Pre and Post Silicon Validation of SoC

A DISSERTATION

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

MASTER OF TECHNOLOGY
IN
VLSI DESIGN & EMBEDDED SYSTEMS

Submitted by:

NEELAM

KUMARI

2K18/VLS/07

Under the supervision of

Dr. Neeta Pandey
Professor

ELECTRONICS & COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042
2018-2020

ELECTRONICS &COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

CANDIDATE'S DECLARATION

I, (NEELAM KUMARI), Roll No. 2K18/VLS/07 student of M.Tech (VLSI & Embedded systems), hereby declare that the project Dissertation titled "Pre and Post Silicon Validation of SoC" which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi(Neelam Kumari)

Date:

ELECTRONICS & COMMUNICATION

ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of

Engineering) Bawana Road,

Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "Pre and Post Silicon Validation of SoC" which is submitted by Neelam Kumari 2k18/vls/07, Electronics & Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi(Dr. Neeta Pandey)

Date: Professor

ACKNOWLDEMENTS

Every endeavor needs deft guidance and helping hands for its accomplishment, and I was fortunate to have both. I wish to express my indebtedness to my mentors, Dr. Neeta Pandey(Professor) and Mr. Rajeev Srivastava, without whose guidance, this work would not be possible, for their kind guidance, motivation and for providing unconstrained freedom to explore. I wish to express my gratitude to my mentor and manager at Qualcomm(Noida), Mr. Anup, who was always lucid in his explanations, dexterous in his guidance, and generous with his care and support.

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

Finally, I would like to thank my parents, family, and friends, for

bestowing their love, patience, understanding, and care, which kept me going.

Date:Neelam Kumari

ABSTRACT

Semiconductor product design”is a central processing unit (CPU) core”based SoC in which additional functionality”is provided by intellectual property (IP) cores connected to the CPU core. In every new generation of the SoC, more functionalities are added. In other words, it has a higher number of IPs compared to the previous one, increasing its complexity. The time required to confirm that the product behavior is in accordance with the specifications and/or datasheets also increases with complexity. So post and pre silicon validation with less consumption of time is required. Here the study of methods of pre and post silicon validation methods with some examples of IPs are taken. They are studied and their results are also studied.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Rapid advancements in the field of very large scale integration (VLSI) technology has pushed the technology node into deep sub-micron figures allowing the industry to observe a shift in the trend from system on board to system on chip (SoC) in which all the components of a system that were earlier on a big printed circuit board, have now been pushed to within the boundaries of a single integrated circuit (IC). The die size on the other hand has remained the same. Consequently more logic devices are accommodated in the same die area allowing most of the required functionality to be present on the chip itself. This has resulted in high performance systems at the cost of an increase in complexity. Modern paradigm for the semiconductor product design is a central processing unit (CPU) core based SoC in which additional functionality is provided by intellectual property (IP) cores connected to the CPU core as shown in fig 1.1. In every new generation of the SoC, more functionalities are added. In other words, it has a higher number of IPs compared to the previous one, increasing its complexity. The time required to confirm that the product behavior is in accordance with the specifications and/or datasheets also increases with complexity.

The activity of verifying the design is a crucial phase in the product development cycle. It starts as soon as the design specifications are completed and runs in parallel with the design implementation activity. The verification phase evaluates the correctness of the design implementation to meet the specifications. Simulation checks and formal methods are commonly used for verification focused on uncovering the register transfer level (RTL) design errors for quality control. Verification does not cater to all the functional bugs as simulation is very slow and formal verification methods face scalability problems [1]. Hence another evaluation process, known as validation, is employed in the product development cycle.

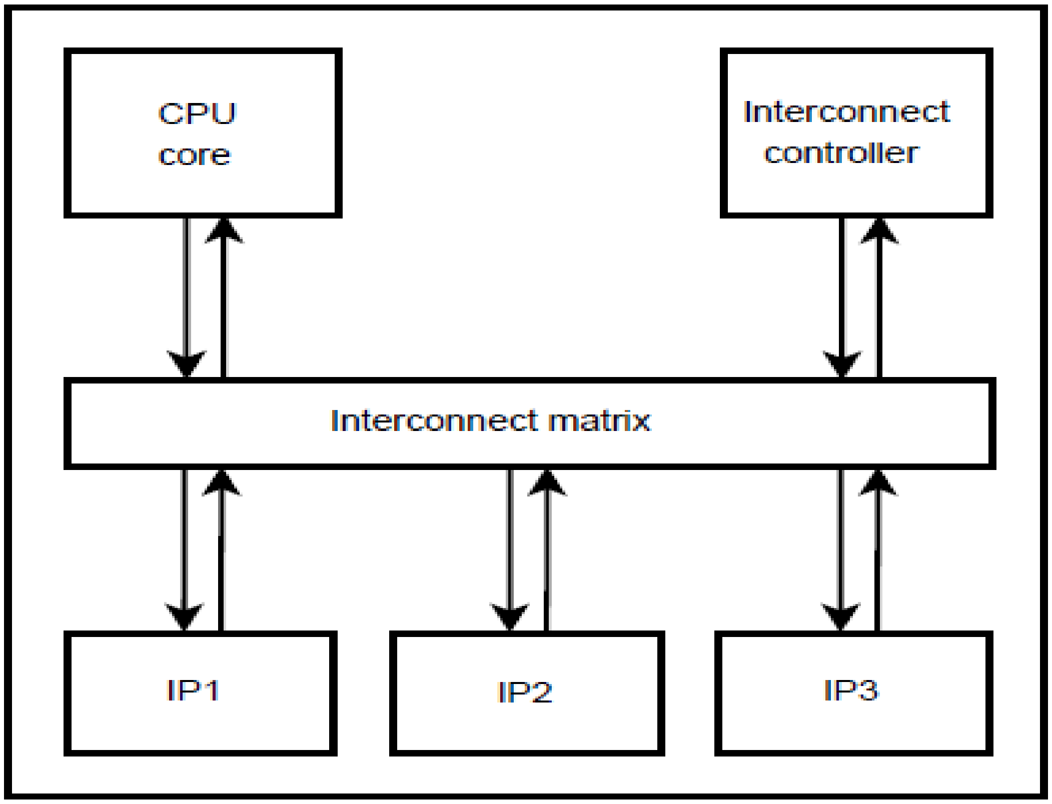


Fig. 1.1 Architecture of SOC.

.

Validation is a quality assurance phase which begins when the design is implemented, and it tests the design to meet the desired operational requirements. The validation process aims to establish evidence that the design is _t for the intended purpose. It involves the activities of test planning, test execution and response analysis. In test planning, the functionalities to be tested and the stimuli for testing them, are determined. In test execution, the test stimuli are applied to the device under validation. The responses obtained by the application of tests, are analyzed for pass/fail results. Modern SoCs contain multiple IPs with some of them working with external devices. Each IP is individually validated and the ones working with external devices are validated multiple times against different external devices manufactured by different vendors. For instance an IP of a ash memory controller works with memory devices that may be externally connected and would be validated multiple times with different ash memory devices.

1.1 MOTIVATION

The growing demand of the electronic market for versatile products and time to market pressure have made the VLSI industry adopt a new electronic design paradigm which is a processor core based SoC. In such a design, one or more CPUs are the central entity and the required capabilities according to the design specifications are provided by multiple IPs interfaced with the core. Above it, the SoC is made programmable so that it is employable in multiple application domains. The hardware is tightly coupled to the embedded software to implement a particular functionality. Software has thus become an integral part of the SoC making it an integrated system rather than a standalone entity. Therefore the software must be validated along with the hardware. The increasing complexity of the hardware and the embedded software has made the validation process a complex time consuming task. Large investments, in terms of both time and capital, are required to verify the design in order to prevent severe bugs to reach the mass manufacturing phase.

1.2 OBJECTIVE

The objective of this thesis is to learn the various validation process which are pre silicon and post silicon in the industry and also to look at the some examples to understand it how validation is done before sending to the customer.

1.3 LITERATURE REVIEW

1.3.1 POST-SILICON VALIDATION OPPORTUNITIES

A approach is given by Mehdi Karimibiuki et al. in (4). They have used the field programmable gate array (FPGA) prototype of an SoC to determine the RTL code coverage of the validation tests instead of the software code coverage of virtual prototypes in [9]. Modifications to the RTL code of the SoC capture the execution of the different sections of the RTL code in order to compute the statement and branch coverage when the validation tests run on the FPGA prototype. These approaches try to establish coverage metric for the PSV tests and allow good quality tests to be ready before the arrival of the silicon prototype.

To enhance the observability and controllability during PSV, debug assisting structures must be added to the SoC in the design phase. Embedded logic analyzer (ELA), discussed by Ho Fai Ko et al. in [10], is such a design for debug (DFD) structure consisting of embedded memories and trigger units. The ELAs are distributed across the SoC to tap important on-chip signals. The trigger unit monitors a set of signals to detect a pre-defined event which is followed by sampling of the tapped signals and then storing them in the embedded memories or trace buffers. The tapping is achieved by using the testing wrappers around the concerned blocks. Miron Abramovici in (11) proposes the addition of reconfigurable structures and supporting logic structures into the SoC for in-system silicon validation. The reconfigurable structures namely programmable trigger engine and reconfigurable logic engine, are interfaced to the standard joint test action group (JTAG) port through a primary controller allowing its reconfiguration according to the validation requirements. A tracer captures the tapped system signals which are selected by a signal probe network. This system can be configured for signal tracing and analyzing the logic, obtaining scan dumps and for performing on-chip functional testing of a block. When the trace buffer overflows, its content is offloaded over the JTAG port. The amount of trace data is limited by the width and depth of the trace buffer which are kept small to minimize the area overhead.

In [13], Kanad Basu and Prabhat Mishra have presented a method for signal selection based on total restorability. The signal paths are classified into dependent and independent paths. Then, using the probabilities of the different input signals to affect the output signal on different paths, their edge values are computed. An edge is a signal path containing combinational circuit elements between two flip flops. The edge value is a measure of how one end of the path controls the other end. Using these edge values, flip flop values are computed to filter out the flip flop signals to be traced for complete restoration of non-traced values. In the attempt to reduce the time invested in validation, different concepts have been consistently proposed. In (6), a validation system for faster validation of microprocessors has been presented by Ilya Wagner and Valeria Bertacco. Microprocessors are validated by running constrained randomized sequences of instructions on the hardware and in simulation. The results from the tests on hardware are compared with those of simulations for pass/fail decisions. Obtaining the results from simulation is a time consuming process as it is very slow in comparison to the actual hardware. The validation time is reduced using the proposed scheme as the simulation stage, required to get the reference outputs of various sequences of random instructions, is bypassed. This is due to the fact that reversible programs are generated by this system enforcing the same state of the processor after the test execution as compared to the state before testing. Hence the final state is already known at the start of the test. By studying the instruction set of a microprocessor, the inverses of various arithmetic, logical, branching, memory, floating instructions are computed and a library is created. After a randomized sequence is generated, the inverse of each instruction from the library is appended to the sequence in the reverse order. Now, the complete sequence has instructions as well as their inverse code present within the same sequence in an order intended to bring the processor back to the initial state after the entire sequence finished execution. Thus the simulation stage is not needed as the final state is known beforehand and if a bug is present, the final state will not match the initial state. In [5], the authors analyze the effectiveness in reducing the complexity of the validation of a core based chip having pre-validated IPs. In such scenarios the integration of the components and the communication architecture is fault prone.

1.3.2 PRE SILICON VALIDATION OPPORTUNITIES

A systematic approach in which all the interfaces are validated first rather than the entire SoC, is proposed to reduce overall validation time. All data transfers, or communications, among different SoC components, are routed through the communication architecture. Common integration problems for an SoC are identified and classified into three categories based on the parts of the SoC involved. These are component to communication architecture, component to component and communication architecture problems. This identification and classification leads to a logical sequence for interface validation. First the components to communication architecture interface is verified by high level test benches. Then the component to component interfacing is validated by enforcing communication among the different components in pairs. Finally the entire communication architecture is validated using all the components simultaneously. Validating the interfaces in that sequence followed by a full system simulation will reduce the overall time consumption as interface validation require small, directed test benches in simulation. In [14] functional self testing is advocated by Krstic et al. to achieve at-speed testing of high speed circuits not possible with external testers. At-speed testing requires execution of test cases at the maximum clock frequency on which the circuit is designed to work. First the processor is self tested by a test program which applies functional patterns for testing. Then the processor is reused to test other components of the system. The processor executes functional tests by generating patterns and analyzing the response. The structural tests of stuck-at fault testing, path tests of delay testing, maximum aggressor tests of bus testing and test patterns for component testing are mapped to sequences of instructions which are executed by the processor core in order to perform the required tests. The use of instructions eliminates the need of scan chains and hence at-speed testing is made possible.

1.4 THESIS ORGANISATION

This thesis is organized into five chapters, of which the introduction, literature review covers the first chapter. The rest of the thesis is organized as follows:

Chapter 2 describes the IP cores in Soc design which are being validated before sending to the foundary

Chapter 3. describes the pre-silicon validation, its method and how we have implemented it on the IP cores.

Chapter 4 describes the post-silicon validation, its method and how we have implemented it on the IP cores.

Chapter 5 includes the result, conclusion and the future scope.

CHAPTER 2
INTODUCTION TO IP CORES IN SoC

With rapid strides in Semiconductor processing technologies, the density of transistors on the die is increasing in line with Moore's law which in turn is increasing the complexity of the whole SoC design. With manufacturing yield and time-to-market schedules crucial for an SoC(System on Chip), it is important to select verification and analysis solutions that offer the best possible performance, while minimizing iteration time and data volume. With the advent of cutting edge technology applications like set top boxes, HDTV, an increasingly evident need has been that of incorporating the SoC the whole system - on a single silicon i.e., Silicon On Chip (SoC) using standard IP-Cores. In an IP-Core based SoC design. A streamlined verification and analysis flow can contribute significantly to the success of a product. A strategy is devised for a more streamlined approach in IP-core based SoC verification which helps in smooth transition from design to chip tape-out stage.

2.1 INTRODUCTION

Hardware designs have reached a mammoth scale today, with over ten million transistors integrated on a single chip[1]. This breakthrough in technology has, in fact, reached the point, where it is hard to design a complete system from scratch. Industry has already started designing SoC's from a large repertoire of Intellectual Property Components or IP Cores sold by many vendors. System-on-chip designs usually involve the integration of heterogeneous components on a standard bus[3,4,8]. These components may require different protocols or have different timing requirements. Moreover, designers often do not have complete knowledge of the implementation details of each component. For example, vendors may want to protect their IP Cores by only providing interface specifications. Consequently, the validation of such designs is becoming more and more challenging. In this paper, we outline a new methodology for formally verifying IP Core based system-on-chip designs. It is well known fact that verification today constitutes about 70% to 80% of the total design effort, thereby, making it the most expensive component in terms of cost and time, in the entire design flow which is expected to get even worse for SoC designs.

2.2 IP CORE BASED SOC DESIGN

Let us open by defining what an SoC is and is not. A System on Chip (SoC) is an implementation technology[6], not a market segment or application domain. SoC's may have many shapes and many different variants, but a typical SOC may contain the following components: a processor or processor sub-system, a processor bus, a peripheral bus, a bridge between the two buses, and many peripheral devices such as data transformation engines, data ports (e.g.,UARTs, MACs) and controllers (e.g., DMA) [1]. In many ways, the verification of an SoC is similar to the verification of any ASIC: you need to stimulate it, check that it adheres to the specification and exercise it through a wide set of scenarios.

SoC verification is special and it presents some special challenges:

Integration: The primary focus in SoC verification is on checking the integration between the various components. The underlying assumption is that each component was already checked by itself. This special focus implies a need for special techniques.

Complexity: The combined complexity of the multiple sub-systems can be huge, and there are many seemingly independent activities that need to be closely correlated. As a result, we need a way to define complicated test scenarios as well as measure how well we exercise such scenarios and corner cases.

Reuse of IP blocks: The reuse of many hardware IP blocks in a mix-and-match style suggests reuse of the verification components as well. Many companies treat their verification IP as a valuable asset (sometimes valued even more than the hardware IP). Typically, there are independent groups working on the subsystems, thus both the challenges and the possible benefits of creating reusable verification components are magnified.

2.2.1 IP CORES

IP cores are pre-designed and pre-verified complex functional blocks. According to their properties, IP cores can be distinguished into three types of cores[11].

Soft-cores: Soft-cores are architectural modules which are synthesizable. They offer the highest degree of modification flexibility. On the other hand, a lot of physical design issues need to be faced before the core can be fabricated. This makes the soft-cores very unpredictable in terms of performance. A synthesizable soft-core consists of a set of technology-independent HDL files, synthesis constraints, test-bench and validation information and adequate information. Firm-cores are encrypted black boxes that are integrated into design flow in the same way as library elements[11].

Firm-cores: Firm-cores are delivered as a mix of RTL code and a technology-dependent net-list, and are synthesized with the rest of ASIC logic. They come ready for routing analysis and do not present significant difficulties for floor-planning, placement, and routing. They have the same routability properties as soft-cores. The performance of the block is still unpredictable[11].

Hard-Cores: Hard-cores are mask and technology-dependent modules that already have physical layout information which give predictable performance. The key deliverable is a fully verified layout in Graphical Design System II (GDSII) format, along with a design for a test structure and test patterns. The drawback of hard-core is that the cores can not be customized for a particular design application. Hard-cores require more model support than firm-cores, which increases development cost. On the other hand, the usage cost is lower because timing validation, test strategies, etc., have already been built into the design. Monolithic hard-cores create a jigsaw puzzle problem for ASIC layouts. When more than one hard-core is used, the ordinary place and route techniques cannot be used due to the existence of a strange, non-rectangular area left for routing other non-core logic[11].

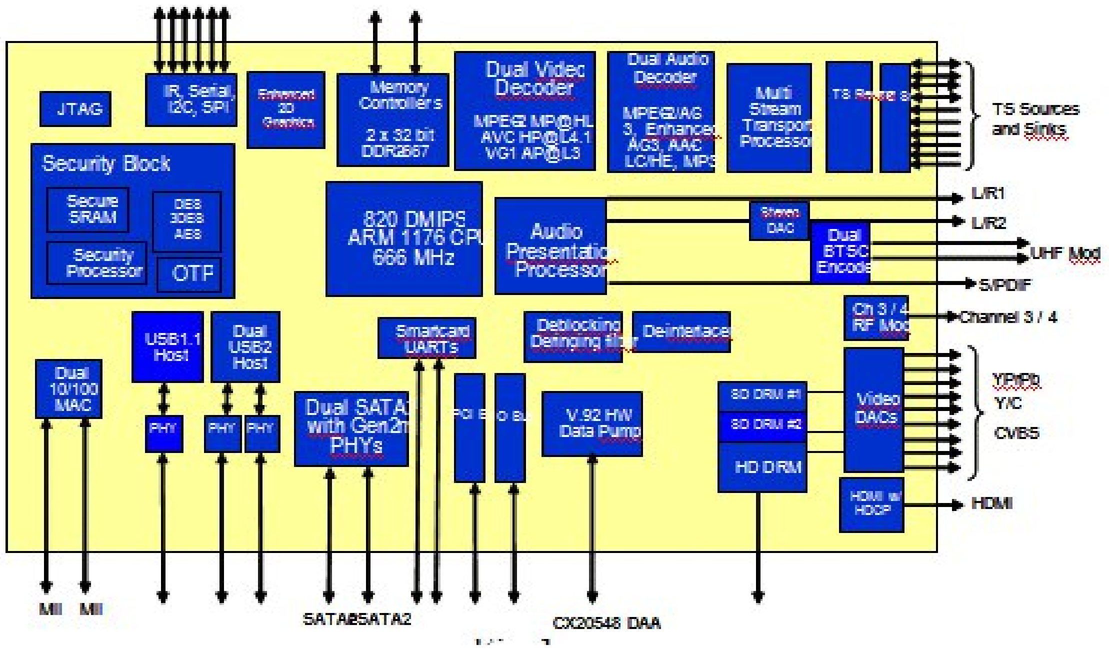


Fig. 2.1 IP core based SoC design.

Integration of Sub IPs/Blocks/Modules/Clusters: Before the actual SoC verification starts, the first step is to integrate/stitches of the sub-blocks/sub-IPs/sub-clusters into the SoC level verification environment. This is one of the major activities of SoC verification.You also have to develop connectivity checkers which will make sure whether the integration of the sub-blocks to SoC is done properly or not. This will play a critical role in SoC level verification because if the integration is not done properly, then you cannot expect SoC level targeted functionality out of it.

2.3 DESIGN METHODOLOGIES

Every advancement in microelectronics processing technology is always followed by the development of new design technology. This new design technology, a so-called linchpin technology[12,13], becomes the building block to lead the design entering the next generation of design methodology. The design methodology responds with an adaptation to the new design process resulting in an incremental increase in productivity. It alters the relationship between the designers and the design by introducing a new level of abstraction.

A linchpin technology always comes along with its specific design methodologies.

2.3.1 AREA DRIVEN DESIGN

Area-Driven Design (ADD)[11] is the most basic and the simplest methodology used in creating ASIC designs. It is driven to achieve the primary goal target in creating a design which can fit into a limited budget area. The designer is challenged to implement as much functionality as possible in a single piece of silicon. The ADD methodology is used to achieve small sized ASIC's. Most ADDs are created from scratch and do not offer any design reuse. The main ADD activity is in logic minimization. The synthesis optimization is to produce the smallest design which can meet the intended functionality. In this methodology, no floor planning information is used at the RTL or gate level analysis.

2.3.2 TIMING DRIVEN DESIGN

Timing-Driven Design (TDD) [11] is a methodology for optimizing a design in a top down, timing convergent manner. It is driven by the design requirement for meeting performance or power consumption. The methodology is used to achieve a moderately sized complex ASIC design. In general, the complexity of a TDD circuit is between 5000 to 250K gates. It is primarily a custom logic design, offering a very slim possibility of design reuse. The TDD methodology imposes a more floor plan-centric design methodology that supports incremental changes of the design. The floor planning and timing analysis tools can be used to determine the location of placement sensitive areas, allowing the results to be tightly coupled into the design optimization process. TDD relies on three linchpin technologies: interactive Floor-Planning (FP) tools, Static Timing Analysis (STA) tools, and using compilers to move design to higher abstraction with timing predictability. FP tools give accurate estimation on the delay and area early in the design process. They address the timing and area convergence problems which occur in the design process between synthesis and 'place and route'. STA enables a designer to identify timing problems and perform timing optimizations across the entire ASIC. It reduces the validation stress in catching bugs using a slower timing-accurate gate-level simulation. The advancement in compiler technology enables the designer to move the design into higher abstractions while retaining timing predictability.

2.3.3 BLOCK BASED DESIGN

Block-Based Design (BBD) [11] is the design methodology used to produce designs that are reliable, predictable, and can be implemented by top-down partitioning of the design into hierarchical blocks. It introduces the concept of creating a system by integrating blocks of pre-designed system functions into a more complex one. The methodology is used to create medium-sized complex ASIC's with complexity between 150K to 1.5M gates. BBDs are primarily created as custom logic designs. In comparison to TDD; BBD offers a better chance for reuse, although in reality, very few BBDs are reusable.

2.3.4 PLATFORM -BASED DESIGN

Platform-Based Design (PBD)[11] is a methodology which is driven to increase productivity and time to market by extensively using design reuse and design hierarchy. It expands the opportunities to speed-up the delivery of derivative products. PBD achieves high productivity through extensive and planned design reuse. Productivity is increased by using predictable, pre-validated blocks that have standardized interfaces. The methodology focuses on better planning for design reuse and less modification on the existing functional blocks. PBD is used to design large sized complex ASIC's with design complexities greater than 300K gates.

The PBD methodology separates the design into two categories of activity: block authoring and block integration. Block authoring uses a methodology which is suited to the block type such as TDD or BBD. Blocks are created with standardized interfaces so they can be easily integrated with multiple target designs. Block integration focuses on designing and verifying the architecture of the system and the interfaces between the blocks. PBD focuses around a standardized bus architecture and increases its productivity by minimizing the amount of custom interface design or modification of the blocks. The test for the design is incorporated into the standard interfaces to support each block’s specific test methodology. This allows for a hierarchical, heterogeneous test architecture.

2.4 TAPE OUT

Every design group ultimately needs to answer this question[14]. The means for answering are always insufficient, as verification quality is so hard to measure. Code coverage, toggle or fault coverage and bug rates are all useful measures, but they are very far from complete, and fail to identify many of the complex combined scenarios that need to be exercised in an SoC. To solve this dilemma, there is need for coverage metrics that will measure progress in a more precise way. To summarize, there is always an element of “spray and pray” in verification, hoping you will hit and identify most bugs. In SoC’s, where so many independent components are integrated, the uncertainty in results is even greater. There are new technologies and methodologies available today that offer a more dependable process, with less “praying” and less time that needs to be invested.

CHAPTER 3
PRE SILICON VALIDATION

With any semiconductor device, design verification is very critical to the functional and operational quality of the finished product, in which a single defect in any one function could cause the failure of the device and necessitate a re-spin. At the same time, the integration of more functions into single devices, shrinking product design cycles, and compressed verification and validation cycles because of time to market pressures are increasing device complexity. Increasing device complexity, shrinking product-design cycles, and time-to-market pressures are compressing the verification and validation cycles. Validation and debugging are complex processes, where a large amount of data without a unified approaches that can address every potential problem. Verification and validation helps to detect functional errors. Functional errors occur because of incorrect capture of the functional requirements, making incorrect assumptions during design, or introduce mistakes in the design.

3.1 INTODUCTION TO PRE-Si VALIDATION

Pre-silicon validation is generally performed at a chip, multi-chip or system level. The objective of pre-silicon validation is to verify the correctness and sufficiency of the design before sending the design for fabrication. This approach typically requires modeling the complete system, where the model of the design under test may be RTL, and other components of the system may be behavioral or bus functional models. Pre-silicon functional verification is time-consuming, some of the validation issues include interactions with other peripherals and subsystems, signal integrity, timing, and board level issues. The full chip verification environment consists of the entire ASIC along with bus functional models driving the various inputs of the ASIC. The RTL is complete and there are no bus functional models or missing RTL from the device under test. The full chip environment is considerably slower than the module level environment. Full chip tests are also harder to debug given the size of the design and the slower run times. Sometimes based on the size of the design the design is regressed/ verified using an RTL accelerator. Most of the tests in this level focus on end to end simulations to ensure that the entire path through the device is clean and free from bugs.

In pre-silicon design verification, each of the functional blocks within the device is verified for logical and algorithmic correctness. Full-chip verification tests the interaction of all functional blocks and includes writing cycle-accurate, event-driven, and transaction-driven tests. Cycle-accurate tests examine the behavior of a given functional block on a cycle-to-cycle basis. Event-driven tests evaluate responses to events such as interrupts and determine how the functional blocks handle these events. Transaction-driven tests evaluate interactions using transaction-level abstractions, such as memory reads and memory writes. Verification involves software-based gate-level and RTL (register-transfer-level) simulations.

3.1.1 Key features

By subjecting the design under test (DUT) to real-world-like input stimuli, pre-silicon validation aims to:

- Validate design sufficiency
- Validate design correctness.
- Verify implementation correctness.
- Uncover unexpected system component interactions

Periodic intervals to re-create real-life traffic scenarios in a pre-silicon validation
environment.

Reactive test generation implies a change in test generation when a monitored
event is detected during simulation.

3.2 PROCESS OF VERIFICATION

Verification is a process used to demonstrate the functional correctness of a design. The main purpose of "functional" verification is to ensure that a design implements intended functionality. Functional verification is the activity where the design or product is tested to make sure that all the functions of the device are indeed working as stated. It is a complex and time-consuming design step, accomplished by converting the specifications into a combination of: Stimuli and expected results scenarios, verifying that the design produces the expected results when applied with the stimuli.

Golden model and stimuli constraints, verifying that, whatever the constraint compliant stimuli, the golden model and RTL behavior are equivalent.

Properties and stimuli constraints, verifying that, whatever the constraint compliant

stimuli, the properties hold true.

3.2.1 GOAL OF VERIFICATION

As a verification engineer our aim is to make sure the device can accomplish that task successfully that is, the design is an accurate representation of the specification. The process of verification parallels the design creation process [1].

Find all, or as many as possible, of the bugs in my unit/cluster/area as quickly as possible, or at least before first silicon. Find bugs in general, not just in my unit/cluster/area. Find bugs at the right level (cluster vs. full chip). Understand the design. Provide continuity of knowledge of the unit (not just how and why does it work, but why was it designed that way). Quickly analyze and root cause bugs as they appear. Identify and prevent others from making harmful or unnecessarily high-risk changes to the design. Create high quality collateral (test plans, tests, checkers, coverage, CTEs, APIs) for testing the design. Continuously improve and increase effectiveness in all activities.
Enable the design team to do their job by identifying issues and increasing the quality of their work. Give the design team confidence that when they make mistakes and code bugs, we will find those bugs.

A system specification derives the verification strategy that is followed for particular chip verification. These specifications along with the test plan answers what to verify. Verification processes answers what tools and processes to use, and verification methodology answers how to verify the chip. Verification methodology and implementation is dependent on the set of available tools and the process adopted for verifying the chip.

3.3 PILLARS OF VALIDATION

Nearly all validation collateral (what the validation team produces) falls into one of three categories: stimulus, checking and coverage. Each of these may look different for various validation groups, but all groups have them and each is critical for success. The overall quality of validation is limited by the quality of the lowest of these three areas:

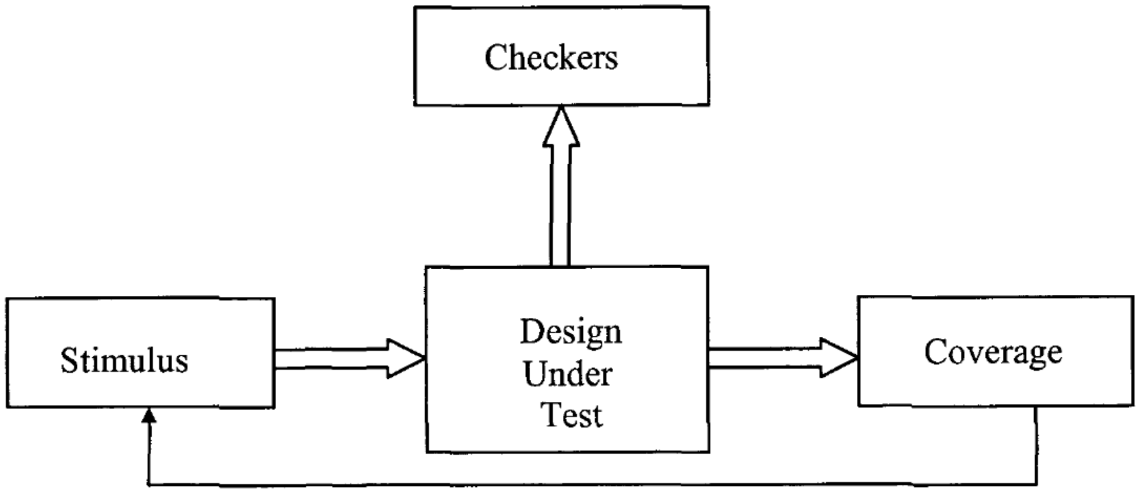


Fig.3.1: Pillars of Validation

3.3.1 STIMULUS

Stimulus is frequently referred to as tests or input. Stimulus is the content that you use to exercise the design under test (DUT). Good stimulus will quickly exercise a large amount of functionality in a minimal amount of simulation time. If you don't have good stimulus, you will not exercise important conditions where bugs exist. In uAV, stimuli are cluster or full chip tests along with tools to inject state or transactions into the model

.

3.3.2 CHECKERS

Checking is verifying that the behavior of the DUT matches some reference model or expectation. For example, a full-chip architectural checker like Chekhov will verify that the results of an increment-register instruction results in the register's value being one more than it was before the execution of the instruction. There are many different checkers that are deployed in a project and include:

Self-checking tests.
Assertions in the RTL Forbidden statements in protocytes.
Mini-checkers in (and outside) the cluster test envirorunents (CTEs).
Bus checkers.

3.3.3 COVERAGE

Coverage is used to identify gaps in the stimulus and make changes in the stimulus to close those gaps. Failure to collect and use coverage results to improve stimulus quality will resuts in a failure to exercise important cases, allowing bugs to escape.

Gate length of the transistor is minimum. It has been observed in other processes that the optimum ESD performance occurs for the minimum channel length, due to the better activation offered by the parasitic NPN transistor. This is the reason why it is advised to use the minimum gate length for ESD protection devices in IO's. Large Width of the transistor, due to current density considerations, it is advised to use large transistors. N-MOS transistor has been proved to be efficient as a clamping device without LDD structure. So LDD structure is suppressed. A complete I/O solution will be characterized on silicon realized during the process development, for more information see the documents provided with the I/O library

3.4 FRONT END DESIGN PHASE

The usual sequence a designer goes through when developing an integrated circuit is shown pictorially in Fig 3.2.

In broad terms the steps can be classified as:

Capturing a description of the design using a HDL (Hardware Description Language). Simulating the design to prove that it works correctly and meets specification. Logic synthesis, where the design-description is converted into a physical gate description. Physical Design involves the activity of organizing the layout of the integrated circuit to satisfy various physical and electrical constraints. E.g. power dissipation, pin-to-pin delays, electrical loading, etc.

Fig. 3.2 Phases of Design.

Starting at the front-end, the first step that needs to be performed is the design capture of each module or design unit and development of the relevant test benches. Graphical entry methods can also be used. The objective at this stage is to ensure that the syntax and semantics of the HDL code written by the designer are correct. Any coding errors not picked up at this stage will be passed through to the simulation process which could have serious consequences as undetected coding errors may cause the simulation to fail or produce spurious results. Once the HDL code is correct a designer can safely move onto the second stage and start the simulation process. Modules that represent self-contained parts of the design can be simulated as soon as they have been captured. Other modules that rely on receiving signals from other parts of the design may require a dedicated test bench to be created in order to be simulated, until all the other relevant and dependent modules have been captured. The overall effect of using this particular technique will be to increase the amount of testing that needs to be done at the sub-system integration level. It can be tempting for a designer to move on to stage 3 and start the synthesis process as soon as all the parts of the design have been successfully simulated. At the fundamental level this means establishing that all executable statements in the design have been executed and that all the decision branches have been taken.

3.4 AUTOMATIC TEST GENERATION

The test creation and/or generation methodology is critical in building a system level pre silicon validation environment capable of generating real-world-like stimuli. The test generation methodology is closely interrelated to the results checking strategy. A dynamic test generator and checker are more effective in creating very interesting, reactive test sequences. They are more efficient because errors can be detected as they happen. An automated test generation tool should be capable of handling directed testing, pseudo-random testing and reactive testing.

In directed testing, users specify the sequence of events to generate. This is efficient for verifying known cases and conditions. Pseudo-random testing is useful in uncovering unknown conditions or comer cases. Pseudo-random test generation, where transactions are generated from user-defined constraints, can be interspersed with blocks of directed sequences of transactions at periodic intervals to re-create real-life traffic scenarios in a presilicon validation environment. Reactive test generation implies a change in test generation when a monitored event is detected during simulation.

.

CHAPTER 4
POST-SILICON VALIDATION

The procedure to accomplish a particular test after determining its stimulus depends on the validation engineer aiming to cover all corner cases for the target. The hardware and software functionality is the prime aspect of validation, as malfunctioning of the system proves very costly when already deployed in the field. The functional validation discipline of PSV is performed to evaluate the functional correctness of different parts of the SoC. Ideally the pre-silicon validation phase should uncover all functional bugs, but this has become impossible to achieve with increasingly complex designs. Hence functional coverage is the prominent industrial coverage metric in which a set of functional points which the validation process should address and verify, are identified by the engineer [4] based on the specifications and datasheets. From a commercial view, successfully verifying all practical use cases in an end user environment is the highest concern.

4.1 INTRODUCTION TO POST-SILICON VALIDATION

PSV involves significant capital and time investment due to the unavailability of global standards and procedures for PSV, limited access to the internal circuitry and lack of automation tools. Efforts are made to establish standard coverage metric for PSV, Kai Cong et al. [9] have used a virtual prototype or simulator of an SoC to evaluate the coverage of the PSV tests. It is the software model of a hardware which allows the unmodified execution of the programs, written for that hardware. The software model is enhanced to capture the execution data whenever a program runs on it. The execution data show which code sections of the software model are executed allowing the computation of the traditional code coverage metric like statement, branch, function and block coverage of a program. Two new hardware specific coverages, register coverage and transaction coverage, are defined which are also computed using the execution data. The register coverage metric is the number of times a register is accessed in a test and the transaction coverage metric is the number of times different transactions or events take place in the virtual device during the test. In PSV, the same tests are executed on the hardware prototype and the device state is recorded at different events. The recorded data from the test execution on the virtual device and the actual device is then compared to discover inconsistencies.

4.1.1 KEY FEATURES

As the amount of the trace buffer is limited, less number of signals and their states can be stored. To increase the observability of the circuit, the non-traced values must be computed from the traced signals and the input test vector using restoration algorithms. Selection of the signals in such cases play a significant role as the restoring capability of some set of signals may be higher than the others. The possibility of a bug capture depends on the extent of signal restoration. In (12), partial restoration calculations are performed for various signals and based on them the signals to be traced are selected.

4.2 VALIDATION PROCESS

Without loss of generality, an IP core is seen as a set of registers which are accessed by their addresses over a bus. An IP block exports its functionality via these registers [15]. The register set includes configuration, command, data and status registers. To exercise a functionality, the associated register bits are modified and tested according to the values mentioned in the datasheet. The test cases are created following this procedure to verify the IP functionality and its integration. The tests are categorized as directed tests and random tests. Directed tests are manually created by a validation engineer to cover all functional coverage points as planned. These can be short tests and long tests. Short tests are aimed at individually validating each and every feature supported by the IP while long tests exercises multiple features and are combinations.

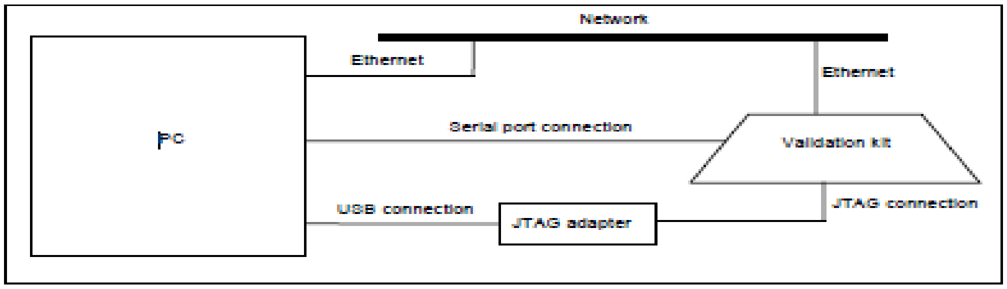


Fig 4.1: Validation set-up

4.3 POST SILICON METHODOLOGIES

Post-silicon validation involves a number of activities including validation of both functional and timing behaviour as well as non-functional requirements. Each validation has methodologies to mitigate these.

4.3.1 POWER-ON DEBUG

Powering on the device is actually a highly complex activity. If the device does not power on, the on-chip instrumentation architecture is typically not available, resulting in extremely limited (often zero) visibility into the design internals. This makes it difficult to diagnose the problem. Consequently, power-on debug includes a significant brainstorming component. Of course, some visibility and controllability exist even at this stage.

The debug activity then entails coming up with a bare-bone system configuration (typically removing most of the complex features like power management, security and software/firmware boot mechanisms) that can reliably power on. Typically, starting from the time the silicon first arrives at the laboratory, obtaining a stable power-on recipe can take anywhere from a few days to a week.

Once this is achieved, the design is reconfigured incrementally to include different complex features. At this point, some of the internal design-for-debug (DfD) features are available to facilitate this process. Once the power-on process has been stabilised, a number of more complex validation and debug activities can be initiated.

4.3.2 BASIC HARDWARE LOGIC VALIDATION

Compatibility validation refers to the activities that ensure the silicon works with various versions of systems, application software and peripherals. Validation accounts for various target use-cases of the system, platforms in which the SoC is targeted to be included and so on. Compatibility validation includes, among others, the following:

1. Validation of system usage with add-on hardware of multiple external devices and peripherals

2. Exercising various operating systems, applications, protocols and communication infrastructures

In addition to the generic complexities of post-silicon validation, a key challenge here is the large number of potential combinations (of configurations of hardware, software, peripheral, use-cases, etc) that need to be tested. It is common for compatibility validation to include over a dozen operating systems of different flavours, more than a hundred peripherals and over 500 applications.

4.3.3. ELECTRICAL VALIDATION

Electrical validation exercises electrical characteristics of the system, components and platforms to ensure an adequate electrical margin under worst-case operating conditions. Electrical characteristics include input-output, power delivery, clock and various analogue/mixed-signal (ams) components. Validation is done with respect to various specification and platform requirements. For example, input-output validation uses platform quality and reliability targets.

As with compatibility validation, a key challenge is the size of the parameter space. For system quality and reliability targets, validation must cover the entire spectrum of operating conditions (voltage, current, resistance, etc) for millions of parts. The current state of practice in electrical validation is an integrated process of (1) sampling the system response for a few sample parts, (2) identifying operating conditions under which the electrical behaviour lies outside specification and (3) optimisation, re-design and tuning as necessary to correct the problem.

Unlike logic and compatibility validation, electrical validation must account for statistical variation of system performance and noise tolerance across different process corners. PRQ requires the average defect to be low, typically less than 50 parts per million.

4.3.4 DEBUGGING IN THE PRESENCE OF NOISE

A consequence of the fact that we are using actual silicon as the validation vehicle is that we must account for factors arising from physical reality in functional debug, that is, effects of temperature, electrical noise and others. A key challenge in post-silicon validation is to consequently find a recipe (for example, via tuning of different physical, functional and non-functional parameters) to make a bug reproducible.

On the other hand, the notion of reproducibility in post-silicon is somewhat weaker than in pre-silicon validation. Since post-silicon validation is fast, an error that reliably appears once in a few executions (even if not 100 per cent of the time) is still considered reproducible for post-silicon. Nevertheless, given the large space of parameters, ensuring reproducibility to the point that one can use it to analyse and diagnose the error is a significant challenge.

4.3.5 SECURITY AND POWER MANAGEMENT CHALLENGE

Modern SoC designs incorporate highly-sophisticated architectures to support aggressive energy and security requirements. These architectures are typically defined independently by disparate teams with complex flows and methodologies of their own, and include their unique design, implementation and validation phases. The challenge of security on observability is more direct. SoC designs include a large number of assets, such as cryptographic keys, DRM keys, firmware, debug mode and the like, which must be protected from unauthorised access.

"

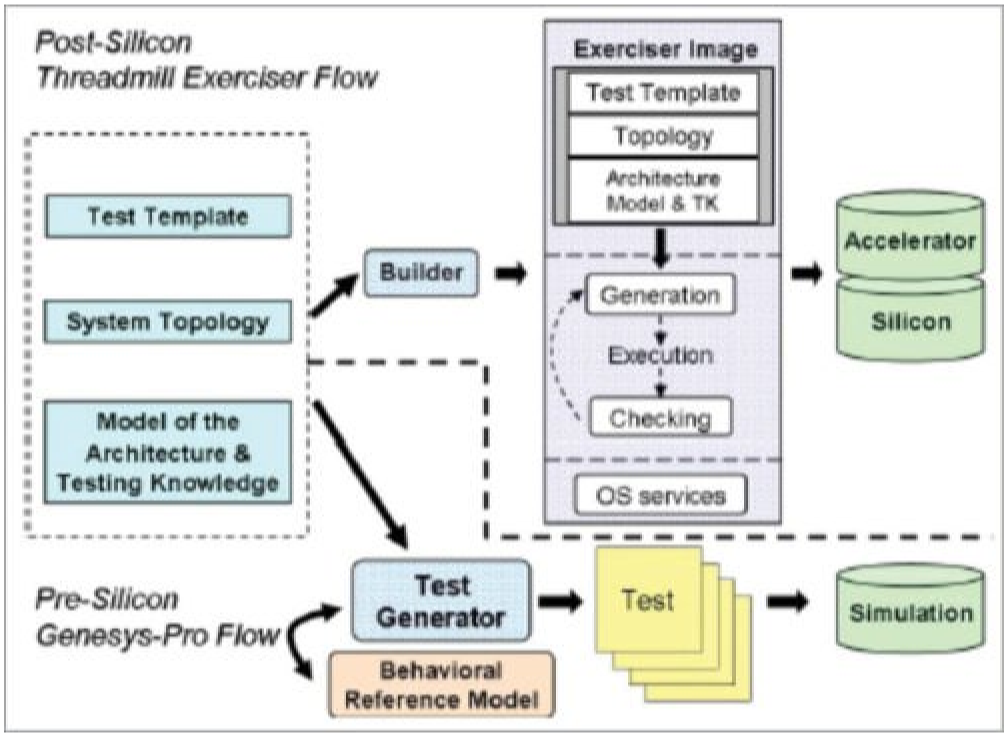


Fig. Security and power management challenges

Further, much of the DfD infrastructure is available on-field to facilitate survivability. This permits their exploitation by malicious hackers to gain unauthorised access to system assets after deployment. Indeed, many celebrated system hacks have made use of post-silicon observability features, causing devastating impact to the product and company reputation once carried out. Consequently, a knee-jerk reaction is to restrict DfD features available in the design.

On the other hand, lack of DfD may make post-silicon validation difficult, long and even intractable. This may delay the product launch. With aggressive time-to-market requirement, a consequence of such delays can be a loss of billions of dollars in revenue or even missing the market for the product altogether.

Power management features also affect observability, but in a different manner. Power management features focus on turning off different hardware and software blocks at different points of execution, when not functionally necessary. The key problem is that observability requirements from debug and validation are difficult to incorporate within the power management framework.

4.4 TEST EXECUTION

This involves setting up the test environment and platform, running the test and, in case the test fails, performing some obvious sanity checks (like, checking if the SoC has been correctly set up on the platform, power sources are connected and switches are set up as expected for the test). If the problem is not resolved during sanity check, it is typically referred to as a pre-sighting.

4.4.1 PRE-SIGHTING ANALYSIS

The goal of pre-sighting analysis is to make failure repeatable. This is highly non-trivial, since many failures occur under highly-subtle and coordinated execution of different IP blocks. For instance, IP A sends a message to IP C within a cycle of another IP B sending a different message to C. This may result in a buffer overflow (eventually resulting in a system crash), when occurring in a state in which input queue of C has only one slot left and before C has had the opportunity to remove some items from the queue.

Making failure repeatable requires running the test several times, under different software, hardware, systems and environmental conditions (possibly with some knowledge and experience on potential root causes) until a stable recipe for failure is discovered. At that point, failure is referred to as sighting.

4.4.2 BUG RESOLUTION

Once a plan of action has been developed for a sighting, it is referred to as a bug. A team is assigned for ensuring that it is resolved in a timely manner based on the plan. Resolution includes finding a workaround for the failure to enable exploration of other bugs and triaging, and identifying the root cause for the bug.

Triaging and root causing bugs are two of the most complex challenges in post-silicon validation. In particular, root cause for a failure observed on a specific design component can be in a completely different part of the design. One of the first challenges is to determine whether the bug is a silicon issue or a problem with design logic. If it is determined to be a logic error, the goal is typically to recreate it on a pre-silicon platform (such as RTL simulation and FPGA). The exact post-silicon scenario cannot be exercised in a pre-silicon platform. One second of silicon execution takes several weeks or months to exercise on RTL simulation.

4.4.3 TEST PLANS

These constitute arguably the most critical and fundamental readiness activity for post-silicon validation. The objective is to identify different coverage targets, corner cases and functionalities that need to be tested for the system being deployed.

Post-silicon test plans are typically more elaborate than pre-silicon plans, since these often target system-level use-cases of the design that cannot be exercised during pre-silicon validation. Test plan development starts concurrently with design planning. When the test plan development starts, a detailed design (or even an elaborate microarchitecture for the most part) is unavailable.

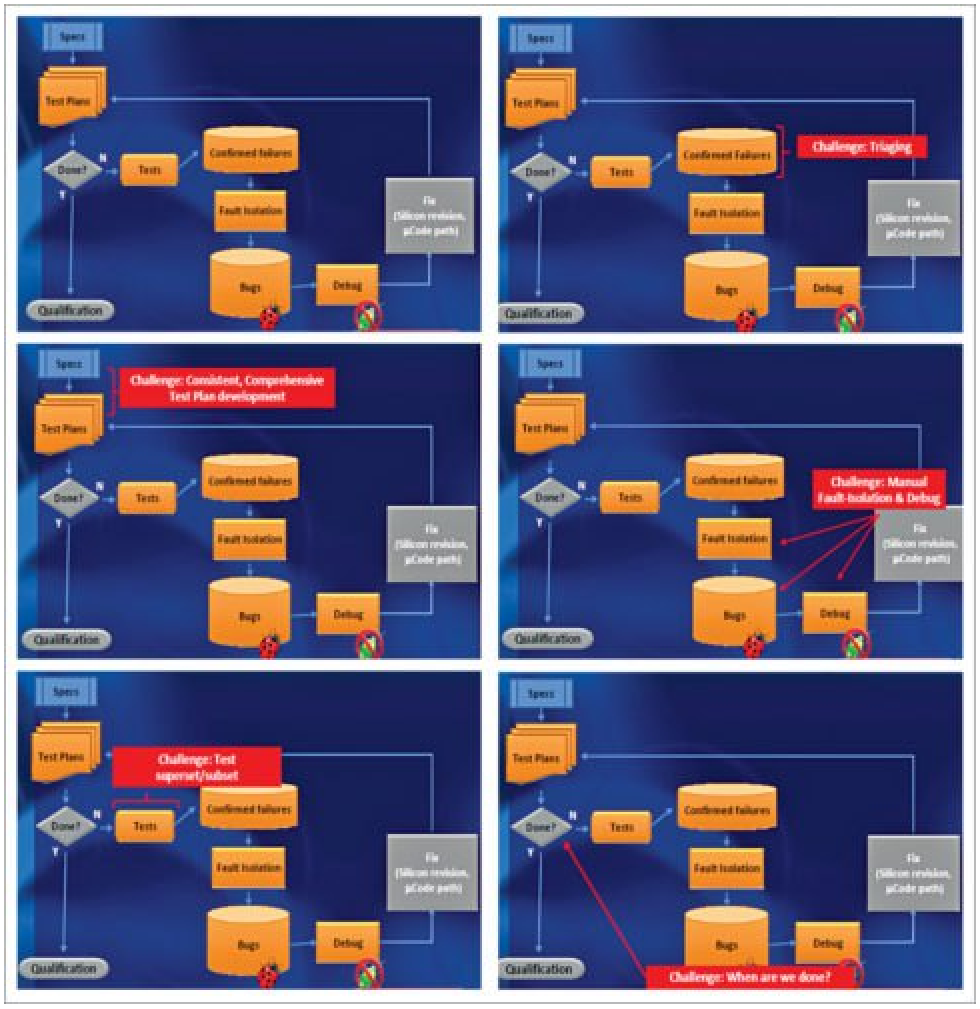

Specs
Test Plans
Done?
N
Y
Tests
Confirmed failures
Fault Isolation
Bugs
Debug
Fix (Silicon revision, µCode path)
Qualification
Challenge: Triaging
Challenge: Consistent, Comprehensive Test Plan development
Challenge: Manual Fault-Isolation & Debug
Challenge: Test superset/subset
Challenge: When are we done?

Fig Different Validation Challenges

4.4.4 TRANSPORT SOFTWARE

Access software refers to tools that enable transport of data off-chip from silicon. Data can be transferred off-chip either directly through pins, or by using available ports from the platform (USB, PCIe, etc). For example, transporting through the USB port requires instrumentation of the USB driver to interpret and route the debug data while ensuring that USB functionality is not affected during normal execution. This can become highly complex and subtle, particularly in the presence of other features in the SoC, such as power management.

Power management may, in fact, power down the USB controller when the USB port is not being used by the functional activity of the system. The instrumented driver ensures that debug data is still being transported while facilitating the power-down functionality of the hardware to be exercised during silicon validation.

4.4.5 ANALYSIS SOFTWARE

Finally, there are software tools to perform analysis on the transported data. There are tools to aggregate the raw signal or trace data into high-level data structures (for example, interpreting signal streams from the communication fabric in the SoC as messages or transactions among IPs), comprehending and visualising hardware/software coordinations as well as tools to analyse such traced and observed data for further high-level debug (such as estimating congestion across the communication fabric, traffic patterns during internal transactions and power consumption during system execution).

.

4.5 TEST GENERATION AND TESTING SETUP DESIGN

The central component of silicon debug is the set of tests to run. For validation to be effective, the tests must expose potential vulnerabilities of the design and exercise different corner cases and configurations.

4.5.1 POST-SILICON TESTS

Post-silicon tests can be divided into the following two categories:

4.5.1.1 FOCUSED TESTS

Such tests are carefully crafted by expert test writers to target specific features of the system (like, multiprocessor and chipset protocols, CPU checks for specific register configurations, address decoding and power management features). Developing such tests involves significant manual effort. Further, tests are often extremely long and targeted, running for several hours on silicon.

4.5.1.2 RANDOM AND CONSTRAINED-RANDOM TESTS

In addition to focused tests, exercise systems feature through random and constrained-random testing. Examples of such tests include executing a random sequence of system instructions and exercising concurrent interleaving. The goal is to exercise the system in ways not conceived by humans. Random instruction tests can include hundreds of millions of random seeds generating instruction sequences.

In addition to the tests, their applications require development of specialised peripherals, boards and test cards. This is specifically pertinent for compatibility validation where the system needs to be exercised for a large number of peripheral devices, software versions and platform features.

4.6 TRACE SIGNAL SELECTION

Trace signals are used to address the observability limitation during post-silicon debug. The idea is to trace a set of signals during run time and store in a trace buffer. This is done so that traced values can be used during post-silicon debug. Since I/O speed (for example, using JTAG) is significantly slower than the speed of execution (for instance, MHz versus GHz), it is not possible to dump the traced values through I/O ports during execution. Therefore an internal trace buffer is required.

Trace signal selection needs to maintain various design constraints. For example, trace buffer size directly translates to area and power overhead. Moreover, routing selected signals to the trace buffer may cause congestion and other layout-related issues. As a result, in a design with millions of signals, a typical trace buffer traces a few hundred signals for a few thousand cycles. For example, a 128 _ 2048 trace buffer can store 128 signals over 2048 clock cycles.

CHAPTER 5

TECHNOLOGY USED

5.1 IP GPU

A Graphics Processing Unit (GPU) is a single-chip processor primarily used to manage and boost the performance of video and graphics. GPU features include: 2-D or 3-D graphics Digital output to flat panel display monitors Texture mapping Application support for high-intensity graphics software such as AutoCAD Rendering polygons Support for YUV color space Hardware overlays MPEG decoding These features are designed to lessen the work of the CPU and produce faster video and graphics. A GPU is not only used in a PC on a video card or motherboard; it is also used in mobile phones, display adapters, workstations and game consoles

5.1.1 TEST STRUCTURE

```
int main( void )
{
int c;int *dev_c;
HANDLE_ERROR( cudaMalloc( (void**)&dev_c, sizeof(int) ) ); add<<<1,1>>>( 2, 7,
dev_c );
HANDLE_ERROR( cudaMemcpy( &c, dev_c, sizeof(int),
cudaMemcpyDeviceToHost ) );
printf( "2 + 7 = %d\n", c ); cudaFree( dev_c );return 0; }
```

5.1.2 LINUX

Linux is the best-known and most-used open source operating system. As an operating
system, Linux is software that sits underneath all of the other software on a

computer, receiving requests from those programs and relaying these requests to the computer's hardware.

Testers to make sure everything works on different configurations of hardware and software, and to report the bugs when it does not. Designers to create user interfaces and graphics distributed with various programs. Writers who can create documentation, and other important text distributed with software. Translators to take programs and documentation from their native languages and make them accessible to people around the world. Packagers to take software programs and put all the parts together to make sure they run flawlessly in different distributions. Enthusiasts to spread the word about Linux and open source in general. And of course developers to write the software itself.

```
=================================================================
================================
                BUILD   SUMMARY

=================================================================
================================

Generated by build_all
Test: test_template
Path: **********
Arguments: -nc
ProcessorBuild ResultBuild Time Image Filename       Warnings

aop_procSUCCESS 00:00:05 cdvi_mannar.axf

apps_proc0SUCCESS 00:00:08 cdvi_mannar.axf

lpass_procSUCCESS 00:00:04 cdvi_mannar.elf
```

```
mss_procSUCCESS 00:00:04 cdvi_mannar.elf

turing_procSUCCESS 00:00:03 cdvi_mannar.elf

Total build time: 00:00:27
```

Table 5.1 Showing compilation results

5.2 EMULATION PLATFORM

FPGA emulation of ASICs is fast enough to run meaningful segments of system and application software, providing additional opportunities for system verification. An FPGA prototype can be available before the physical ASIC, allowing for early testing and debugging of software.

5.2 .1 TRACE 32

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

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

5.2.1.1 USE OF T32

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

First of all, install the software TRACE 32, there are different sources to download this software, this software designed in python language, so in industry for different projects their TRACE 32 is differently designed according to their project specification.

Secondly, we will connect our debug target to the system through JTAG

Then we will flash the Meta file of that project on the devices, basically we get two types of hardware for debugging first one is CDP and the second one is MDP, this MDP is the compact form of CDP.

These all the basic steps that we will do before starting any project, after this we plan according to our debugging target and for any debugging activity script is written in various languages like .cmm, tcl, perl and python. Besides, it must be expressed that the Trace32 program does not collect, compile, or connect your program. This must be done by a few other application specific for this reason.What trace32 does arrange to the user is download the executable to the target run the program debug it and the rest of the features recorded underneath the software. The software provided with the Trace32 system, gives support for all the capacities that the Trace32 system[14] orders:

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

Fig.5. T32 window

5.3 JTAG ADAPTER

Traditionally PSV followed scan and trace based methods. The scan based method reuses the JTAG test access port and the scan chain design for test (DFT) structures. A scan chain is a chain of flip flops connected back to back, allowing the input of data from the last flip flop and output of data from the last flip flop using the JTAG interface captured and then offloaded.

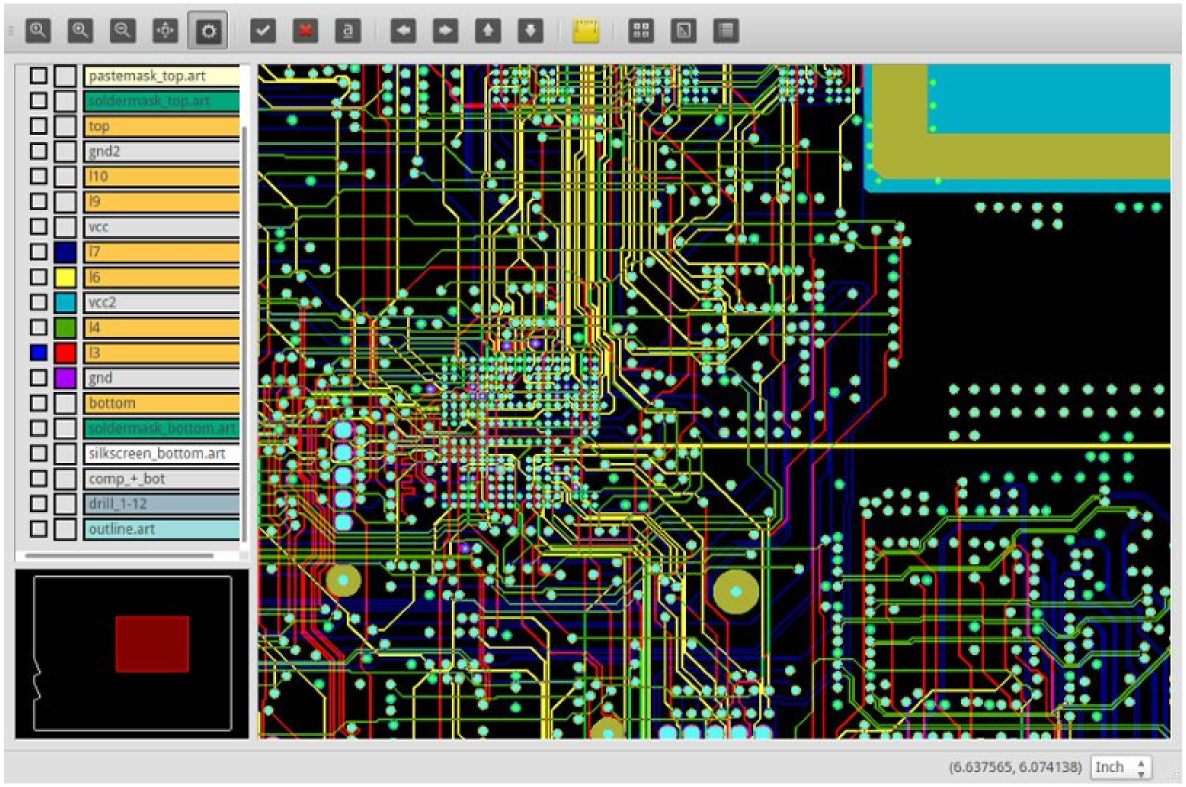


Fig. 5.2 Schematic of JTAG

The flip flops are enhanced with a multiplexer to select from two inputs, one is the normal data input of the circuit and the other is the test data input, based on a scan enable input. By asserting the scan enable signal, the test mode is selected in which the scan flip flops select the test vector input and the test vector advances through the scan chain on each clock cycle. After the proper test vector is loaded into the scan chain, it is applied to the circuit by deasserting the scan enable signal followed by a clock cycle. The resulting values of the internal state elements are he same way as loading of the test vector.

Specialized validation platforms known as hardware validation kits made for a particular SoC family, are required for functional validation. These have additional probings and connections for enhancing the controllability and observability for functional validation [5, 6]. The validation kit is connected to a normal personal computer (PC) to receive the inputs. The debug hardware consists of a JTAG controller which allows access to the internal SoC components. The software used for debugging, runs on the PC attached to the validation kit via JTAG. The PC and the validation kit are also connected to each other over the serial port. In addition, both are connected to the local network. The complete set-up for validation is shown in figure 4.1.

In PSV, the hardware prototype must run under the expected load. Thus an operating system boot has become an important validation test for a modern complex SoC [4, 7]. The Linux operating system is the most favorable choice by semiconductor companies for this purpose as it is open source, fully customizable and most probably it will be used in the end product. Linux has a low level software known as device driver which implements the different functionalities of an IP. JTAG vie is shown in figure 4.3.

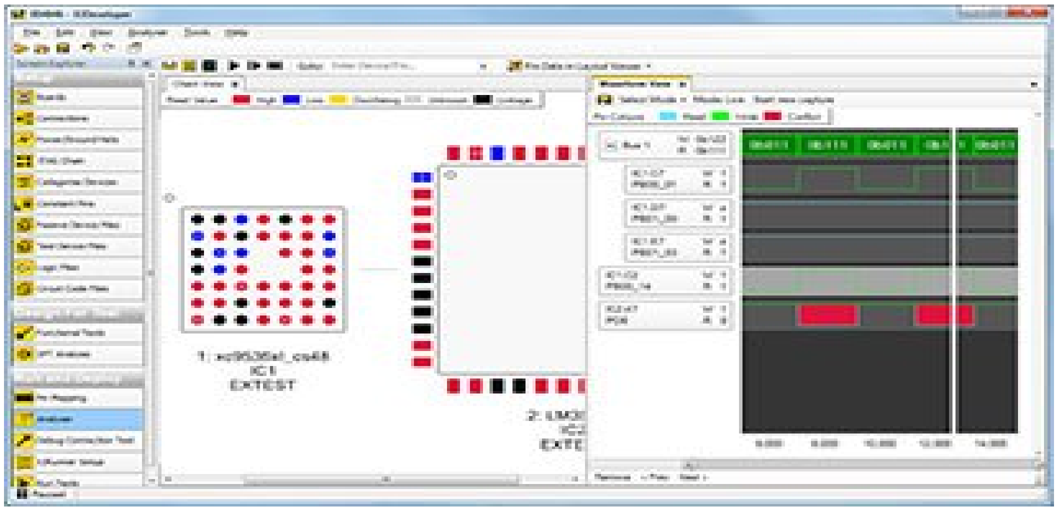
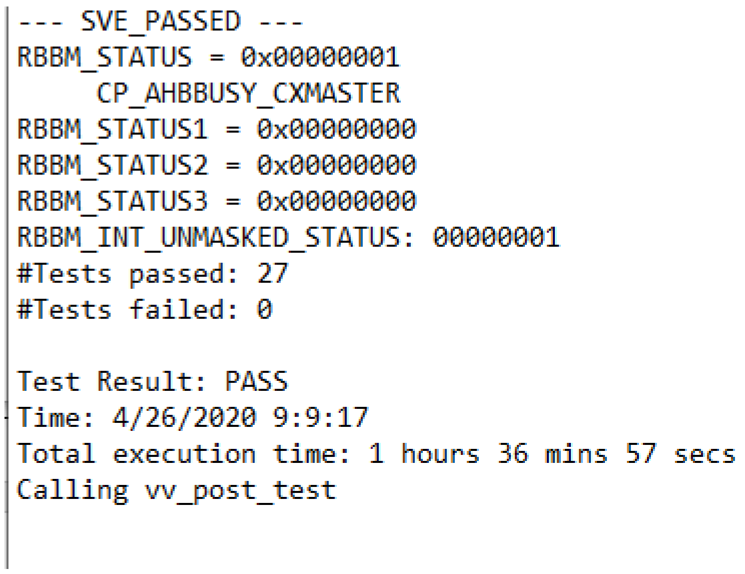


Fig.4.3: JTAG view

5.4 TEST REPORT



```
--- SVE_PASSED ---
RBBM_STATUS = 0x00000001
     CP_AHBBUSY_CXMASTER
RBBM_STATUS1 = 0x00000000
RBBM_STATUS2 = 0x00000000
RBBM_STATUS3 = 0x00000000
RBBM_INT_UNMASKED_STATUS: 00000001
#Tests passed: 27
#Tests failed: 0

Test Result: PASS
Time: 4/26/2020 9:9:17
Total execution time: 1 hours 36 mins 57 secs
Calling vv_post_test
```

Fig. GPU TEST REPORT

5.4.1 RESULT

Since SVE is passed here then it is well. Had it been Failed then Engineers have to debug the problem and report the issue if its software issue and solve the problem.

5.5 SCRIPTS

Scripts are lists of commands executed by certain programs or scripting engines. They are usually text documents with instructions written using a scripting language.

5.5.1 Cmm script

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

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

Fig. Core clock enable script

5.5.1.1 Results

It will enable all the clock of the system so that our main script will find the environment as real soc environment and we can emulate.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

According to the result genererated whether SVE is passed or failed the engineers have to take actions. If SVE is passed it is ok but if SVE is failed they have to debug and solve the issue. We have also concluded that the pre silicon and post silicon validation is done now the chips are ready for the tape out that is ready for the out to the market and now can be dispatched to the companies. As we have shown the results of some IPs, there are hundreds of IPs which is allotted to each and every person and they are working on them to improve their performance. Since company has to reduce the cost so they are allocating the multiple IPs to the single person which reduces the cost of the company.

In future the debugging can still be done by person but the software can be automated to release the test report which reduces the work load of the individual and also decreases the cost to company. Man power mental stress will reduce and they will be able to utilize it to make things more productive.

.