC’06, pp. 25-29.

[11] S. Safarpour, A. Veneris, and H. Mangassarian, “Trace Compaction using SAT-based Reachability Analysis”, ASPDAC’07, pp. 932-937.

[12] A. Veneris and I. N. Hajj, “Design Error Diagnosis and Correction via Test Vector Simulation”, IEEE TCAD, Dec. 1999 pp., 1803-1816.

[13] H. Xiang, L.-D. Huang, K.-Y. Chao, and M. D. F. Wong, “An ECO Algorithm for Resolving OPC and Coupling Capacitance Violations”, ASICON’05, pp. 784-787.

[14] Y.-S. Yang, S. Sinha, A. Veneris and R. E. Brayton, “Automating Logic Rectification by Approximate SPFDs”, ASPDAC’07, pp. 402-407.

[15] J. Zhang, S. Sinha, A. Mishchenko, R. Brayton, and M. Chrzanowska-Jeske, “Simulation and Satisfiability in Logic Synthesis”, Proc. IWLS’05, pp. 161-168

VLSI

by Manish Kumar

Submission date: 03-Jul-2019 04:02PM (UTC+0530)
Submission ID: 1148923455
File name: 2k17vls12.pdf (1.24M)
Word count: 11830
Character count: 56182

VLSI

ORIGINALITY REPORT

1%	0%	1%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to RMIT University Student Paper	1%
2	Ehab Anis, Nicola Nicolici. "On using lossless compression of debug data in embedded logic analysis", 2007 IEEE International Test Conference, 2007 Publication	<1%
3	Submitted to Nanyang Technological University, Singapore Student Paper	<1%
4	"Functional Design Errors in Digital Circuits", Springer Nature, 2009 Publication	<1%
5	Submitted to University of Southern California Student Paper	<1%
6	Muroga, Saburo. "Full-Custom and Semi-Custom Design", LOGIC DESIGN, 2003. Publication	<1%
7	Anis Daoud, Ehab, and Nicola Nicolici. "On	<1%

Using Lossy Compression for Repeatable Experiments during Silicon Debug", IEEE Transactions on Computers, 2011.
Publication

Exclude quotes On
Exclude bibliography On
Exclude matches < 10 words