EFFICIENCY ANALYSIS ON CORE LEVEL VALIDATION AND BACK-END LAYOUT DESIGN

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Jay Shukla
(13MECE17)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2015

EFFICIENCY ANALYSIS ON CORE LEVEL VALIDATION AND BACK-END LAYOUT DESIGN

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Jay Shukla
(13MECE17)

Under the guidance of

External Project Guide:

Tejinder Singh Syan
Engineering Manager,
Intel Technology India Pvt. Ltd.,
Bangalore.

Internal Project Guide:

Dr. N. P. Gajjar
Professor (EC Eng.),
Institute of Technology,
Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2015

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgment has been made in the text to all other material used.

- Jay Shukla
13MECE17

Disclaimer

"The content of this thesis does not present the technology, opinions, beliefs, or positions of Intel Technology India Pvt. Ltd., its employees, vendors, customers, or associates."

Certificate

This is to certify that the Major Project entitled **"EFFICIENCY ANALYSIS ON CORE LEVEL VALIDATION AND BACK-END LAYOUT DESIGN"** submitted by **Jay S. Shukla (13MECE17)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place:Ahmedabad

Internal Guide & Course Co-ordinator

Dr.N.P.Gajjar
(Professor,EC)

Section Head,EC

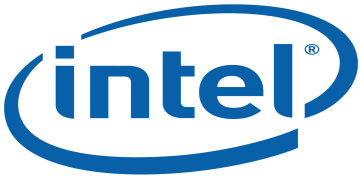
Dr.D.K.Kothari
(Professor,EC)

HOD

Dr.P.N.Tekwani
(Professor,EE)

Director

Dr.K.Kotecha
(Director,IT-NU)



Intel Technology India Pvt. Ltd.

Certificate

This is to certify that the Major Project entitled **"EFFICIENCY ANALYSIS ON CORE LEVEL VALIDATION AND BACK-END LAYOUT DESIGN"** submitted by **Jay S. Shukla (13MECE17)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place:Bangalore

Mr.Prasad Mahajan,
Physical Design Engineer,
Intel Technology India Pvt. Ltd.,
Bangalore.

Mr.Tejinder Singh Syan,
Engineering Manager,
Intel Technology India Pvt. Ltd.,
Bangalore.

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr.P.N.Tekwani**, Head of Electrical Engineering Department, and **Dr.N.P.Gajjar**, PG Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I am deeply indebted to my thesis supervisor **Dr.N.P.Gajjar**, Professor, E.C.Dept., Nirma University and my managers at Intel Technology India Pvt. Ltd., **Mr. Rajat Gupta**, **Mr. Tejinder Singh Syan** and **Mr. Saurabh Sharma**, for their constant guidance and motivation.

I also wish to thank **Mr. Ravishankar Ramaswamy**, **Mr. Sekhar Katta**, **Mrs. Sunitha Chinnasamy**, **Miss Deepthi Chilukuri** and all other team members at Intel for their constant help and support. Without their experience and insights, it would have been very difficult to do quality work.

I also take this opportunity to thank **Mr. Rajesh Navale**, **Mr. Mahajan Prasad**, **Mr. Sreenivas Subramanian** and **Mrs. Poonam Srinivas** for helping me to complete my project work successfully through their experience and suggestions.

I wish to thank my friends of my class for their delightful company which kept me in good humor throughout the journey.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

- Jay Shukla
13MECE17

Abstract

Today's micro-processors contain highly complex and sophisticated design on to a single chip. The higher complexity can increase the bugs in the chip which ultimately affects the functionality of the chip. The complexity in the layout of nanometer scale design also increases the time to market. Therefore, new microprocessors demand efficiency improvement at various stages of VLSI design cycle which can be achieved through automation. The efficiency in VLSI design cycle can be improved at core level validation and back-End layout design.

The pre-silicon validation plays an important role in order to minimize the bugs in the chip design before giving it to the fabrication facility. Pre-silicon validation, a multi-step process, verifies if a chip design sticks to the pre-defined specification and satisfies the designer's objective. The new generation processors demand the high amount of efficiency and coverage in pre-silicon validation.

Firstly, efficiency improvement of core level validation for the three different features AFD (Array Freeze and Dump), PMON (Performance Monitoring) and SMM (System Management Mode) can be achieved through randomization and intelligent algorithms. AFD, the feature of DFT (Design for Testability), compares the expected values of all the arrays of various clusters with its RTL (Register Transfer Level) values at the freeze time and dumps the correct value if the matching is incorrect. The randomization of array freeze and dump in pre-silicon validation helps to check all the arrays in a single test and makes to get closer to post-silicon debug scenario. The performance monitoring monitors the various events at the cluster level and at the core level. To increase the coverage of validating those events in performance monitoring unit at pre-silicon validation, the concept of performance monitoring multiplexer is introduced. This algorithm helps to achieve the coverage of 81.01% with lesser amount of test cases. At last, the automated validation strategy for system management mode is suggested which can help to do debug easier with saving valuable amount of time. All these improvements of the feature validation tries to make the design with minimal bugs before tape-out.

Secondly, the efficiency improvement of back-end layout design is described. The physical design of the chip is one of the important phase in VLSI design cycle. The increment in complexity of physical design of the chip is due to continuous shift towards nanometer scale in the design. Due to some floorplanning and placement changes, re-routing should be required again with respect to new placement of input and output

logic cells. These repetitive and manual efforts are tedious and cumbersome.

The proposed automated group routing algorithm exploits modular tile based placement and simplified track patterns effectively. It has reduced routing time by 2x with better control on routing resources and automating the routing at the same time. The algorithm can be improved with merging more physical properties and various design rule checks.

Thus, different strategies like Randomized AFD Testing Mechanism, Performance Monitoring Multiplexer, Automated Validation of System Management Mode and Automated Group Routing Strategy can be used to improve efficiency at pre-silicon validation and at back-end layout design.

Contents

List of Figures

List of Tables

Acronyms

AFD ..... Array Freeze and Dump
DFT ..... Design For Testing
DRC ..... Design Rule Checking
DUT ..... Device Under Test
ERC ..... Electrical Rule Checking
FDIV ..... Floating Point Divide
FUB ..... Functional Unit Block
HDL ..... Hardware Description Language
HVL ..... Hardware Verification Language
ITRS ..... International Technology Roadmap for Semiconductors
JTAG ..... Joint Test Action Group
MSR ..... Model Specific Register
LVS ..... Layout Vesus Schematic
PEBS ..... Precise Event Based Sampling
PMON ..... Performance Monitoring
PMU ..... Performance Monitoring Unit
PNR ..... Place And Route
RTL ..... Register Transfer Level
SMI ..... System Management Interrupt
SMM ..... System Management Mode
TCM ..... Time Consuming Method
VHDL ..... VHSIC Hardware Description Language
VHSIC ..... Very High Speed Integreted Circuit
VLSI ..... Very Large Scale Integration

Chapter 1

Introduction

1.1 Motivation

The design complexity of the new generation microprocessors increases the amount of bugs into the released products. These escaped errors can produce dangerous effects on the stability and security levels of the systems, undermine the reputation of the manufacturing companies and cause considerable financial trouble. The recent trends of complex memory subsystems and non-deterministic communication delays in multi-core processors, exacerbate the problem. On the other hand, as the chip area advances to nanometer technology, the complexity of the interconnect design also increases with great extent. In compact microprocessor chip, the manual efforts of routing and design rule checking are also increased. To overcome these types of worsening situation, the processor needs automation with high-coverage and high-efficiency of validation and routing respectively.

1.2 Project Overview

Pre-silicon functional validation techniques provide platform to find out bugs before giving the chip design to fabrication facility. If more number of features are validated at pre-silicon validation with higher coverage, the chances of bugs in post-silicon validation can be minimized. Sometimes, it happens that the simulation takes hours of time in pre-silicon validation to validate the event which is actually takes only few minutes in the real-time. This problem can be solved by increasing validation efficiency of the feature. In the project, it is shown that how efficiency of core level pre-silicon validation can be improved for different features like Array Freeze and Dump, Performance Monitoring and System Management Mode.

AFD (Array Freeze and Dump), feature of DFT (Design for Testability), allows to

freeze the state at single point in time and extract arrays data through control register buses for post-silicon debug usage. To make DFT feature more efficient, it should be necessary to check the data of arrays are matching with its expected value or not at the time of signal freeze within the single test to handle all the arrays. It can be achieved successfully at post-silicon validation only if this kind of analysis done at pre-silicon validation first.

Second part shows the improved validation strategy for performance monitoring. Performance monitoring provides a mechanism to check the utilization of the available hardware resources for various programs and applications. The validation of this feature is required at pre-silicon validation due to its high capability of monitoring internal micro-architectural events. Previously, the pre-silicon validation was not able to validate all the events of the cluster in performance monitoring with the defined number of test cases. So, the concept of performance monitoring multiplexer is introduced which intelligently assigns the events to performance monitoring counters to achieve more efficiency and coverage with lesser number of test cases.

In the later part, SMM (System Management Mode), one of the feature of IA-32 architecture, is described. System management mode plays a major part in handling of platform-specific power management or thermal events. The validation of this feature requires different test cases and debugging of those test cases can be done more easily if the script can somehow help in automated debugging.

The last part of the thesis describes the different routing strategies and required manual efforts to route the nets in the processor chip. To reduce manual effort and to save valuable amount of time, automated group routing algorithm is introduced. It considers different physical properties like routing length, high fanout net, thin hierarchy nets and multi-instance nets while doing group routing for various nets. The strategy can help to do quality routing with less congestion and encourage for exploring different floor-plan possibilities.

1.3 Thesis Organization

The objective of the thesis is to introduce different strategies which can ultimately improve efficiency of various features in pre-silicon validation and back-end layout design.

Chapter 2, *Rudiments of VLSI Design*, introduces with typical VLSI design flow.

Chapter 3, *Randomized AFD testing mechanism*, describes the scanning mechanisms can be used for DFT. It also explains the random testing mechanism to validate all the arrays in the processor.

Chapter 4, *Performance Monitoring Multiplexer*, explains the performance monitoring unit architecture and shows that how coverage can be improved with combination of proposed multiplexer using minimum number of test cases.

Chapter 5, *Automated Validation of SMM*, introduces with one of the processor feature, System Management Mode and suggests automated validation of it.

Chapter 6, *Automated Group Routing Strategy*, explains different routing strategies and introduces automated routing strategy to reduce manual effort in layout.

Chapter 7, *Conclusion and Future Scope*, summarizes the thesis work.

Chapter 2

Rudiments of VLSI Design

2.1 Introduction to VLSI Design Flow

A traditional microprocessor's design to manufacturing flow, shown in **Figure 2.1** consists of various steps that includes a high-level description of processor design specification, structural specification and physical specification, and then finally, implements the specified functionalities on a silicon die at fabrication facility.

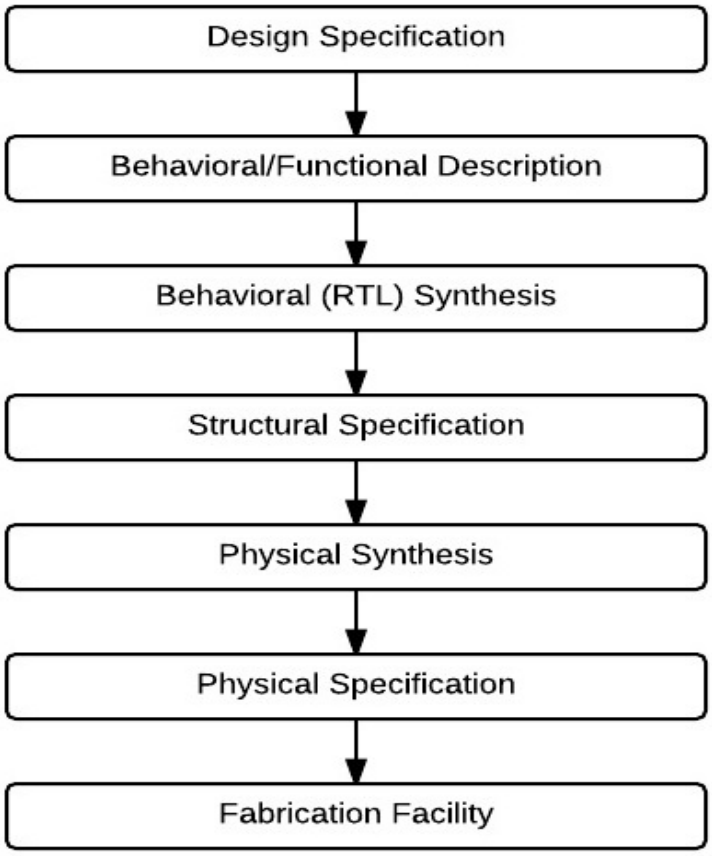


Figure 2.1: VLSI Design Flow

The VLSI design flow starts with required functionalities and characteristics of the microprocessor, and how to interface with other digital systems. The architectural model is based on this predefined specification which is written in a high level programming language[1]. This model then converted into any Hardware Descriptive Language like VHDL, Verilog etc. The codes written in the hardware description language design describes the operation of individual sub-modules of the processor with the and is also known as the register-transfer level (RTL) model[1]. This RTL model is then verified to establish its equivalence to the architectural model through simulation based and formal techniques[1]. These simulation based values are then compared with actual architectural known values, called as pre-silicon validation or verification. The flow till this point is also known as "Front End". The next step is structural specification to implement this design through synthesis software which maps the logical cells and different nets according to logic for creating netlist of the design[1]. The physical synthesis and specification stages take care of place and route of the logical cells of the netlist can be placed on the silicon chip to fulfill the requirement of the timing, power and die size. From front end to this step is called as "Back End". And finally, the design is sent for manufacturing to fabrication facility.

2.2 Fundamentals of Validation

2.2.1 Different Validation Strategies

As described in section above, the processor has to go under several stages before going to the manufacturing. The validation also continues at various levels with that flow to check the functionality of the processor. In general, there are three types of validation done for the processor. Those three validation strategies can be more understood at various stages through **Figure 2.2.1**.

Firstly, the pre-silicon stage comes in which the specified architectural model converted into register transfer level model in hardware description language (HDL). This RTL model then converted into design netlist through synthesis. After the generation of the netlist, the place and route of the logical gates and wire connection is done which is the actual placement of the logic in the silicon die[1]. Thus, prototype of the processor is manufactured after achieving all the timing and power requirements of the design. Once prototype is ready, the post-silicon validation is done on that to validate all the features and functionality of the chip. Then, the mass production of the processor happens and the processor is released and becomes available to the market.[1]

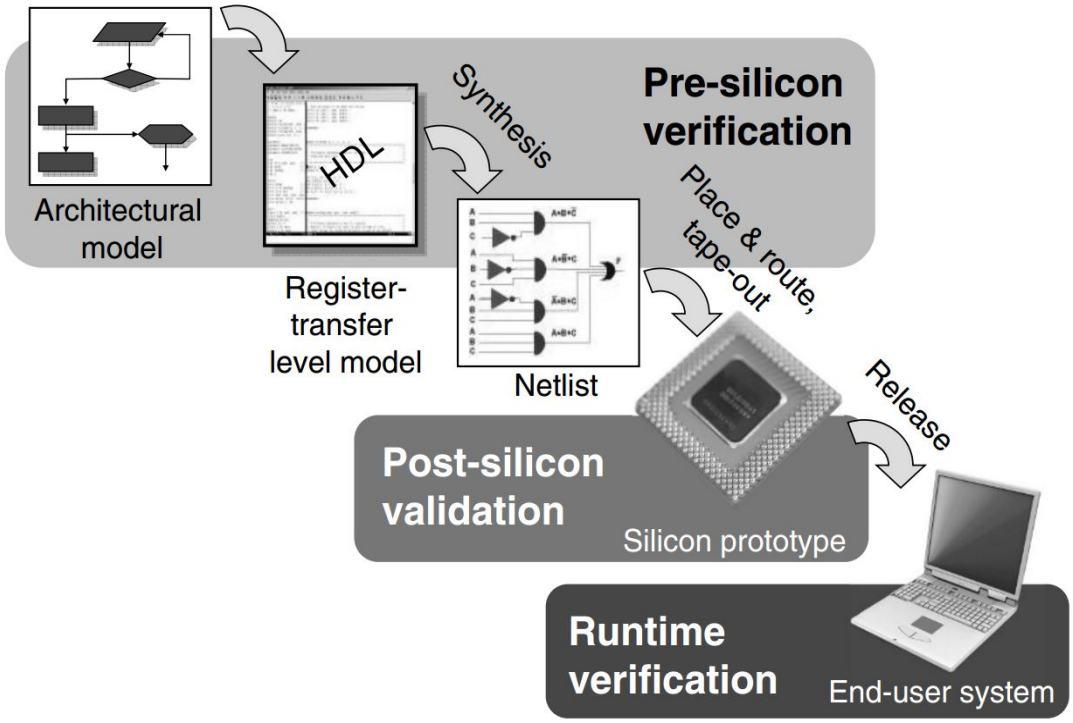


Figure 2.2: Different Validation Strategies[1]

2.2.2 Various Components of Validation

As discussed so far, it is easy to understand that the main purpose of the pre-silicon validation is to verify the processor functionalities with respect to design specification. Pre-silicon validation process follows certain steps to verify the processor. The different components of validation is shown in **Figure 2.2.2**.

The generator, the initial step in the pre-silicon validation flow, which generates all different possible relevant input data to the processor design chip with some useful constraints and data structures[2]. When testing of the chip is going on, the one should specifically concentrate on pre-defined standard protocol and specification of the design[2]. Suppose, for example DUT is going to handle all types of USB (Universal Serial Bus) packets, then the generator should be able to generate all possible USB packets like, TOKEN, DATA and HANDSHAKE. It should also able to generate the error packets like that contains CRC (Cyclic Redundancy Check) error[3]. The specific test contains additional constraints, whose purpose is to direct the generation of the input stimulus to a specific area. For example, if you are constraining the packets to be of one of the TOKEN or DATA, then the generator will generate those two packets only, thereby limiting the area of interest. If you are not giving any specific constraints then there is a requirement of many different test cases.[3]

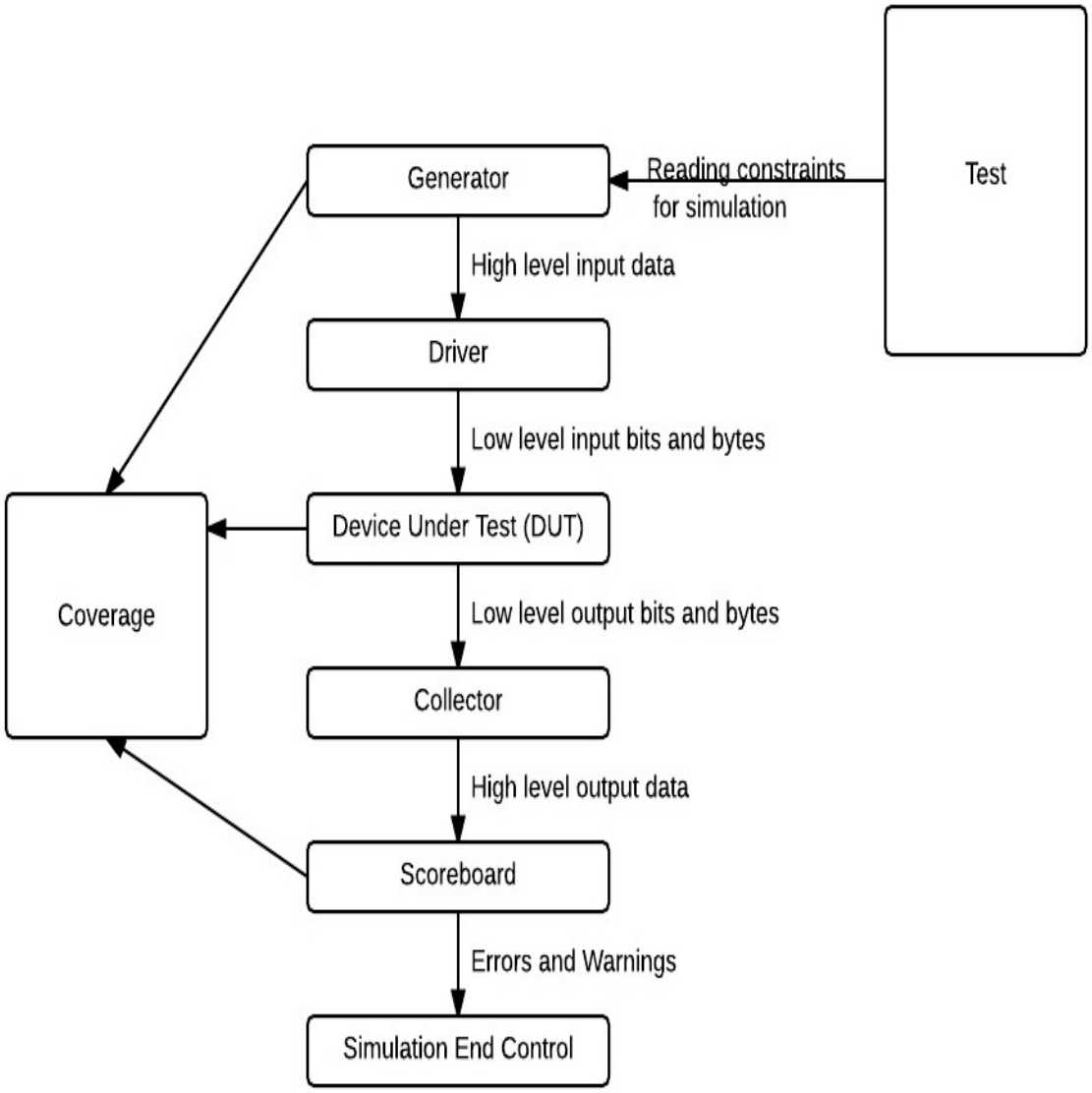


Figure 2.3: Different Components of Validation

After the valid input stimulus are generated as mentioned in the generator section, now it is time to inject it into the DUT. The driver object performs the function of taking one stimulus item at a time and injects into DUT until all the stimulus has been injected[3]. The high level input data is given to the driver and then converted into list of bits and bytes[3]. Thus, physical level protocol is implemented by the driver-send the SETUP token and DATA packet that contains the information about how to configure the USB device, wait for some clock cycles as per the spec, receives the acknowledge back from the USB device[3]. This output data is given to the collector component.

The collector is on the output side of your DUT. The main function of this is, to collect the low level data of bits and bytes from the DUT and convert it into high level

data[3]. This high level output data is transferred to the scoreboard, the important and crucial component of the test bench, to check the correct output data came out as expected and on time.[2]

The process of validation should be end at one point or at the another. After the completion of the scoreboard stage, the errors and warnings are sent to the simulation end control. Scoreboard also send the results to the coverage for checking the how much functionality is covered through the applied components. The coverage will have information from the generator, the device and the scoreboard.[2]

Coverage will provide analysis on actual testing points covered by the validation environment. The framework for simulation based validation is shown in **Figure 2.4**.

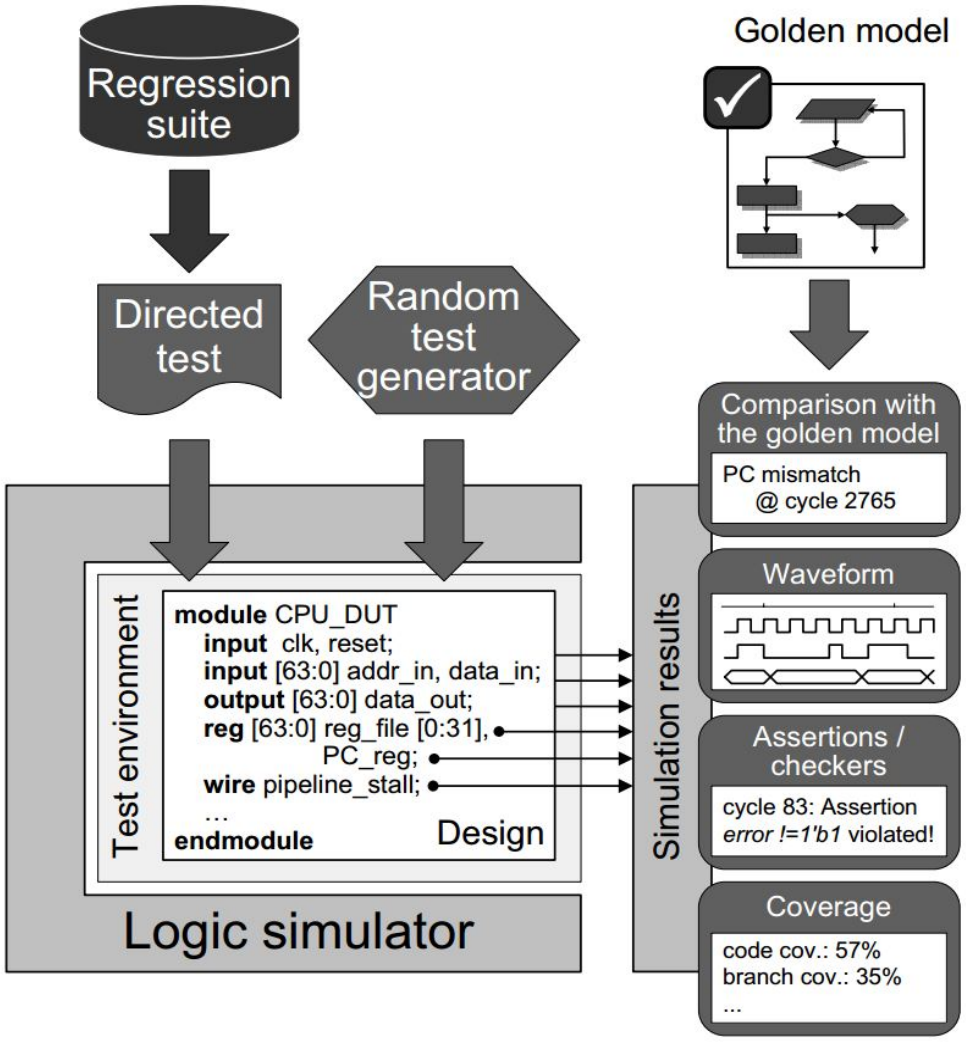


Figure 2.4: Typical framework for simulation-based validation[1]

The Coverage is one method to observe how much affective the applied components[3]. There are majorly two types of coverages- objective coverage and subjective coverage. Objective coverage includes code coverage to check how much lines of code covered and toggle coverage to check which signal toggled. Subjective coverage is for interesting scenario we want to hit. It includes frequency coverage to monitor set of event frequency and functional coverage to monitor user defined scenarios. For example, in USB verification environment, the input checker checks whether all the USB packets are generated or not[3]. Output coverage shows that all the parts are tested in the design or not.

2.2.3 Pre-silicon Validation approaches

Validation process approaches the problem of finding flaws in different ways. there are basically three functional verification approaches, namely black-box, white-box and grey-box approaches.

Black-box:

In this approach, we don't care about what exactly is the system made of. We care only about the functionality of the DUT and we must be able to predict the outputs based on the input. The black-box can be chip or an unit of chip.

White-box:

It is opposite to black box approach. Here we have the knowledge of exact implementation of the unit. Hence we have full control of the behavior of the unit and therefore we can program for some interesting errors.

Grey-box:

Grey box verification means that a limited number of facilities are utilized in a mostly black-box environment. Most environments are example of grey-box approach. Prediction of correct results on the interface is occasionally impossible without viewing an internal signal.

2.2.4 Importance of Validation

Processors are capable of performing computation at astonishingly high speeds and are extremely integrated, occupying only a few square centimeters of silicon die. The task of verifying a modern microprocessor and guaranteeing the correctness of its operation

is increasingly challenging, even for most established processor vendors.[1]

As the number of features grows, in some cases full-system simulation may become unfeasible. Likewise, increasing capabilities of formal verification tools in the future will be outpaced by the complexity of critical modules requiring formal analysis. In this worsening situation, the number of total bugs in processor products and the speed with which they are discovered in the field is rapidly increasing.

The FDIV (Floating point devision) bug of Intel P5 Pentium floating point unit incurred a loss of $475 million to the chip manufacturer[4][5]. The bug resulted in incorrect return value on floating point division operation due to the missing entries in the look up table[1]. Another case, an error in the translation look-aside buffer of the third level cache in the Phenom processor by AMD, forced the manufacturer to delay the market release by several months[1][6]. Not only this delayed the distribution of the product to the market, but also created much negative publicity for the company, and influenced the price of its stock. This concern voiced in the International Technology Roadmap for Semiconductors, which states that "without major breakthroughs, verification will be a non-scalable, show-stopping barrier to further progress in the semiconductor industry". ITRS also reports that there are no solutions available to provide high quality verification of integrated circuits and sufficiently low rate of escapes beyond the year 2016.[1]

From above, it is clear that because of the expanding gap between complexity and verification effort, in the future errors will continue to slip into silicon, potentially causing much more damage than the infamous FDIV bug[4]. To deliver always higher performance to end-users, processor manufacturers are forced to design progressively more complex circuits and employ immense verification teams to eliminate critical design bugs in a timely manner.[1]

2.3 Fundamentals of Physical Design Flow

2.3.1 Physical Design Flow

Physical Design is a very important step in VLSI design flow after which the chip is sent to the manufacturing facility. Physical design stage is a combination of physical synthesis and physical specification. The structural specification implement the design through synthesis software which maps logic gates, registers and wires, and, generates a netlist of the circuit. This netlist contains the details of the different cell used and their interconnections through out the circuit. After verifying the functionality of the

netlist and its timing, it is sent to physical design flow.

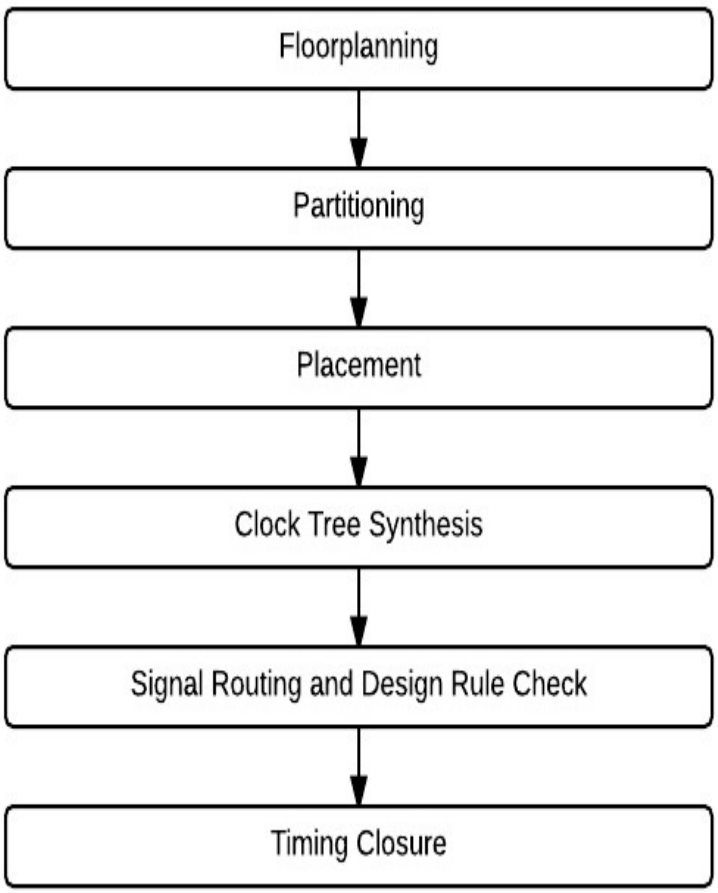


Figure 2.5: Physical Design Flow

As shown in **Figure 2.5**[7], physical design flow contains various steps like floorplanning, partitioning, placement, clock tree synthesis, signal routing, design rule checks and timing closure. All of them are described in detail as below.[8]

Floorplanning:-

It is an essential design step for hierarchical, building-module design methodology because it determines chip quality. It provides early feedback about architectural decisions, estimates chip area delay as well as congestion caused by different wiring. The defining size of the chip, allocating power routing resources and spacing of different standard cells are decided in this phase. All the subsequent stages are dependent on how good floorplan it is. The various iterations should be required before arriving at one optimum floorplan of the chip.

Partitioning:-

It enables structural implementation of big complex system into number of small blocks by using divide and conquer approach. It also requires to handle engineering change orders. For huge systems, design iterations require very fast turn around time. A hierarchical partitioning methodology can localize the modifications and reduce the complexity as well as routing. The partitioning of different small blocks is also referred as logical partitioning.

Placement:-

It is most crucial step of the physical design flow as the interconnection of the cells and timing requirement of the system mostly dependent on the placement of the logical cells. It also determines the routability of the design. There are many reasons for considering placement as critical step like determining the performance of the circuit, distribution of the heat on a die surface and power consumption due to placement. The good placement of the cell can play major role in reducing capacitive load of the wires.

Clock Tree Synthesis:-

Clock tree synthesis is for meeting design constraints mainly in multi-clock system design. Generally, it is done after the placement of the cells and before the routing of the signal nets , to utilize best routing resources for the clock signal. Clock trunk and spines are routed depending upon the system topology. A good clock tree network will minimize the clock skew and jitter.

Signal Routing and Design Rule Checks:-

The main objective of the routing is to complete all the required connections, otherwise the chip would not function well and may even fail. The other objective is to reduce the routing wire length. There are two types of routing: global routing and detailed routing. The routing should also be prioritized in order of power routing, clock routing and signal routing. After completion of routing for the each functional unit block, the physical design verification of that is very necessary. There are different types of physical verification methodologies like Design Rule Checks(DRC), Layout Versus Schematic(LVS), Antenna Rule Checking and Electrical Rule Checking(ERC). The design rule checks contains checking of shorts, space violation, off-grid, min-length, via to via violation, corner to corner violation etc.

Timing Closure:-

After all the above steps are passed successfully, timing of the all the nets should be checked which should satisfy the timing requirement of the design. Then the chip design can be send to fabrication facility.

2.3.2 Physical Verification of the design

The physical verification of the design checks the different rules of the design which should not be violated before giving the design for tap-out. It includes Layout Versus Schematic(LVS), Electrical Rule Checking(ERC), Antenna and Design Rule Checking(DRC). LVS verifies functionality of the transformed geometries with the netlist of the design. ERC checks floating power pins connection as well as VDD and VSS connection of the functional unit blocks.[8] Antenna rule is checked to provide protection to the gate-oxide against the damage while manufacturing process.[8] The various design rule checks are as below:

Short: When two different nets are routed on same metal grid, metal short is created. Short can also be between signal routing, via, power lines, etc.

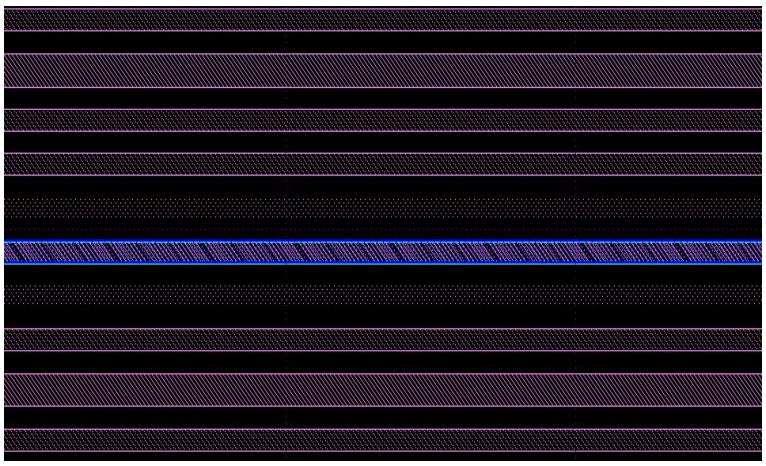


Figure 2.6: Short Violation

Space: Space violation can further classified into two categories. Min space is the minimum allowed spaced between tow wires in same metal layer. Max space rule provides maximum allowed separation between two wires in same metal layer. This rule is introduced to obtain uniform metal fill.

Min length: Min length rule provides the minimum length of the wire required in particular metal layer.

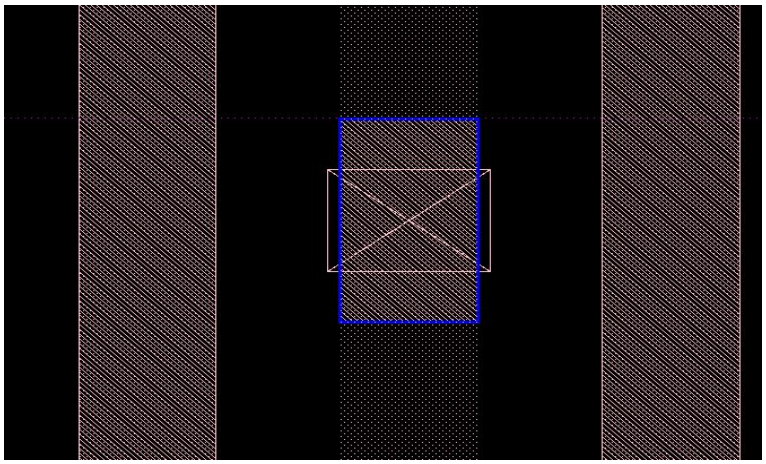


Figure 2.7: Min Length Violation

Max width: Max width provides maximum allowed width of the wire in given metal layer.

Corner to corner: Corner to corner specifies the minimum separation required between corners/ends of two wires in same metal layer.

Min jog: Min jog violation is observed when the length of the jog is less than the threshold values. This is observed when two segments of the same net in same metal layer have different width.

Valid width: Valid width specifies the set of the allowed metal layer grid widths.

Via to via: Via to via rule provides the required minimum separation between two vias of the same metal layer. The vias are between same two metal layers.

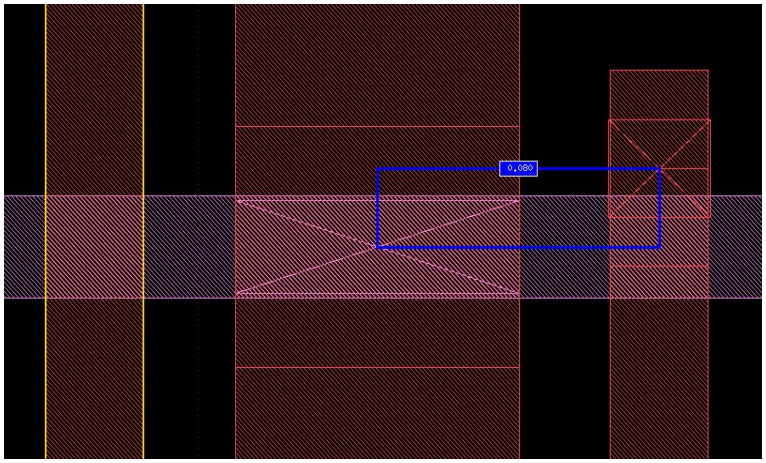


Figure 2.8: Via to Via Violation

Via to via n-1: Via to via n-1 is similar to the via to via rule. The only difference

is consideration of vias between one higher and one lower metal layer with respect of given metal layer.

Cut size: Via cut size rule provides valid via size for a given width of the metal layer.

Via coverage: There are many other rules are classified under this rule.This rule provides the minimum wire length required to cover up the via. If the wire is shorting with other wire or it is not on the proper grid or if the dimension of the via is not proper.

Non-fill: Non-fill violation is observed when the wire in given metal layer is not sitting exactly on the metal grid. During metal the region which have off-grid wires is left without filling.

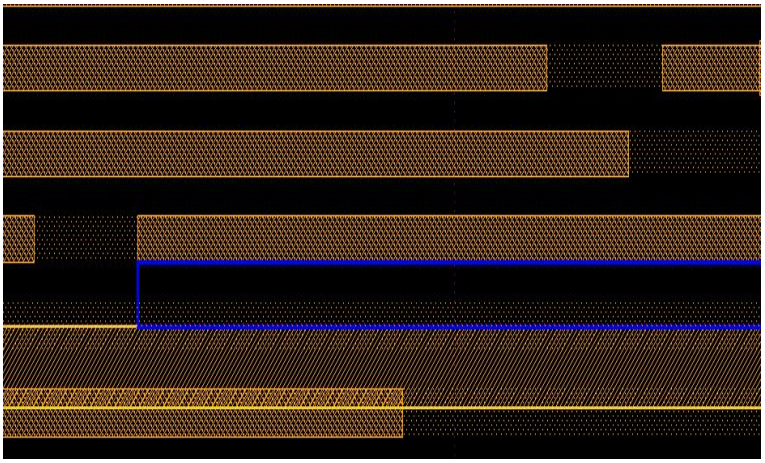


Figure 2.9: Non-fill Violation

Grid rules: Grid rules can be classified into several categories. Off grid issue is observed when the wire of given metal layer is not sitting on the grid or wire width is not valid.

2.4 Summary

In summary, this chapter introduces with the full VLSI design flow and different validation strategies like pre-silicon validation, post-silicon validation and run-time validation. The chapter includes the description of different components of pre-silicon validation and three various approaches in it. The later part focuses on physical design flow with its different verification strategies which is very important part from design perspective.

Chapter 3

Randomized AFD Testing Mechanism

3.1 Introduction to DFT

The AFD, array freeze and dump, is a feature of DFT (Design for Testability). The most exciting single day in the life of any semiconductor product is first silicon, the day when the first chip completes fabrication[9]. Pre-silicon validation attempts to find and correct as many bugs as possible before first silicon is created, but the slow speed of pre-silicon simulation requires years to test only minutes of actual operation. Sometimes to avoid this kind of testing time at pre-silicon validation, some of the features are only validated at post-silicon validation. But, to validate the feature at post-silicon validation often data of pre-silicon validation is used[9]. When at post-silicon validation bugs are detected, their cause must be found and fixed. The process of finding the root cause and eliminating design flaws in silicon is called silicon debug. To aid in post-silicon validation, silicon debug and silicon test, all modern processors add some circuits used specifically for testing. These are called design for testing (DFT) circuits.[9]

The main drawback of the modern processors is its decreasing observability and controllability. Without DFT circuits, the only signals that can be directly observed or controlled are the processor pins. The number of processor pins is increasing but far more slowly than the number of total transistors[9]. This makes it more difficult with each processor generation to directly read or write a specific node within design by the use of the pins alone. DFT circuits help by expanding observability and controllability[9]. The DFT circuits increase the cost of the design by taking up die area and consume leakage power, yet, they are used because of great advantages like reduction in number of steps to fix the bugs and time from tapeout to shipping.[9]

3.2 Differernt Types of Scan Mechanism

In design for manufacturability domain, the scan chain components are widely used techniques. There are three types of scan-chains.[1]

1. Scan flip-flop design
2. Hold-scan flip-flop design
3. Boundary-Scan Mechanism

3.2.1 Scan flip-flop design

To combine a scan chain mechanism in existing design, the data input (scan in) and output (scan out) with the enable line are attached to the filp-flops as shown in **Figure 3.1**.

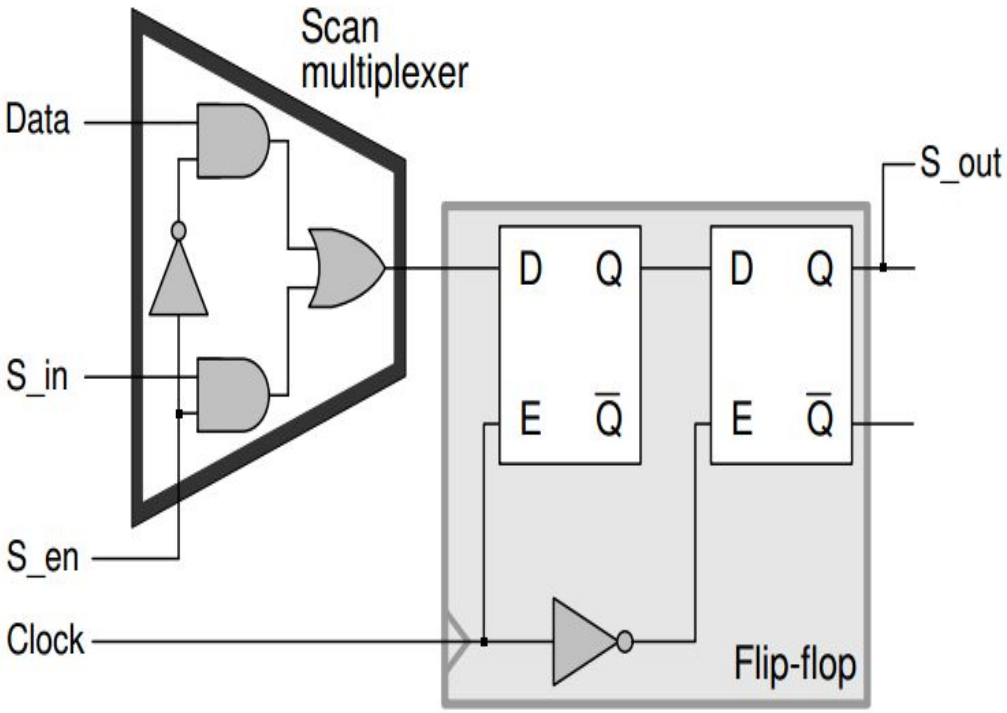


Figure 3.1: Scan Flip Flop Design[1]

The scan chain is having input and output at the starting and at the ending of it respectively with serial connection of the each scan in and scan out of the various storage elements. The processor works in normal mode when the enable signal of the scan chain mechanism is de-asserted. Here, the flip-flops operate as storage elements. While enabling the signal, the one flip-flop passes data to another serially and the whole

mechanism work as serial register. The validation engineer can get value of the cells with output wire through suspending the execution[1]. Moreover, an arbitrary internal state can be pre-set through the scan in functionality, enabling fine-grain controllability of the device, in addition to observability. There can be variation in different scan techniques from partial chain, full scan and multiple parallel scan in which some of the flip-flops are connected, all the elements are connected and parallel arrangement of the scan chain respectively. The downsides of the scan chain are additional die area for logic implementation, very slow latch operation and the requirement of the several additional clock cycles for suspending the current state of the device under test and enabling the new state.[1]

3.2.2 Hold-scan flip-flop design

To overcome the additional clock cycle issue of changing the states of the system, the more complex modern design scan chain can be used.

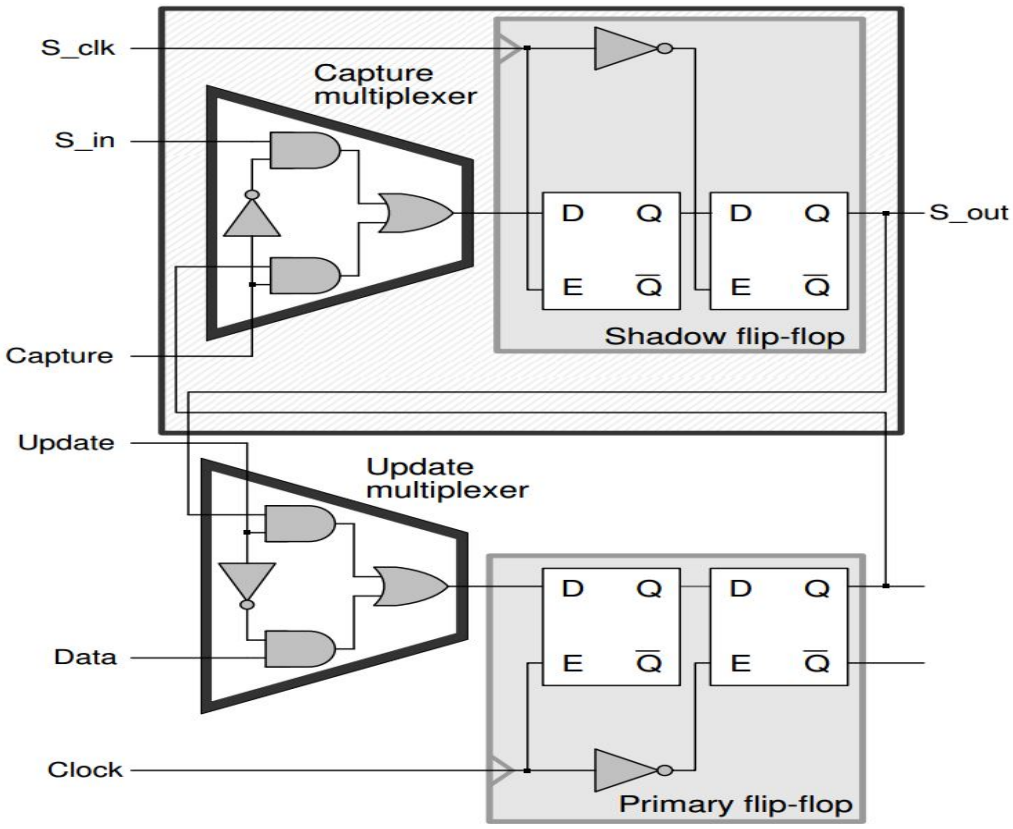


Figure 3.2: Hold Scan Flip Flop Design[1]

Hold-scan flip-flops consists of two basic scan flip-flops connected together, called a primary and a shadow flip-flop, as shown in **Figure 3.2**.

This mechanism uses the concept of the capture line and update line. The main aim of the capture line is to sample the shadow elements and to store the vale of the main latch. To route the sample data outside the device, the reconfiguration of the flip-flop can be done as and when required. Meanwhile the device can work normally and captured data can be taken out as the chain focuses only one shadow flip-flop elements with separate clock[1]. With asserting the update line, the new state can be activated on the shadow latches without affecting the operation of the device. The area required by the hold-scan chain mechanism is quite large as compared to the scan flip flop design, but, the flexibility of the debugging process using this mechanism is much more. So, the usage of this mechanism is frequent in the microprocessors.[1]

3.2.3 Boundary-Scan Mechanism

Boundary-scan is another technique often used in structural testing and validation, to allow individual modules of the processor to be tested in isolation.

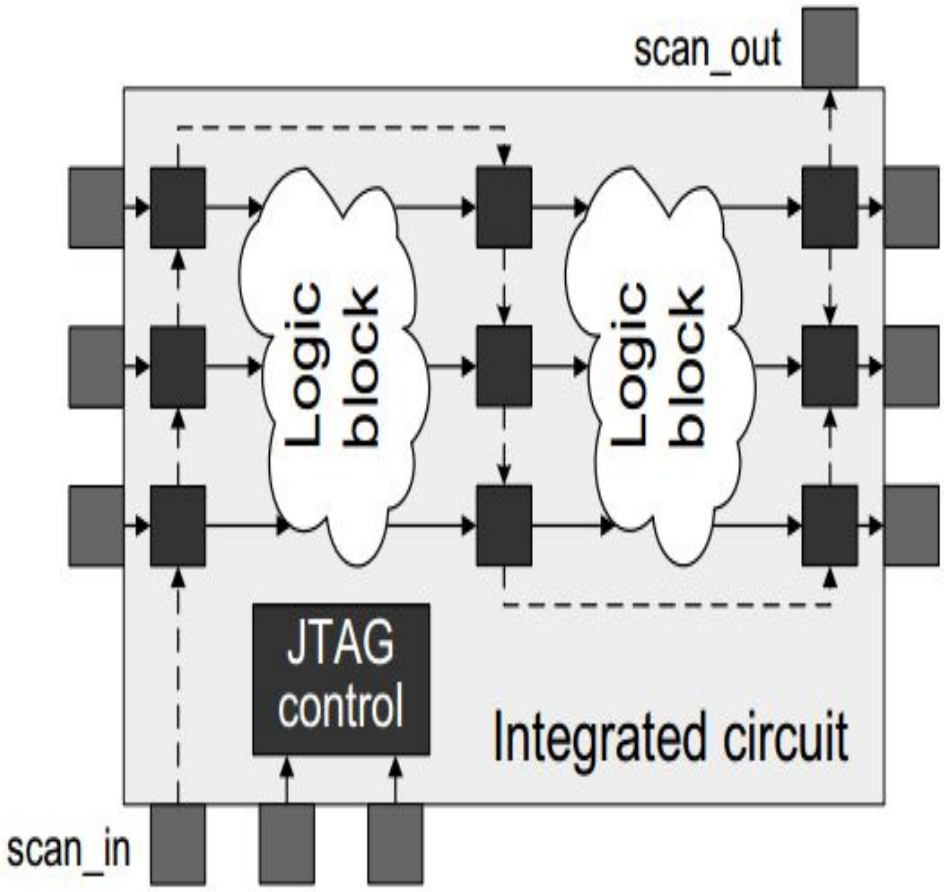


Figure 3.3: Boundary-Scan Technique[1]

To test the complex circuit design boards, the boundary-scan chain mechanism was introduced by the Joint Test Action Group (JTAG). In this, there is JTAG control which is connected to different logic blocks and scan in and scan out of the integrated circuit, as shown in **Figure 3.3**.

These cells carry different functional specification based on the requirement of the design circuit. Through the inputs of the block, various stimulus can be provided to the block as well as the verification of the block can be done[1]. The outputs of the block connected to boundary-scan flip-flop provide the results of the given stimulus. From the boundary scan chain mechanism, today, it is possible to enable various testing operations of the chip and also accessing the memory or various registers of the chip.[1]

3.3 Array Freeze and Dump

AFD (Arrays Freeze and Dump) is a design for testing feature which allows to freeze the state at a single point in time and extract all the arrays data through control register bus for post-silicon debug usage. To cover full flow of AFD, we need to find such a break-point at which we can hold the ongoing operation and fetch all the data, so that we can compare that with the expected data value at that time.

3.3.1 Introduction to AFD

The processor contains large amount of internal arrays spread all over the clusters which contain data and information for respective cluster. The clusters are having different arrays like, re-order buffer, memory order buffer, mid-level cache, page miss handler, etc. A re-order buffer, used in a tomasulo algorithm, allows instructions to be committed in-order. For re-order buffer, array is having all the stages information of issue, execute and write result with instruction type, destination, result and validity. A memory order buffer in modern microprocessor is used to utilize cache and memory banks fully. It contains different memory bank information. Likewise all the arrays contain different information regarding their states and cluster classification.

For example, pentium pro processor is having 39 internal arrays[10]. The arrays are having different size. These range in size from the 8-Kbyte, single-ported instruction and data caches down to highly multi ported register files with only a few hundred storage elements. The information of the various states of the arrays are very much important and crucial for the debug success. To observe the array contents, there is a requirement of cost-efficient debug mechanism.[10]

3.3.2 AFD Validation Mechanism

To validate the state and data information of the arrays, we need have some design for testing mechanism. Therefore, array freeze and scan dump, a design for testing feature, is used for comparing actual value which came from RTL with the expected value came which came from scan mechanism. The broadcasted freeze command in the processor is responded by each and every array for debug. When an array receives the freeze command, it immediately disables the write logic of it, so that the clock of the processor clock can still run, but, the contents of the array become freeze. The special enabling mechanisms at the arrays dump the frozen state of the processor.[10]

The processor can be returned back into normal execution mode only after the debug mode for the array is completed. In that debug mode, an interactive testing operations are done while keeping the processor stop from its particular running track.

The AFD validation mechanism can be done at two different levels in pre-silicon validation. First, AFD mechanism at each cluster level can be done. But, it can gradually increase complexity for handling the data as well as risk of not covering the full AFD flow and less chances to reach the post-silicon debug scenario. Second, super cluster level AFD validation, is a much more appropriate way to reach post-silicon debug scenario. Hence, this method also has drawbacks due to such a large number of array size and for intercepting the test dynamically with assurance of test passing with checker on normal test flow.

AFD testing mechanism freezes all these arrays information at particular period of time and their extraction of the information is done. Then, this information is going to be compared with expected values at that amount of time and if the state is not having the proper value, we can find out easily what causes the arrays to have faulty values. As mentioned earlier, we need to deal with large amount of arrays, the test writing for all them is very difficult task. To overcome this problem, randomized AFD testing mechanism is introduced at core level pre-silicon validation.

3.4 Randomization of AFD Testing

As mentioned earlier, we need to deal with large amount of arrays in the processor. To validate them, the test writing for all them separately can consume much more space. To overcome this problem, randomized AFD testing mechanism is introduced at core level pre-silicon validation as shown in **Figure 3.4**.

Randomization of AFD testing mechanism uses only a single test to handle all the

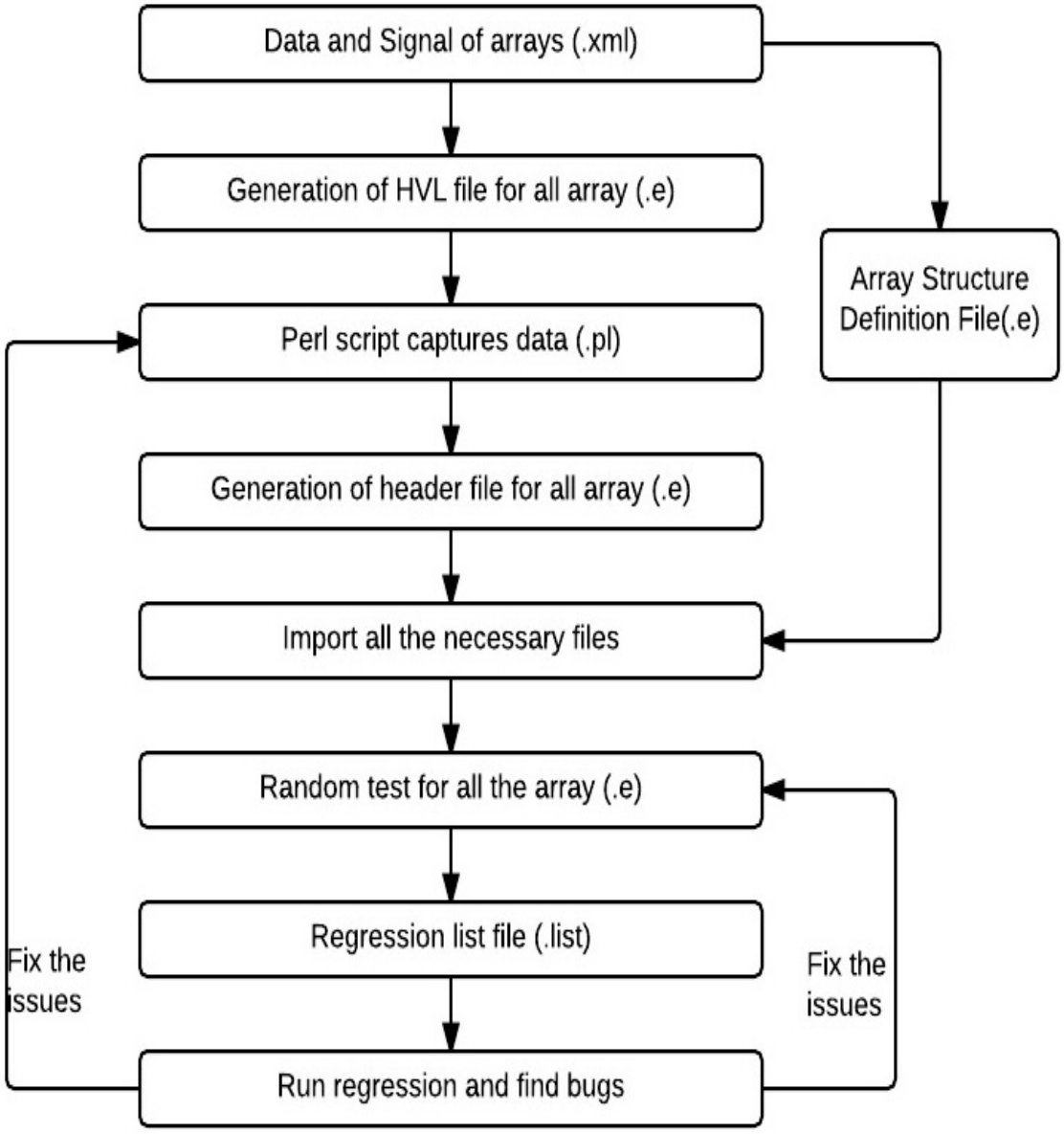


Figure 3.4: Randomization of AFD Testing

arrays of various clusters. Actually, all the arrays are having similar fields inside them like state information, data information, register bank information, freeze signal information, etc. So, instead of writing the tests for all of them separately, it is better to have randomized approach which can handle all the arrays one after the other not in any particular order. But, to make all the arrays work fine with single test will require the necessary unique data information from all the arrays before running the test. For that automation must be required.

First, hardware verification language (HVL) for files all the arrays are generated through data and signals from .xml files. Then, automation perl script can gener-

ated header file for only necessary fields and make new reduced form of HVL file from the original HVL array file. These files save large amount of memory. With the use of definition file and all the header files of the different arrays are imported into the random test file. Then, regression list which is generated automatically by script, is having each of the array listed out with required field to handle test. So, that only included arrays in that list are going to become a part of testing. In random test, at freeze state, the actual RTL value is compared with the expected one. If we get the same result, then the test will pass. Otherwise, we need to resolve the errors either in the automation side or in the random test.

3.5 Challenges Faced During Implementation

During the implementation of the randomized AFD flow, the biggest challenge is to pass all the arrays with the single test. Sometimes, the arrays are such big that the test takes a lot of time to check the whole array. First of all, I have tried to pass all the arrays separately with the test. In that, I have faced problem with the mismatch of fields declared in the header files and definition files. The thorough understanding of definition file helped to resolve those errors. Then, I came across the errors like the signal and data information were not specified in the header files perfectly. I have resolved those errors with fetching the fields more powerfully from the respective array files. Then, the test encountered new error suggesting that the signal names with particular bits and proper bank information is required. To resolve that I have written method of hardware verification language [11] in the header file which can take care of bank and bits information. This method helped to take the proper signal for the array. I have checked all the arrays separately with the test several times, the result showed that all the arrays are working fine. Then, I have tried to combine all the arrays in a single test. There, I had to maintain the flow of setting up all the arrays one by one and checking data for those data. I had put one constraint to take a single array multiple times to check that it has not changed any value in the array. Once all the arrays checked successfully, the test shows the pass result. I checked the random test with all the arrays several times as I have done for each array separately. Every time it showed the pass result and I confirmed randomized flow is working fine.

Then, the normal test flow which contains IA-32 architectural instructions is intercepted with array freeze and dump randomized testing mechanism and tries to find out the bugs in the chip design before sending it to the fabrication facility. This mechanism verifies the arrays of all the clusters with various IA-32 instructions and check the information they contain. The result shows that this mechanism is very helpful

to get closer to the post-silicon debug scenario and makes it easier to add logic as per requirement.

3.6 Summary

In summary, this chapter introduces with the different scanning mechanisms of design for testing. It also explains how those mechanisms combined with randomized AFD testing mechanism at core level in pre-silicon validation and makes it easier to get closer to post-silicon debug scenario as well as issues can be found more easily in different arrays of various clusters.

Chapter 4

Performance Monitoring Multiplexer

4.1 Introduction to Performance Monitoring

The performance monitoring is a valuable for measuring performance of a program which can be analyzed to identify the bottlenecks in the program. The performance monitoring is done through the performance monitoring unit hardware which contains model specific performance monitoring counters. These on-chip counters are gaining popularity as analysis and validation tool to observe the application performance.[12]

The performance monitoring counters are broadly classified as fixed counters and programmable counters. Each thread in processor is having 3 fixed counters and 4 programmable counters. So, the availability of counters is purely depend on the number of threads working on CPU at that time.

In performance monitoring unit, the performance monitoring counters measure the events. The classification of the events can be done with two types: retirement events and non-retirement events[12]. Retirement events are counted at the retirement phase of the particular instructions life cycle. Whenever CPU encounters that the particular event should be monitored at retirement, then during the execution time of the instruction particular micro-operation is tagged and track it till the end of the instruction to check whether it is actually retired or not[12]. The other, non-retirement events are monitored at time of instruction execution.[12]

The performance monitoring capabilities are having two classes. The first class uses sampling or counting usage for monitoring performance of the events[12]. These monitoring events are non-architectural and they vary from processor to processor. They

are specific to the micro-architecture and can be changed with feature enhancements in them. The second class refers to architectural performance monitoring events[12]. When the events behave consistently across micro-architectures are called architectural events. The architectural events act consistently across the whole processor implementations.[12]

4.2 Performance Monitoring Unit Architecture

The basic performance monitoring unit architecture is shown in **Figure 4.1**. The per-

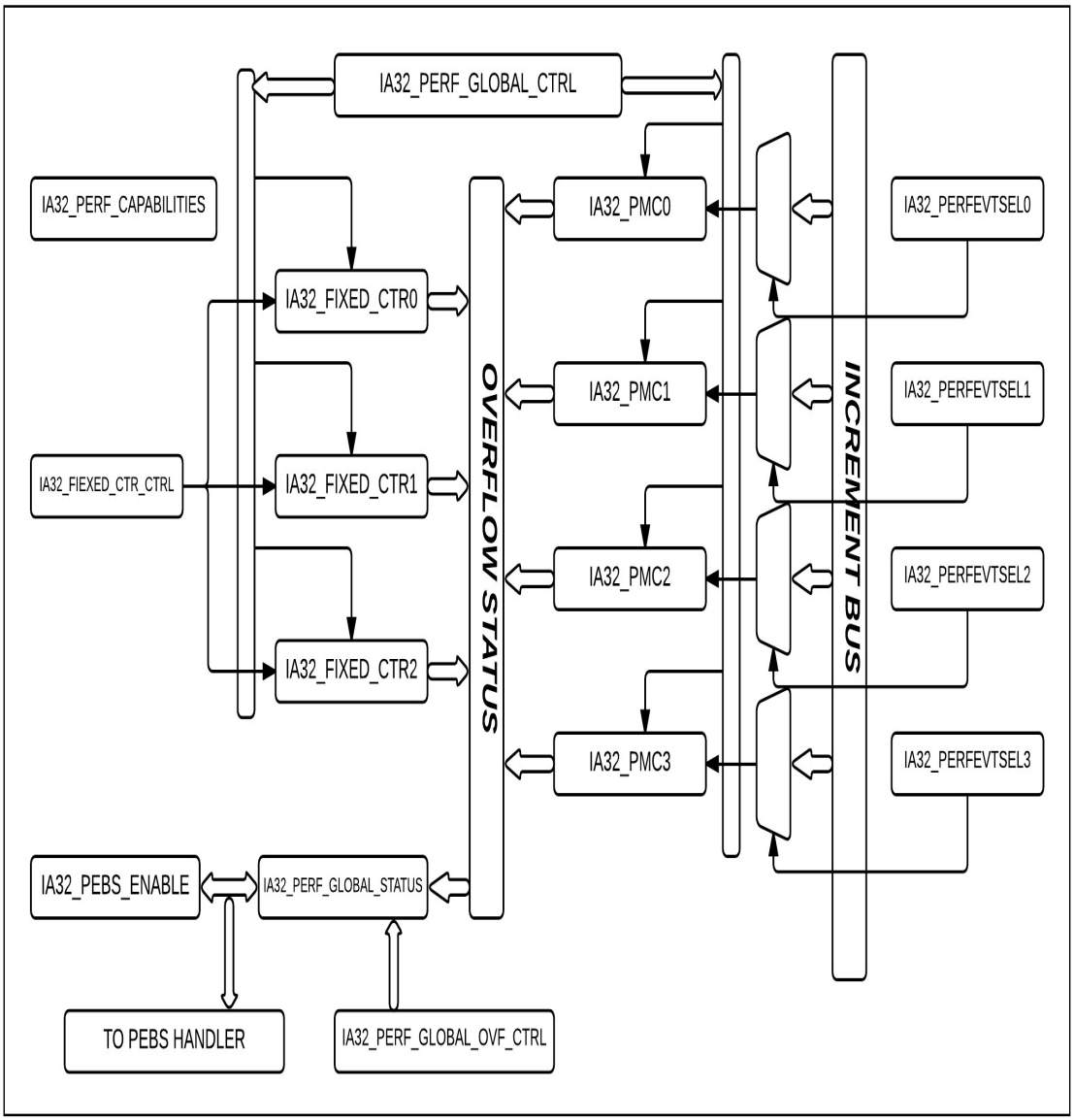


Figure 4.1: Performance Monitoring Unit Architecture

formance monitoring unit provides the ability to count the micro-architectural events which can observe internal working of the processor as it executes various codes. This unit contains various registers which guides it to monitor the events.

The performance monitoring hardware is spread across different clusters and main counters are located in the out of order cluster. The performance monitoring unit receives the event count through increment bus and different internal performance monitoring register writes.

The register IA332_PERF_CAPABILITIES defines the capability of the hardware to monitor the events. The unit is having 3 fixed and 4 programmable counters which can be handle by IA32_FIXED_CTRx and IA32_PMCx registers respectively. IA32_FIXED_CTR_CTRL controls interrupt and threading mechanisms of all the three fixed counters. To assign particular events to each of the programmable counters, IA32_PERFEVTSEL register is there. IA32_PERF_GLOBAL_CTRL is used to control all the fixed and programmable registers. There is a facility of simultaneous multi-threading in it as well. IA32_PERF_GLOBAL_STATUS is used to know status of the counter whose high bit shows the overflow condition for the respected counter. To release the overflow bit of the counters, IA32_PERF_GLOBAL_OVF_CTRL can be useful. IA32_PEBS_ENABLE is used for handling precise event based sampling. Here, both the counter types and various internal performance monitoring registers are described briefly.[12]

4.2.1 Fixed and Programmable Counters

Both, the fixed and programmable general purpose performance monitoring counters are 48 bits wide. The counter width can be enumerated using the features in CPUID instruction. In the previous processors, the performance monitoring counters were 40 bits wide.[12]

4.2.1.1 Fixed Counters

Fixed counters count architectural events that constitutes the basic events that will be used to form the basic performance metric. These IA32_FIXED_CTRx fixed general counters are controlled and monitored by registers IA32_FIXED_CTR_CTRL, IA32_PERF_GLOBAL_CTRL and IA32_PERF_GLOBAL_STATUS. The fixed counters have the capability to trigger a performance monitoring interrupt upon overflow.

Fixed Counter 0:

This counter counts the number of instructions that are retired. The term retirement represents the instructions that are already executed by the processor core and the changes will be omitted to the whole of the core including the architectural registers in the order of instruction dispatch. The MSR address of the fixed counter 0 is 0x309.[12]

Fixed Counter 1:

This counter counts unhalted core cycles when the thread is active. This counter will not increment on certain conditions like frequency switching phase of a performance state transition, HLT state of the core etc. The MSR address of the fixed counter 1 is 0x30A.[12]

Fixed Counter 2:

The reference cycles are counted by fixed counter 2. The reference cycles indicates the maximum frequency cycles at which the core can run, even though the core may be running at lower frequency. The reference clock works at particular fixed frequency irrespective of frequency changes in the core processor due to the transitions. The MSR address of the fixed counter 2 is 0x30C.[12]

4.2.1.2 Programmable Counters

There are four general purpose performance monitoring counters used to count the events programmed in IA32_PERFEVTSEL model specific register. These counters can be read or write through RDPMC instruction.

4.2.2 IA32_PERF_CAPABILITIES MSR

It holds the encoding of the available features in the performance monitoring unit implemented in the processor. The layout of the register is shown in **Figure 4.2**.

The LBR_FMT contains the information of the instruction pointer. The information, like, flags and overflow status, should be recorded in PEBS (Precise Event Based Sampling) record will be decided by PEBS_REC_FMT. The SMM_FRZ bit is set to freeze the PMU on SMM (system management mode). The set bit of PEBS_ARCH_REG suggests that PEBS record contains the architectural registers. If PEBS_TRAP is set, then the PEBS will happen before the retirement of macro instruction and PEBS RECORD will reflect the pre-retirement architectural state. If PEBS_TRAP is not

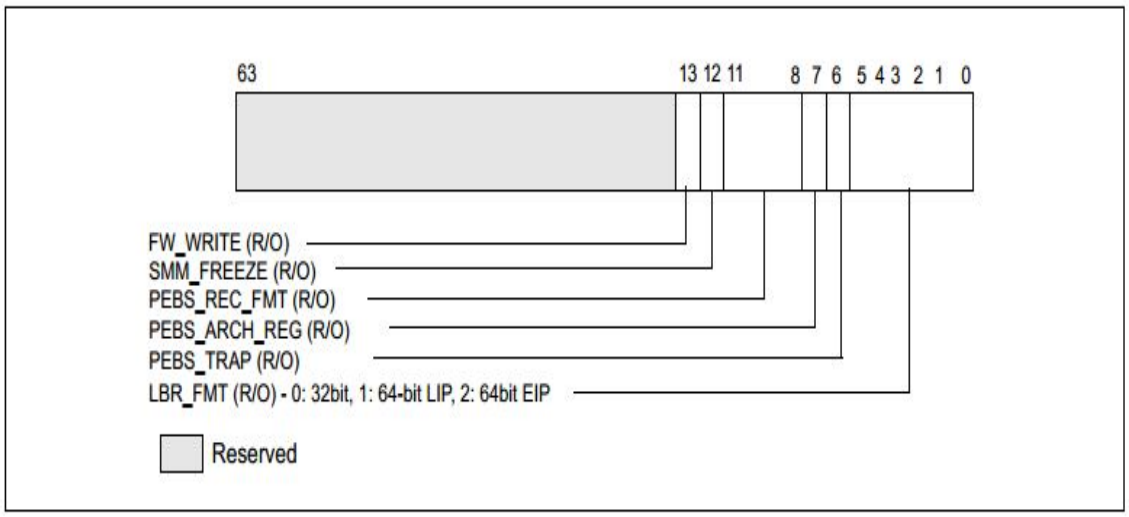


Figure 4.2: The layout of IA32_PERF_CAPABILITIES MSR[12]

set, then the PEBS will happen at the end of the instruction retirement and PEBS will reflect the machine state after the instruction retired. FW_WRITE bit specifies the architecture supports the full-width or not.[12]

4.2.3 IA32_PERF_GLOBAL_CTRL MSR

The global control register is used to operate the fixed and programmable general purpose counters. The respected bits of this register are ANDed with the local enable bit of its corresponding IA32_PERFEVTSEL or IA32_FIXED_CTR_CTRL[12]. The layout of IA32_PERF_GLOBAL_CTRL is shown in **Figure 4.3**.

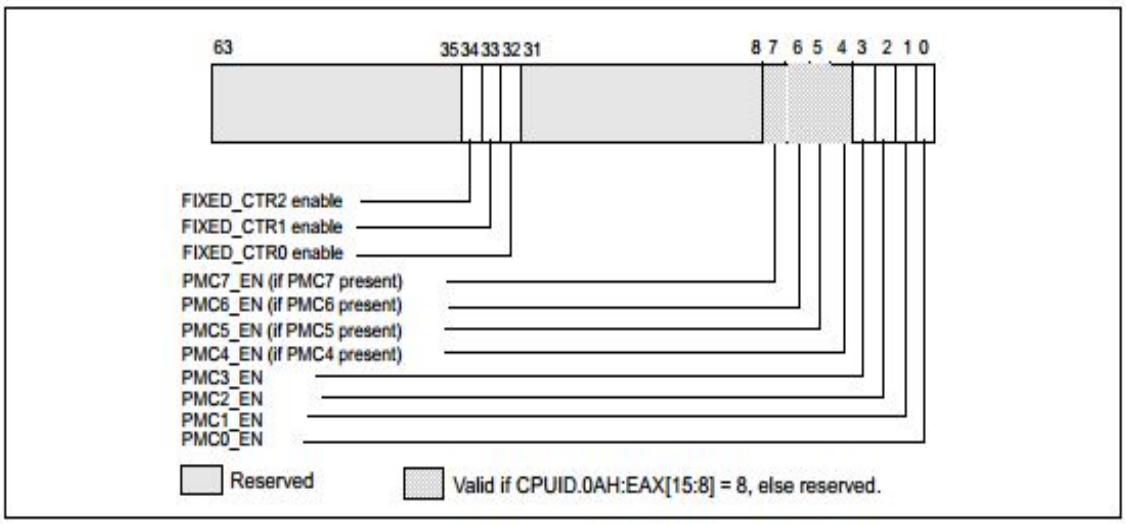


Figure 4.3: The layout of IA32_PERF_GLOBAL_CTRL MSR[12]

4.2.4 IA32_FIXED_CTR_CTRL MSR

The IA32_FIXED_CTR_CTRL MSR includes three sets of 4-bits field to control the operation of all the three fixed counters. This register controls on what privilege level the counter has to count, whether a PMI has to be serviced and whether a fixed counter work on any thread or not[12]. The layout of IA32_FIXED_CTR_CTRL MSR is shown in **Figure 4.4**.

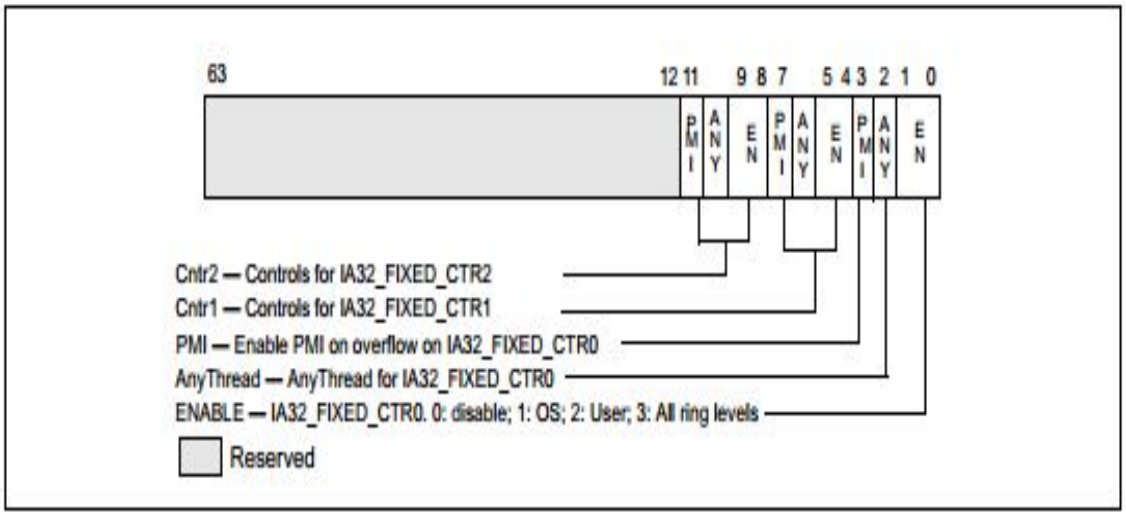


Figure 4.4: The layout of IA32_FIXED_CTR_CTRL MSR[12]

4.2.5 IA32_PERFEVTSEL MSR

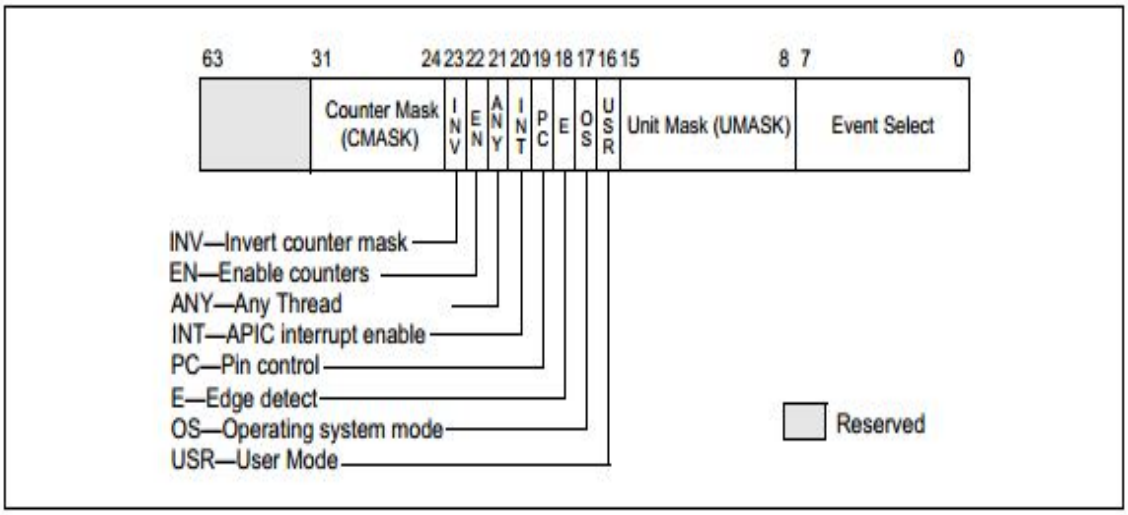


Figure 4.5: The layout of IA32_PERFEVTSEL MSR[12]

This MSR is used to assign the event, which is to be monitored by the performance monitoring programmable counters. As there are four programmable registers in the

unit, to assign the event to each of the counter, there are four performance event selection registers as well. This register holds several controls like interrupt control, thread control, pin control, edge detection and operating mode operation and user mode information. The MSR address of the IA32_PEREVTSELx is start from 186H[12]. The layout of IA32_PERFEVTSEL MSR is shown in **Figure 4.5**.

4.2.6 IA32_PERF_GLOBAL_STATUS MSR

This MSR is used to monitor the overflow status of the fixed and programmable counters and availability of the hardware through condition change bit. If the condition change bit is set then performance monitoring hardware is available to monitor the events[12]. The layout of IA32_PERF_GLOBAL_STATUS MSR is shown in **Figure 4.6**.

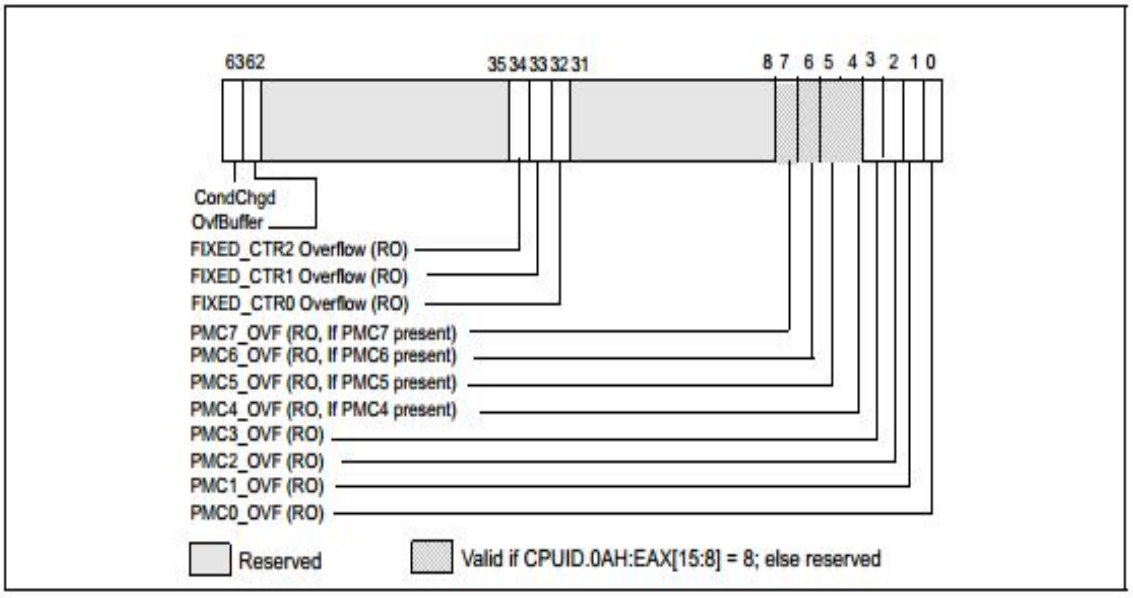


Figure 4.6: The layout of IA32_PERF_GLOBAL_STATUS MSR[12]

4.2.7 IA32_PERF_GLOBAL_OVF_CTRL MSR

Whenever the overflow bit of the IA32_PERF_GLOBAL_STATUS is high for the fixed or programmable performance monitoring counter, to release the overflow bit this MSR is used[12]. The layout of IA32_PERF_GLOBAL_OVF_CTRL MSR is shown in **Figure 4.7**.

4.2.8 IA32_PEBS_ENABLE MSR

The IA32_PEBS_ENABLE model specific register is used to enable the precise event based sampling facility for the programmable counters[12]. The layout of this MSR is

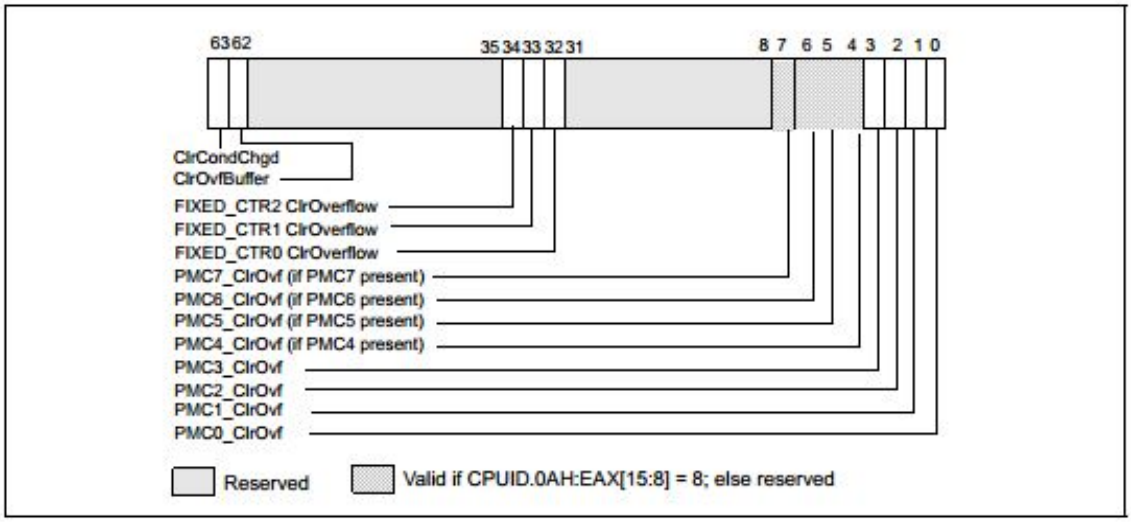


Figure 4.7: The layout of IA32_PERF_GLOBAL_OVF_CTRL MSR[12]

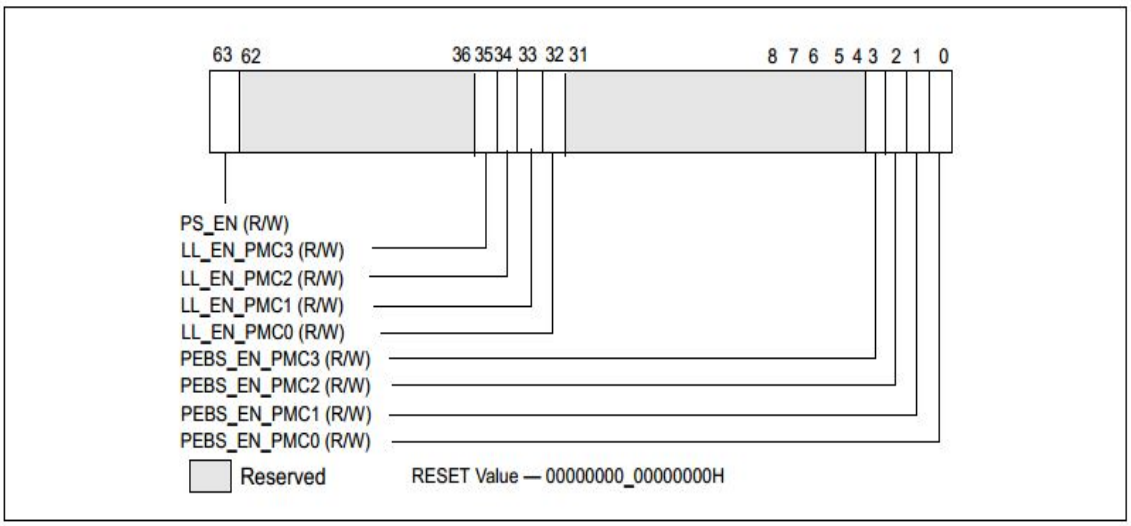


Figure 4.8: The layout of IA32_PEBS_ENABLES MSR[12]

shown in **Figure 4.8**.

The PEBS facility can be used after assigning the precise event to programmable counter through IA32_PERFEVTSEL MSR and setting up the corresponding PEBS bit in IA32_PEBS_ENABLE MSR. It allows software to profile workload behavior relative to a limited set of events. When event counter can reach overflow condition after the occurrence of a predefined number of events. On overflow of a PEBS-enabled counter, the PEBS facility is armed. At the occurrence of the next precise event (counter transition from 0 to 1), the processor will take an assist and capture machine state in a predefined memory buffer. The load latency facility allows the performance analyzing tool to calculate the latency in the load visibility in terms of clock cycles

from micro-operation dispatch to the global visibility of the loaded data. The PS_EN bit allows programmable counter 3 to store precise information.[12]

4.3 Introduction to Performance Monitoring Multiplexer

The performance monitoring mechanism can monitor the given micro-architectural events through general purpose performance monitoring programmable counters situated in performance monitoring hardware unit. Currently, the hardware unit is having only the four programmable counters to monitor the events. There are hundreds of events in various clusters to be monitored through those four counters. To validate the monitoring operation of all those events and increasing the coverage for that requires huge amount of test cases in pre-silicon validation. This scenario is impractical because it is very difficult to handle all those tests simultaneously in the environment. So, the intelligent algorithm is required to handle the validation of all the events with the same counter size and also with lesser amount of test cases to validate most of them. The block diagram of performance monitoring validation environment is shown in **Figure 4.9**.

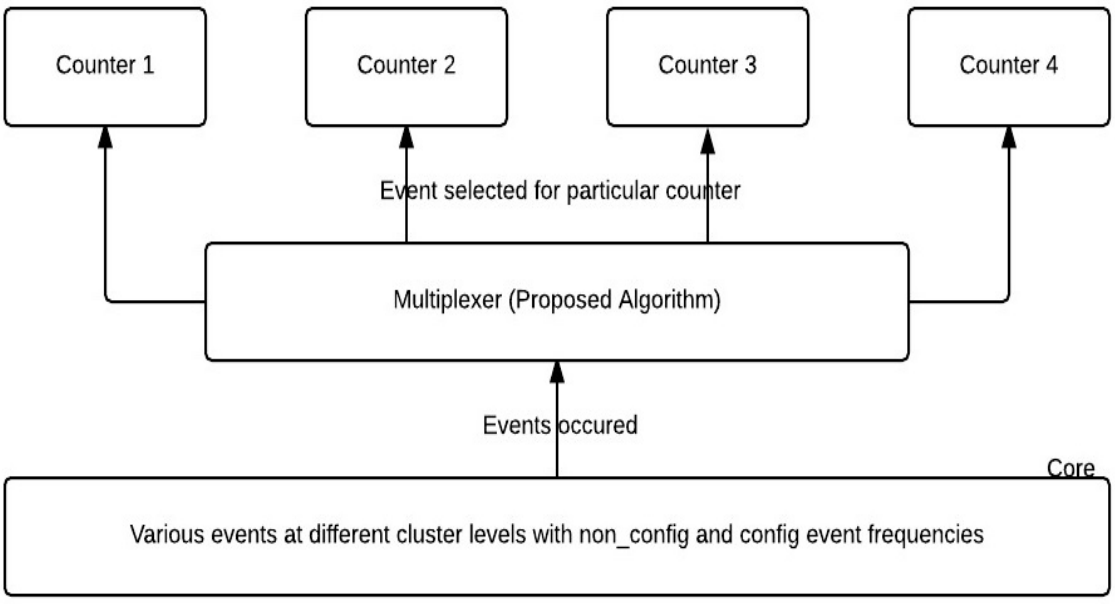


Figure 4.9: Performance Monitoring Validation Environment

In a given test, the events are having two frequencies. One is non-programmed (non-config) event frequency which suggests how many number of times the event occurred. The other is, programmed (config) event frequency which suggests how many number

of times the occurred event selected by the counter. Initially, both, the non-config and config event frequencies are at zero. When the event occurs, the count of non-config event frequency is increased. But, to monitor that event, it should be given to the counter. There is a possibility that non-config event frequency increased as far as test cases come, but, very few events get chance to be assigned to the counter. This scenario can make huge difference between the non-config event frequency and config event frequency. According to the non-config and config event frequency, we can measure non-config and config coverage of events. The past analysis without performance monitoring multiplexer shows that the non-config and config coverages were around 70% and 40% respectively. So, to improve the overall coverage by increasing config coverage, the algorithm of performance monitoring multiplexer is introduced. The flow chart of that is as shown in **Figure 4.10**. There are different test cases for various

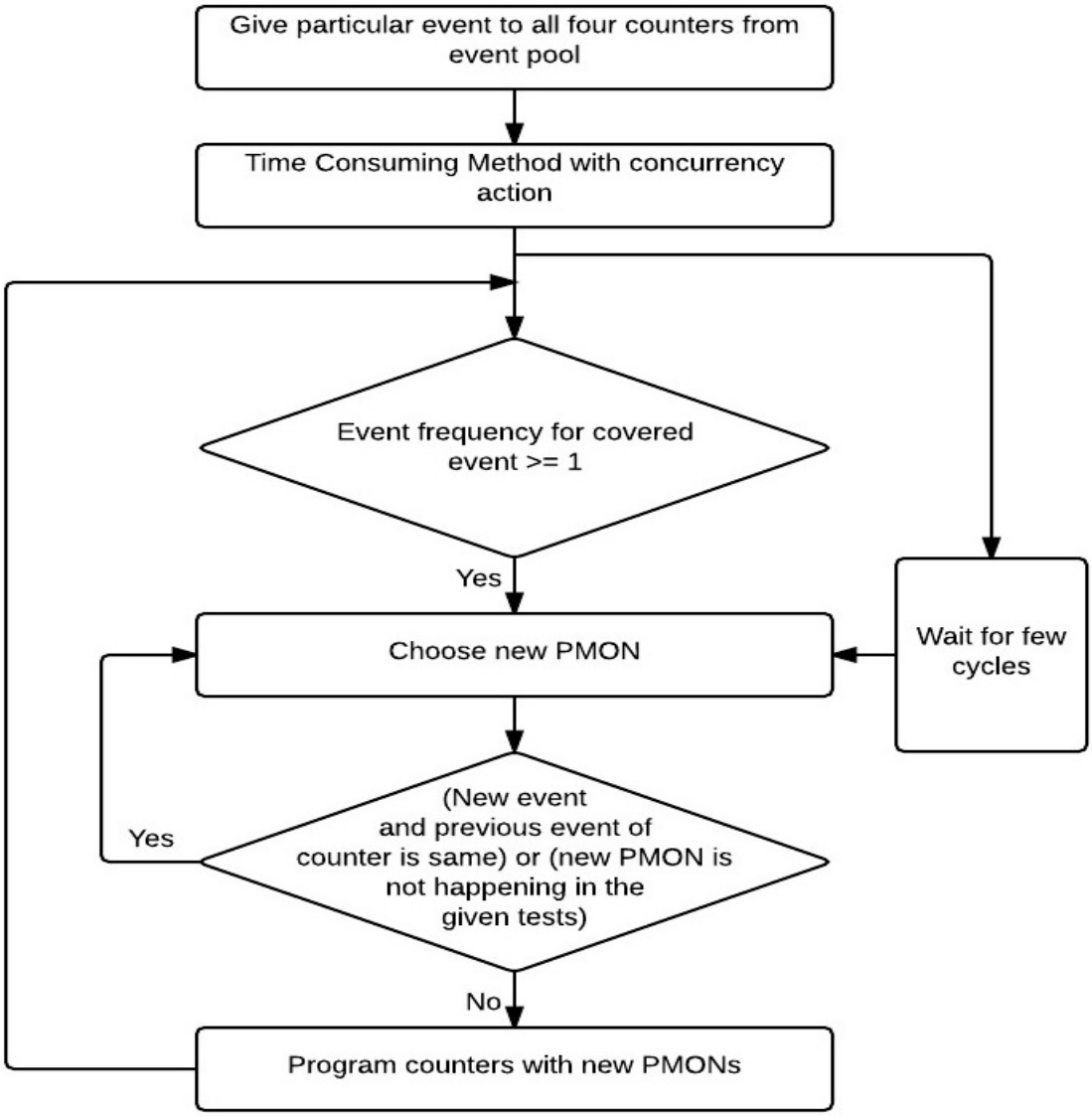


Figure 4.10: The flowchart of Performance Monitoring Multiplexer Algorithm

clusters to validate the features. Different test cases verifies different groups of events. This groups can be called as event pool. So, from those test cases events enter to the performance monitoring unit and given to the counter.

As per the flowchart explains, when the events enter into the performance unit from the event pool, particular event will be given to the all four counters. Then, it will enter into the time consuming method (TCM) with concurrency action[11] which is a concept of hardware verification language. TCM's are defined as the methods that have the notion of time, as designated by their sampling event. TCM can be subjected to synchronization or wait. Here, synchronization with two parameters will be given to the time consuming method, one is the thread and the other is the counter. So, according to the thread and counter number, the event is given to calling method. This sync TCM will check for the config event frequency to be minimum one or wait for some cycles. This methodology is used to permit the previous event, monitored by that counter, to complete with sufficient amount of time. After synchronization between both the actions, it should assign new event to the counter randomly from the event pool. The new assigned event is compared with the previous event and it should have occurred once means the new event should have non-config event frequency greater than zero. The counter should be assigned the new event till both the condition is not satisfied. After condition became true, the counter will have that event for monitoring and the config event frequency of the counter is increased. For all the test cases with given event pool, two threads and four counters the time consuming event is executed to overcome the problem of less non-config and config coverage.

4.4 Results and Analysis of different strategies used

While implementing the performance monitoring multiplexer, I have gone through several versions of the algorithm. The different strategies which I followed to increase coverage are described below. There were three test case lists, but, till the success of the algorithm, I have taken only one test case list.

In the initial version, I made the counter to check that the event frequency should be minimum one and the event should not match with the previous event monitored by that counter. After getting false result of that condition, generate the new event till condition satisfied. But, it had given 18% overall coverage.

In the second version, I have added concept of rare events on result of the first version. This was to assign the rare event field to the events which were not covered in the first

version. This addition of the rare event field to the respected event can be done by script. The rare events should be handle by three counters out of four. In this version, the coverage had improved, but, only till 25% with nominal number of events.

The result of previous case showed that the decision of assigning three counters to rare events was not correct. So, the new version was made with assigning three counters with different constraints to select events from event pool and only one counter for rare event. This time overall coverage was 36.72% including small improvement in config events with constraints of counter 1, counter 2 and counter 3 with 7, 7 and 6 respectively. The modification with different constraints to the counters were tried in multiplexer version three, but it had not made much impact.

All the three versions and addition of rare event field had not made improvement in coverage as expected. The main reason behind the failure is that the counter, assigned to the rare events, is not giving sufficient amount of time to the event to be monitored by that counter. So, the concepts of rare event field and constraints to counters were dropped.

Then, in version four, time consuming method with concurrency action is introduced. In this version, the algorithm is same as in first version, but, the only addition is concurrency. It helped to get 45.7% overall coverage with good amount of config events.

The new concept of two separate event frequencies, non-config and config event frequency, was introduced. Based on config event frequency the time consuming method try to make synchronization and after that, on the minimum value of non-config event frequency as one, the new event should be taken or the counter should be given to monitor that event. This helped to get overall coverage of 36.47% which was lesser than the previous case, but, the amount of config events covered was increased. Ultimately, the main concentration is to increase the coverage for config events which was achieved by this version. It suggests that the direction of the algorithm is perfect. The same version with some what modification of the environment was applied on different test case lists. Different test case lists were taken because at the same time the test cases should be minimal as well with high coverage. The results of different test lists, test list 1, test list 2 and test list 3 got coverage of 56.47%, 68.23% and 73.73% as shown in **Table 4.1** and **Figure 4.11**.

Table 4.1: Comparison of Coverage Results of Three Different Test Lists

Test List Name	**Test List 1**	**Test List 2**	**Test List 3**
Total Tests	114	100	3091
Non-Config Coverage	86.96%	88.43%	92.18%
Config Coverage	25.98%	48.03%	55.28%
Overall Coverage	**56.47%**	**68.23%**	**73.73%**

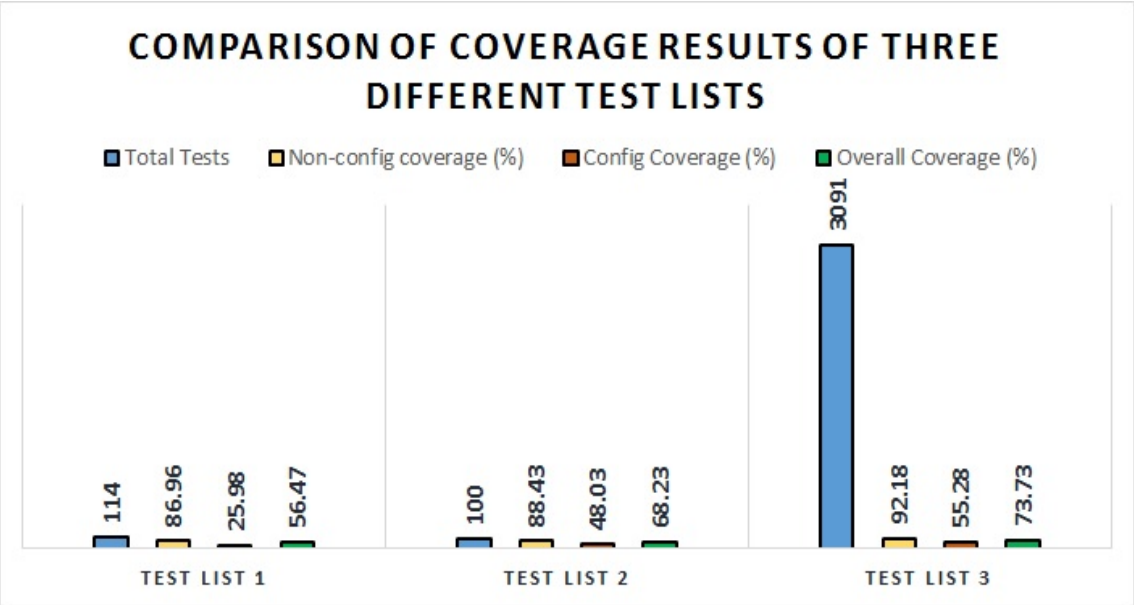


Figure 4.11: Comparison of Coverage Results of Three Different Test Lists

At last, the events which are having on reference signals were removed. This scenario helped a lot to get coverage of 62.65% and 81.01% for test list 1 and test list 2 respectively. As test list 3 contains much more number of the tests, so the coverage measurement was not taken for that list. The full detail of test list 1 and test list 2 is as shown in **Table 4.2** and **Figure 4.12**.

Table 4.2: Final Coverage Results of Two Different Test Lists

Test List Name	**Test List 1**	**Test List 2**
Total Tests	114	100
Non-Config Coverage	80.03%	82.24%
Config Coverage	45.29%	79.77%
Overall Coverage	**62.65%**	**81.01%**

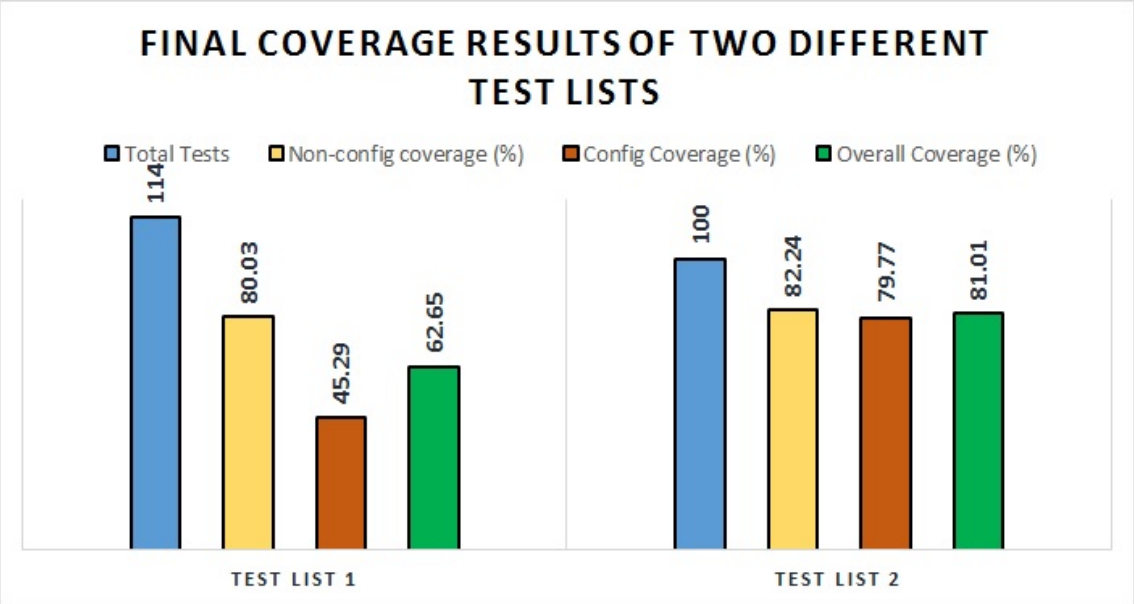


Figure 4.12: Final Coverage Results of Two Different Test Lists

The result of **Table 4.2** and in **Figure 4.12** shows that the test list 2 is the best suited test case list. Therefore, with use of the performance monitoring multiplexer, the overall coverage can be increased with increasing non-config and config coverage as shown in **Figure 4.13**. Here, the version 5 was taken with old environment to compare the actual values.

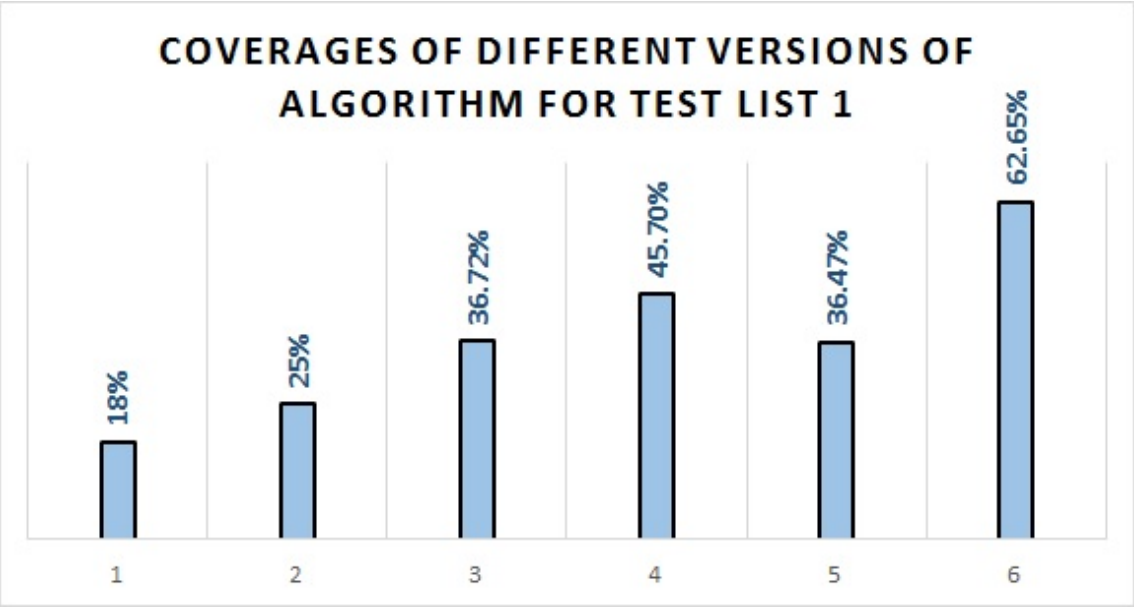


Figure 4.13: Coverages of Different Versions of Algorithm For Test List 1

4.5 Summary

In summary, the chapter explains performance monitoring unit and various model specific registers. The later part introduces with the performance monitoring multiplexer algorithm which helps to increase the overall coverage to 81.01% in pre-silicon validation with less number of test cases.

Chapter 5

Automated Validation of SMM

5.1 Different IA-32 Architecture Modes

IA-32 architecture contains various operating modes like virtual-8086 mode, protected mode, real-address mode and system management mode.[12][13]

The Real-address operating mode provides the programming environment of the Intel 8086 processor architecture with ability to switch to protected mode or system management mode. After hard reset happens, the logical processor always starts operation in real-address mode.[12][13]

The Protected mode is also one of the important operating mode of the processor. It provides a high performance, rich set of architectural features, flexibility and backward compatibility to existing software base. It implements an environment wherein the operating system task scheduler permits the processor to execute a particular task for a given period of time while all the other tasks are suspended temporarily. It prevents unauthorized access to operating system services and resources.[12][13]

The System management mode is a architectural feature of the processors which handles system design specific events like platform-specific power management or thermal events. When the processor encounters a system management interrupt to the chip set, it enters to the system management mode. It automatically saves the contents of processor's register set when the interrupt occurs and restores the state of register after the execution of the system management interrupt.[12][13]

Virtual-8086 mode makes the processor to feel like it is running in real mode. Due to disruptive behavior in a multi-tasking operating system environment of protected mode, the task scheduler switches logical processor into virtual mode to resume the

execution of real mode. In this mode, the processor activates hardware which observes the sensitive instruction in manner that it cannot disrupt the operation of the overall software environment.[12][13]

The switching of mode happens when specific bit set which affects that mode. For system management mode, system management interrupt forces the processor to jump into system management mode.

5.2 Importance of System Management Mode

The SMI (System Management Interrupt) is the higher than the non-maskable interrupts as well as normal interrupts of the processor. Therefore, system management mode code is having unique position in the processor to monitor the software running on top of it, whether it is an operating system or a virtual machine monitors. The SMM also assists to hardware manufacturers for debugging. The system management mode is used for several activities.[13]

It handles the system events like memory or chip set errors. It manages the safety function of the system like shutting down the high CPU temperature with turning on the fans[12]. It supports security functions and power management. It emulates hardware of the motherboard which is unimplemented, mouse and keyboard. It also used to centralize the system configuration, for example, Toshiba and IBM notebook computers. There are still many more advantages of system management mode.[13]

As the system management mode carries so many uses, it is necessary to validate the system management mode feature more accurately at pre-silicon validation. Therefore, if there is an availability of efficient validation technique for system management mode, it is a great time saving effort. For making the efficient validation of feature, thorough understanding of the feature is required.

5.3 Switching between SMM and other processor operating modes

The **Figure 5.1** explains the transition of the operation of different modes in the processor. For system management mode, the processor jumps to system management mode whenever it receives system management interrupt and the current mode is either virtual-8086, protected mode or real-address mode. After execution of Return form system management mode (RSM) instruction in system management mode, the processor returns to the same state where system management interrupt occurred.

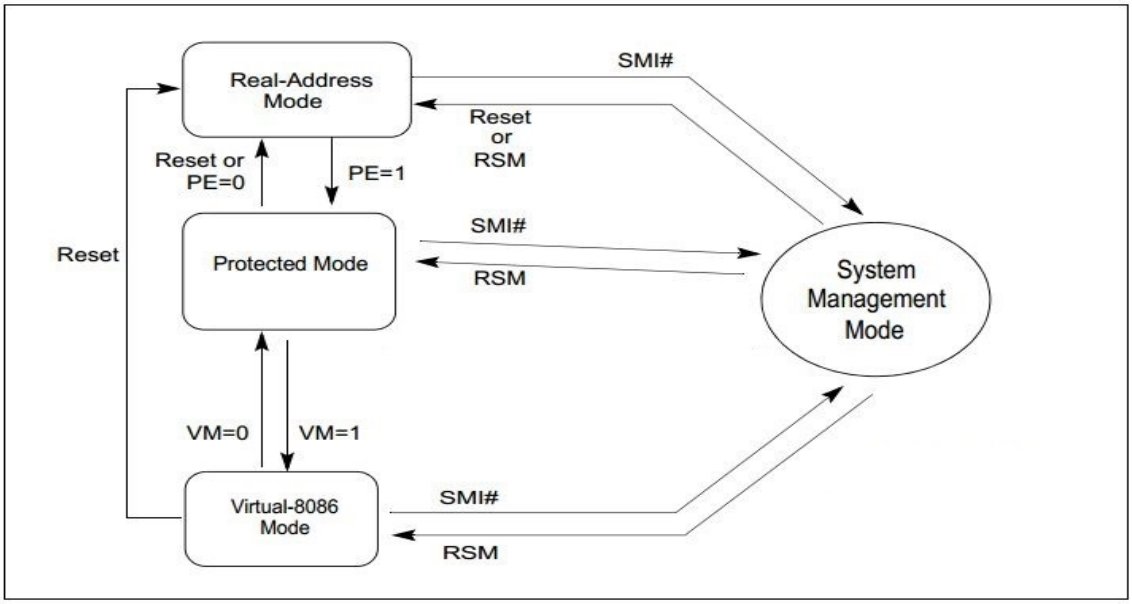


Figure 5.1: Transition Among IA-32 Architecture Modes[12]

5.3.1 Entering in SMM

The processor handles a system management interrupt on an IA-32 architecture instruction boundary, defined as interruptible point in the program execution. When the processor receives an SMI, it waits for ongoing instruction operation to complete and stores the state information of that time. The SMRAM saves all the current values of the processor. Then, the processor enters into system management mode and starts execution of the SMI. The processor also signals the processor's external hardware that the handling of system management interrupt has started. This kind of signaling mechanism is dependent on hardware implementation. The system management mode cannot acknowledge the sequences of system management interrupt.[13]

5.3.2 Exiting from SMM

After entering into system management mode, the processor can exit from it with only one instruction i.e. RSM. The RSM instruction cannot be executed in any other mode than SMM, otherwise an invalid-opcode exception is generated and the processor stops executing. The RSM instruction is used to restore the processor's context to the registers from SMRAM. Then, the interrupted program is called back through SMIACK. Again, the signal is sent to the external hardware for successful completion of SMI. If the SMRAM holds the invalid state information, the processor enters into the shutdown state and indicate that it has entered into shutdown state through generation of a special bus cycle.[13]

5.4 Automated Validation of SMM

The **Figure 5.2** shows the automated validation strategy of the system management mode.

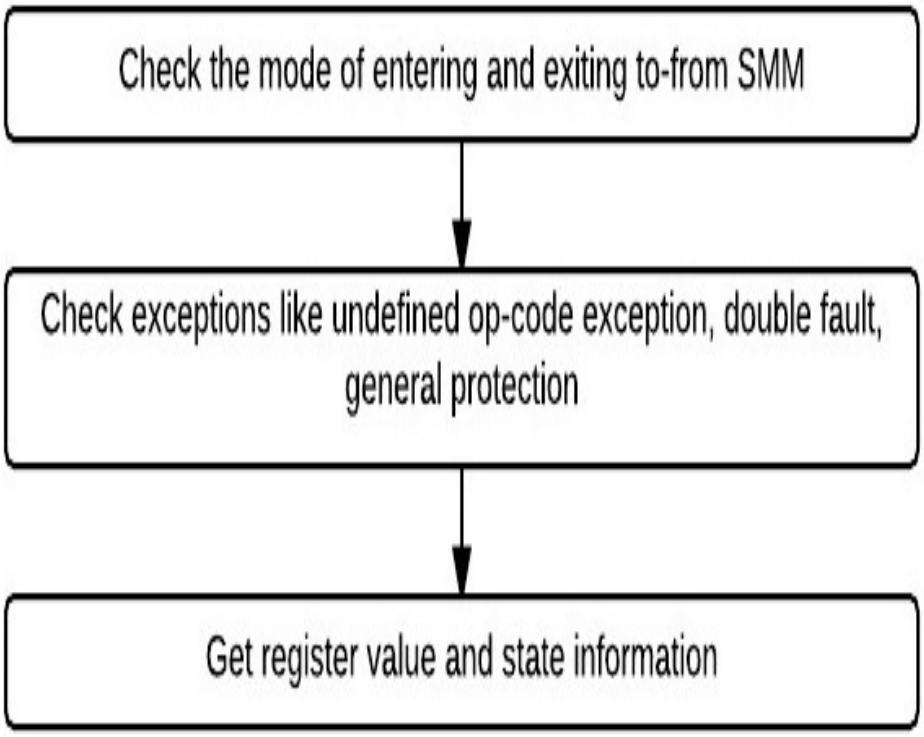


Figure 5.2: Automated Validation of SMM

All the described steps in previous section should be followed in that particular manner to run the processor successfully in system management mode. If any of the stage is not behaving properly, then there is a possibility of different exceptions like undefined opcode exception, double fault, page fault, general protection exception, etc. Those

errors can be checked by self-check mechanism. To validate them with automated strategy, one can make such a script which can help to debug the different errors. In that script, the implementation of the intelligent algorithm should be done which can fetch the different necessary register information, state information at the time of changing the mode to SMM and each of the instruction with their cycle timing from different logger files, so that it can help to find the exception and failing point very easily. This kind of automated script can save a lot of time of a person by not going into different logger files deeply each and every time.

5.5 Summary

In summary, this chapter includes description of different modes and their switching to system management mode. The later part shows the importance of SMM with its flow after SMI generation and proposes the automation of the SMM validation to save time by not doing debugging manually.

Chapter 6

Automated Group Routing Strategy

6.1 Introduction to Routing

The physical design flow process is divided into two major parts, a placement phase followed by a routing phase. Both the phases are much important in design of the chip because they deal with timing requirement and available area resources of the chip. During placement, the different components are placed onto substrate surface so as to minimize cost criteria such as the total estimated wire length. The routing phase is the one that determines the actual course of the wires connecting the cells that have been placed[15]. The routing has to obey various design rules like width of wires and wire crossing that ensure a technically functioning circuit. The selection of router type and the prioritizing the layout of critical signals such as clock signals. The structure of the routing phase is influenced by the design and fabrication technology. The basic terminologies of the routing are as below:

Nets: Set of two or more pins having same electrical potential
Netlist: Set of all the nets
Via: Connection of two different metal layers
Congestion: Where the tracks of metal layers are not sufficient enough for routing

The following section describes different routing types, hierarchy of the design and, at last, automated group routing strategy.

6.2 Types of Routing

The routing is typically complex combinatorial problem. The routing may need to route tens of thousands of nets simultaneously without overlapping. To make it manageable, the routing is usually done by two-stage approach of global routing followed by detailed routing[15]. The global routing first partitions the routing region into tiles and decides tile-to-tile paths for all nets, whereas detailed routing determines the exact tracks and vias for nets.[16]

6.2.1 Global Routing

The main aim of the global routing is to provide a routing plan in which each net is assigned to a particular routing region. In global routing, the minimal usage of the total wire length, priority of the signals and balance of the congestion across the routing region should be taken in consideration[16]. It plays an important role in obtaining a good overall layout.

Based on the timing constraints and criticality of the net, the routing resources are assigned from one block of the chip to the another block of chip based on the defined connectivity in the netlist. There is no end-to-end connection happened in global routing. But, it decreases the open length between two pins using efficient routing resources. The global routing generally uses the higher metal layers for routing so that good timing can be achieved.

6.2.2 Detailed Routing

The main aim of the detailed routing is to facilitate the precise route to every net within their assigned routing regions. The detailed routing does the end-point pin to pin routing and close the opens of the chip. Most of the time, lower level metal layers are used for detailed routing than the global routing. The downgrading of the nets are done inside the block and connection is done with pin through appropriate metal.[16]

The detailed routing should take care of layout versus schematic issues of pins, valid antenna length and different design rule check like via-to-via, min-jog, via size etc. The detailed routing should be done such as to meet timing constraints of the design chip.

There can be two different models for routing: grid-based model, position of grids vary from one metal layer to the other and grid-less model, totally randomness in placement of the metal layers.

6.3 Hierarchy of the design

As described in **Section 2.3.1**, partitioning enables structural implementation of big complex system into number of small blocks by using divide and conquer approach. This methodology can localize the modifications and reduce the complexity.

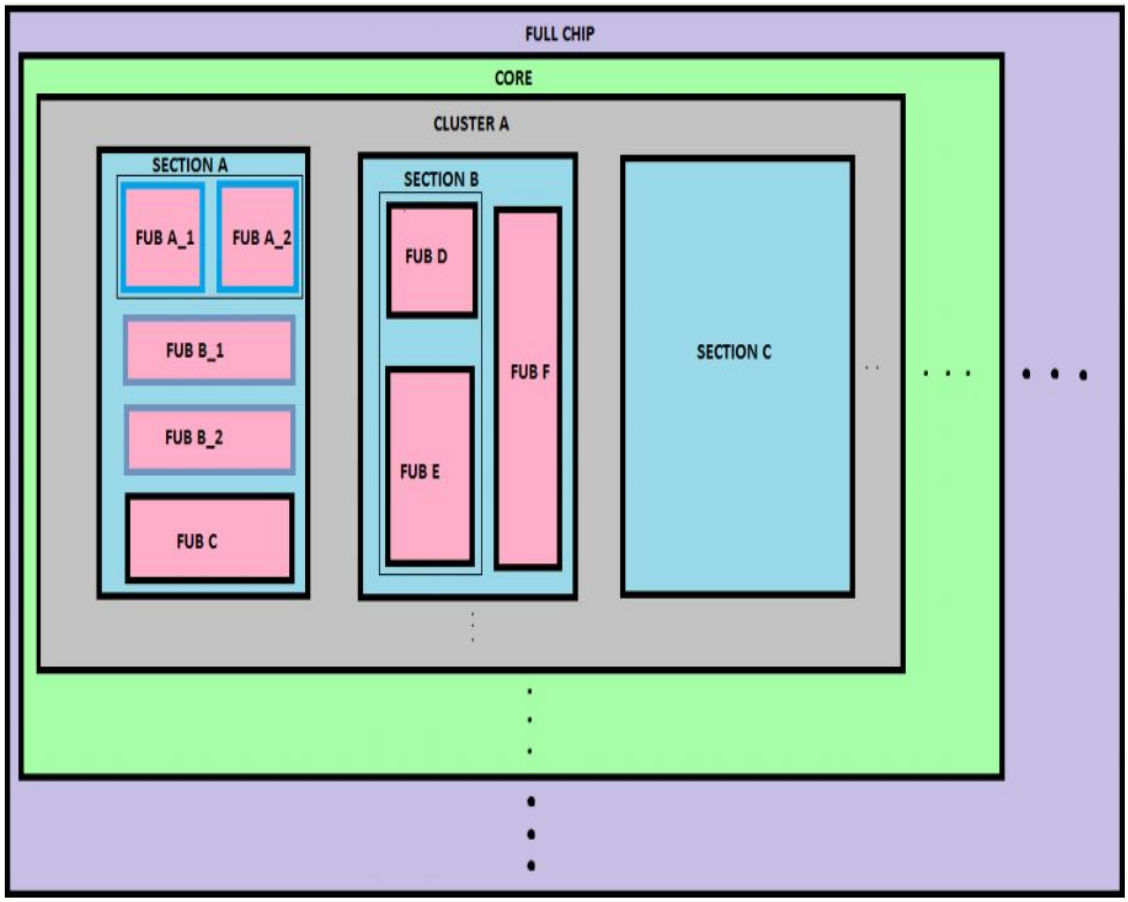


Figure 6.1: Hierarchy of the design

A commonly used hierarchical partitioning model is shown in the **Figure 6.1**. The lowest design hierarchy level is functional unit block(FUB). A functional unit block implements the basic functional designs such as adder, multiplier, divider, etc. A number of functional unit blocks constitute a section. The number of functional unit blocks are grouped based on their characteristics and physical location in layout. The functional unit blocks are considered to be a black box at the section level and only shared attributes are visible at section level. A cluster is made up of different sections. The group of cluster constitute the full chip. Such methodology enable the designer to debug design issues very efficiently and effectively with less amount of turn around time.

6.4 Needs of Automated Routing

As the chip area advances to nanometer technology, the complexity of the interconnect design also increases with great extent. The floorplan decides how much routing resources you can have for efficient connection between logical cells. According to criticality of the nets and timing requirement of those nets decide the metal allocation for them. But, this strategy takes lot of iterations and huge manual effort. If some logical change occur in middle of the layout implementation, then the new logic cells will be implemented or removed. It results in repetitive work of layout with removing old routing and implementing the new routing which is time consuming and tedious task. To overcome this cumbersome task, there is a requirement of the automated routing strategy which can do routing with efficient use of available routing resources. As newer technologies emerge, either existing routing algorithms should be able to cope with the new routing constraints, or else new algorithms should be developed. The following section suggest one of the automated routing strategy i.e. group routing strategy which saves lot of manual effort and time with better control on routing resources.

6.5 Automated Group Routing Algorithm

The proposed automated group routing strategy chooses the routing resources based on the priorities and the physical properties like routing length, fanout etc. The group routing algorithm is proposed to automate the routing at section level. There are majorly three steps in automated group routing: distribute all the nets of the section into groups based on the physical criteria, prioritize the various groups and route the nets of all the groups in similar manner.

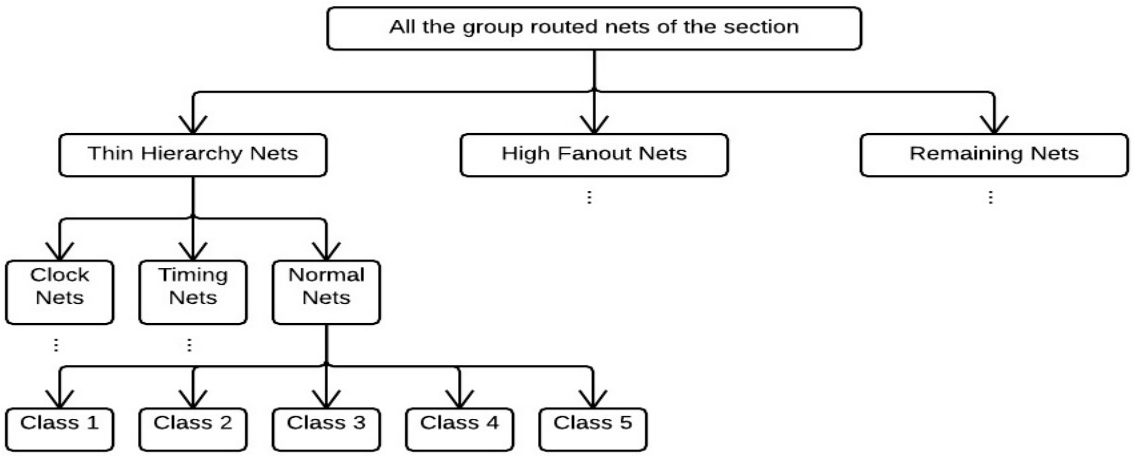


Figure 6.2: Distribution of Nets for Group Routing Strategy

Figure 6.2 shows distribution of the nets to be routed through group routing. This distribution of the nets is prioritized in order of thin hierarchical nets, high fanout nets and remaining nets. All the groups are further divided into priority of clock nets, timing critical nets and normal nets. After distribution of the nets, there are sub-divisions as per the routing length of the nets. As per the length of routing, different class groups with combination of two metal layers are assigned to nets for the routing. If the routing length is too much then higher metal layer class should be used. The class 1 uses the higher metal layer as it contains the group of nets which is having highest routing length. Then, depending on predefined routing length threshold, all other class from 2 to 5 assigns metal layers in decreasing order. The algorithm for group routing strategy is as below.

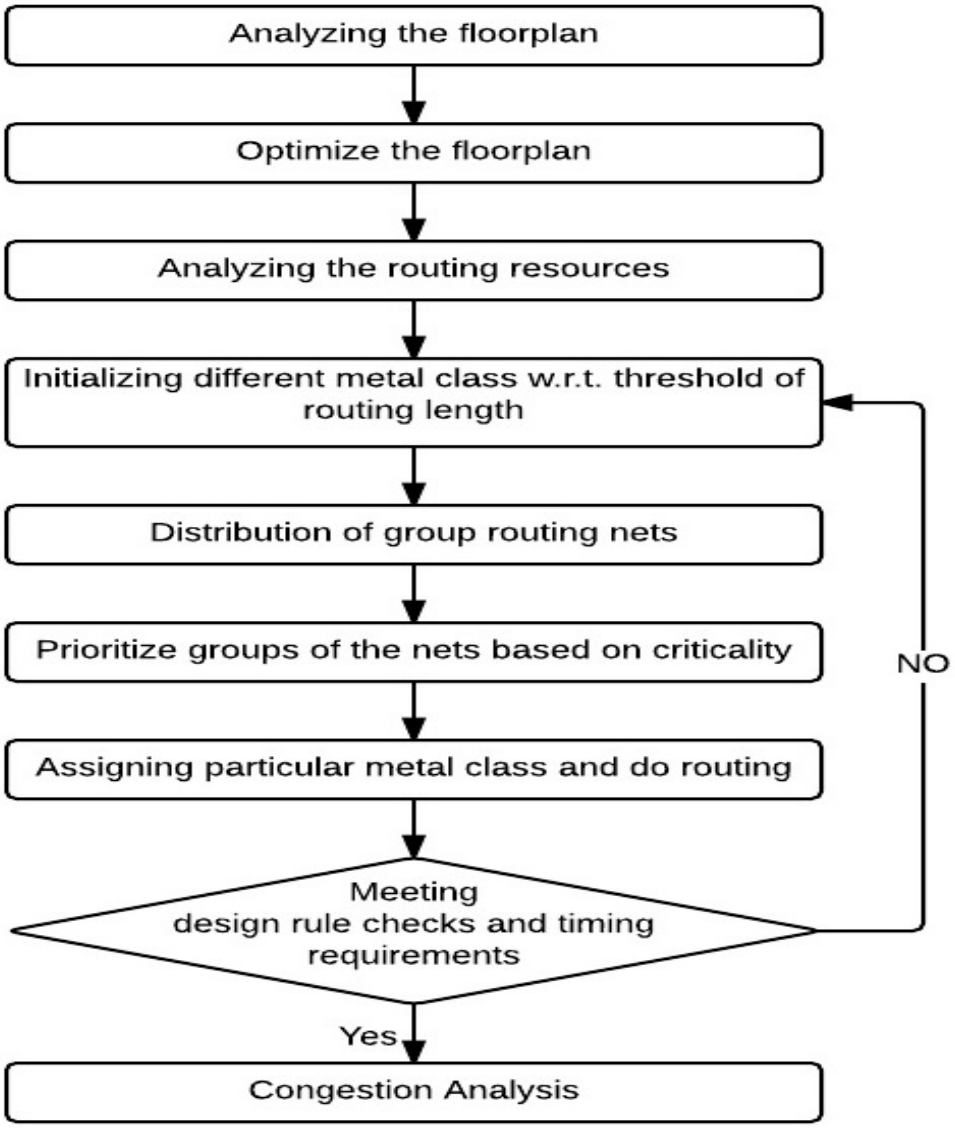


Figure 6.3: Automated Group Routing Strategy

As shown in **Figure 6.3**, the automated group routing strategy starts with analyzing and optimizing the floorplanning which helps to change block placement according to logic and timing criticality of the net. From that the design engineer can have idea about the need of routing resources for various nets. Then, initialization of different metal layer classes will be done with respect to the threshold of the routing length. After that, the distribution of nets are done according to priority and assignment of the metal classes as shown in the **Figure 6.2**. At last, routing is done through existing automated routing strategy.

After the successful routing of the group of nets, we need to check the various design rule checks described in **Section 2.3.2**. We also need to check timing requirements of the logical cells are met or not with the group routed nets. From both these steps, we can analyze the efficiency of the automated group routing and if it does not match the requirement, run the automated group route again after changing the metal allocation of classes and threshold of routing length. Though fully automated routing solution never matches up to the quality of the manual routing, still it helps to get rid of repetitive work and manual effort.

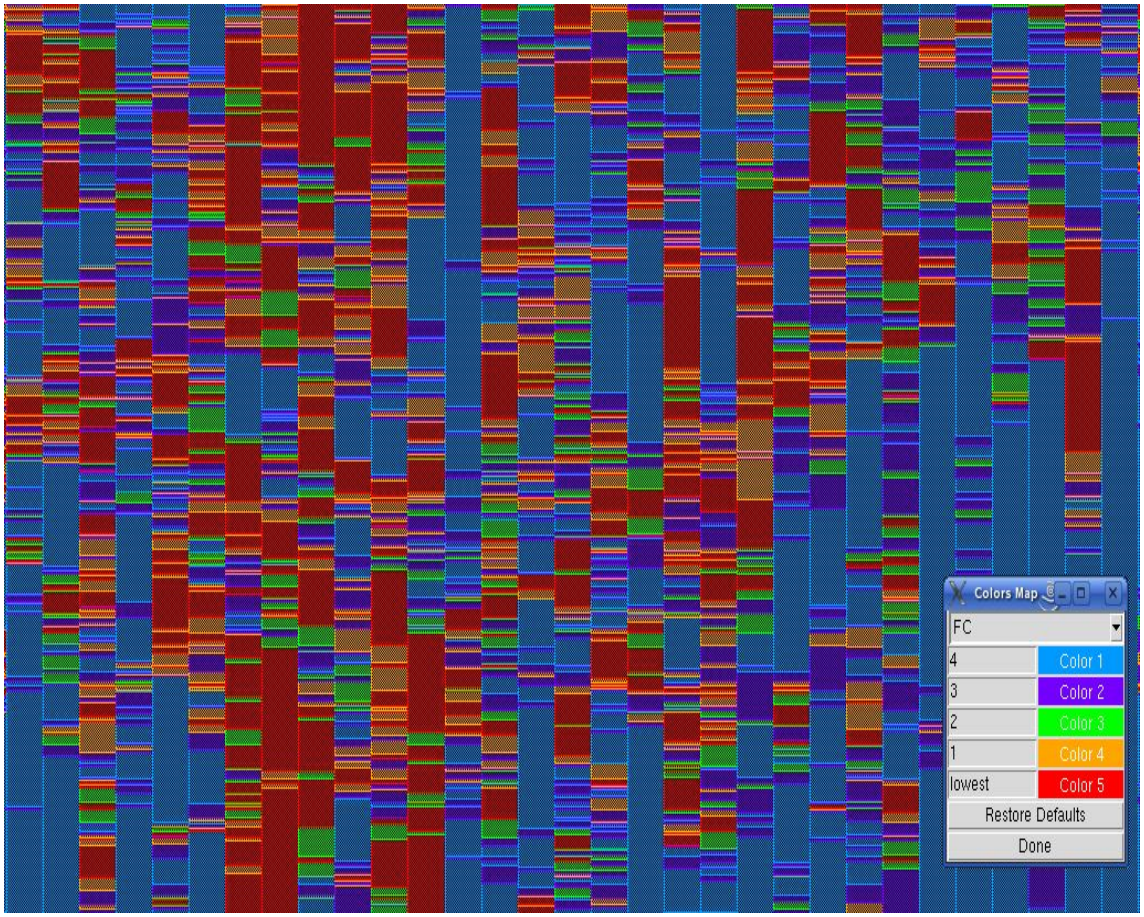


Figure 6.4: Congestion Analysis

The **Figure 6.4** of the congestion analysis of the automated group routing shows the congestion of one of the metal layer with different color styles dependent on the routing track availability for that metal layer in that particular region. for example, green color shows two tracks are available for metal x routing in that particular region. The early routing strategy was taking 4x amount of time due to manual routing efforts. The new automated group routing strategy has reduced routing time by 2x.

6.6 Summary

In summary, this chapter includes two different routing strategy and hierarchy of the design. Later part explains the need of automated routing and how automated group routing strategy can help to resolve the problem to some extent. This strategy encourage to observe different floorplan strategy and reduces manual effort of routing by 2x amount of time.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

As described in introduction part of the thesis, the high-coverage and high-efficiency is recommended in pre-silicon validation and layout design respectively.This can be achieved by introducing the new strategies to the validation of the features and different automated routing strategies. The result and analysis sections of the different chapter in the thesis shows that the efficiency in AFD and coverage in PMON are increased by AFD randomization and PMON multiplexer respectively.

The randomized AFD testing mechanism compares the RTL values of all the arrays with its expected values at the freeze time in the single test. As it is done at super cluster level, it can help one to reach closer to the post-silicon debug scenario. It can make post-silicon debug easier than earlier and allows to put necessary logic at required places with less effort.

The performance monitoring multiplexer increases the coverage for monitoring the various events in pre-silicon validation. It is also conforming that the number of test cases require to validate the various events are much lesser than the number of events which we need to validate. For pre-silicon validation, the performance monitoring multiplexer has increased the non-config and config coverage from the 70% and 40% to 82.60% and 79.77% respectively. The final overall coverage of 81.01% is achieved through performance monitoring multiplexer.

After that, the importance of the system management mode is discussed with its operation with different other modes. That has suggested the automation of the debugging for system management mode which can help to increase the overall efficiency of system management mode in architecture validation.

The earlier routing strategy takes 4x amount of time due to manual efforts. The automated group routing strategy saves half of the time by completing routing in 2x time. It also targets total manual effort through automatic selection and management of critical nets, faster timing convergence, while ensuring quality layout, considerable reduction in turnaround time and encouraging for different floorplan possibilities.

7.2 Future Scope

The thesis has discussed various strategies to increase the efficiency of different features in pre-silicon validation and back-end layout design. In the thesis, the coverage improvement of performance monitoring was done at a cluster level. The same algorithm can be implemented at super cluster level with changing the necessary fields in code written in hardware verification language. As per the suggested strategy to improve debugging for system management mode, the script can be developed for that. There are many other features for which we can find out the root cause of less efficiency and try to make the pre-silicon validation more efficient.

In back-end layout, the group routing strategy can be improved through concentrating on different physical aspects which ultimately helps to route more number of the nets automatically. The merging of the design rule checks can also be done to reduce the manual efforts of design engineer.

Bibliography

[1] Ilya Wagner, Valeria Bertacco, Post-Silicon and Runtime Verification for Modern Processor, Springer Publication, 2011. ISBN:9781441980335.

[2] Specman in one day. URL: http://www.asic-world.com/specman/specman_one_day4.html (Visited on 04/18/2015).

[3] ASIC Verification. URL: http://chipverification.blogspot.in/2008/03/introduction-to-specman.html (Visited on 04/18/2015).

[4] FDIV Replacement Program, Statistical Analysis of Floating Point Flaw:Intel White Paper- Section 3. URL: http://www.intel.com/support/processors/pentium/sb/CS-013007.htm (Visited on 04/21/2015).

[5] Thomas R. Nicely. Pentium FDIV flaw. URL: http://www.trnicely.net/pentbug/pentbug.html (Visited on 04/20/2015).

[6] Scott Wasson. Phenom TLB patch benchmarked-A look at how AMD's BIOS workaround impacts Phenom performance. URL: http://techreport.com/review/13741/phenom-tlb-patch-benchmarked (Visited on 04/20/2015).

[7] Physical Design, URL: http://en.wikipedia.org/wiki/Physical_design_(electronics) (Visited on 04/26/2015).

[8] Sini Mukundan, Physical Design Flow IV:Routing. URL:http://vlsi.pro/physical-design-flow-iv-routing(Visited on 04/25/2015).

[9] Grant MacFarland. Microprocessor Design-A practical guide from design planning to manufacturing, Tata McGraw-Hill, 2006. ISBN:0070619298.

[10] Adrian Carbine, Derek Feltham, "Pentium Pro Processor Design for Test and Debug", IEEE Design & Test of Computers, Vol. 15, No. 3, Jul-Sep 1998, pp. 77-82.

[11] Specman e Language Reference Manual, Verisity Design,2002.

[12] Intel 64 and IA-32 Architecture Software Developer's Manual, June 2014.

[13] Tom Shanley, x86 Instruction Set Architecture Comprehensive 32 and 64 bit Coverage, Mindshare Inc., 2009. ISBN: 0977087853.

[14] Delagado B., Karavanic K.L., "Performance Implication of System Management Mode", IEEE International Symposium Workload Characterization (IISWC), Sept. 2013, pp 163-173.

[15] R. Venkateswaran, P. Mazumder, "Routing Algorithms for VLSI Design", Department of Electrical Engineering and Computer Science, The University of Michigan, USA.

[16] Laung-Terng Wang, Yao-Wen Chang, Kwang-Ting (Tim) Cheng. "Electronic Design Automation: Synthesis, Verification, and Test", Elsevier Inc., 2009. ISBN: 978-0-12-374364-0.