Subsystem Verification & Analysis of Intellectual Property of Arm® Cortex®-A65AE Architecture

A Thesis submitted in partial fulfillment of the requirement for the Award of the Degree of

MASTER OF TECHNOLOGY

in VLSI Design

Submitted By

VINAY TILWANI

602262028

Under Supervision of

Dr. Vinay Kumar

(Associate Professor, ECED, Thapar Institute of Engineering & Technology)

&

Mr. Arun Kumar

(Senior Group Manager, STMicroelectronics)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY
(A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB

JULY 2024

DECLARATION

I, **Vinay Tilwani,** declare that the work presented in this thesis entitled "**Subsystem Verification & Analysis of Intellectual Property of Arm® Cortex® -A65AE Architecture"** in partial completion of the requirement for the award of the degree of **Master of Technology (VLSI Design**) submitted at **Electronics & Communication Department**, Thapar Institute of Engineering & Technology (Deemed to be University), Patiala. The project was supervised by **Mr. ARUN KUMAR (Senior Group Manager, STMicroelectronics)** and **Dr. Vinay Kumar (Associate Professor, ECED, Thapar Institute of Engineering & Technology) at STMicroelectronics.** The matter presented in this has not been submitted in part or full to any other university or institute for the award of any other degree.

Date:19th of July, 2024 (Vinay Tilwani)

Place: TU, Patiala Roll No.:(602262028)

This is to certify that the above statement made by the student is correct to the best of my knowledge and belief.

Date:- 19th of July,2024

Dr. Vinay Kumar
Associate Professor,
Department of Electronics and Communication Engineering
Thapar Institute of Engineering & Technology
Patiala, Punjab

Date:- 19th of July,2024

Mr. Arun Kumar
Industrial Coordinator
Senior Group Manager
STMicroelectronics

CERTIFICATE

STMicroelectronics Private Ltd.
Plot No.1, Knowledge Park-III,
Greater Noida – 201 308.
Uttar Pradesh, India
Tel : +91-120-4003000
Fax :+91-120-4003007
Email : infostm.india@st.com
CIN : U32109DL1990FTC039906
www.st.com

31 May 2024

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Vinay TILWANI** has undergone internship in our **ADG** Group from **Jun 02,2023 to May 31,2024**. He has successfully completed his project on:**AU1E, SR6G3.**

During his internship period, he was found to be sincere and professional in his conduct.

We wish him all the best in his future endeavors.

for **STMicroelectronics Pvt. Ltd.**

Parminder Singh WALIA
India Talent Acquisition Shead

ACKNOWLEDGEMENT

The Task could not have been completed without acknowledging those who guided and supported us continuously to make our efforts successful. Taking this opportunity, I express my deepest gratitude and respect to my supervisor, **Dr. Vinay Kumar**, Associate Professor, Department of Electronics and Communication Engineering, Thapar Institute of Engineering & Technology for his guidance and encouragement throughout this project.

I deeply thank **Mr. Arun Kumar** (Senior Group Manager, STMicroelectronics) for his useful advice, suggestions, and unwavering support throughout the project. I express my deepest gratitude to **Mr. Rajesh Jeswani,** Head of the verification team, ADG Department, STMicroelectronics for his valuable support. Many thanks to my mentors for their valuable guidance during the project, who provided me with useful advice and support that helped me to understand the technical deeply thank Mr. Arun Kumar (Senior Group Manager, STMicroelectronics) for his useful advice, suggestions, and aspects of the project. I would also like to thank **Dr. Kulbir Singh,** Head of Department, Electronics & Communication Engineering, Thapar Institute of Engineering & Technology for giving me such an opportunity to intern at STMicroelectronics Pvt. Ltd and provide all kinds of support throughout the project. Finally, I thank my family, friends, andcolleagues for their timely help and valuable suggestions.

Vinay Tilwani

ABSTRACT

The development of autonomous vehicles is accelerating rapidly, creating significant challenges for advancing integrated circuits within the automotive sector. One primary challenge is ensuring devices' functional safety without relying on backup or redundant circuits. Verification involves comparing a product's design against its specifications to ensure that the design and the produced items meet customer requirements and can be reliably delivered. Errors, which occur randomly and can originate from various modules on the SoC, must be meticulously managed to prevent potentially life-threatening situations. This study presents the IP alongside its verification process and coverage analysis.

Verification teams face obstacles in verifying bus performance due to increasing design complexity and constrained development timelines. They need faster execution times, straightforward test bench creation, and efficient test pattern generation. Renesas encountered similar challenges in assessing bus performance in their chips, particularly in latency and bandwidth over time. To address these issues, Renesas utilized Cadence tools and implemented operational flows integrating AVIP and ATP, employing Palladium to reduce prolonged emulation times.

The Arm Cortex A65AE Architecture is a power-efficient, mid-range, throughput-oriented, Simultaneously Multithreaded (SMT) architecture that implements the AArch64 execution state of the Armv8-A architecture within the Dynamic IQ Shared Unit (DSU) cluster. It does not implement the AArch32 execution state of the Armv8-A architecture. As an SMT core, the Cortex A65AE supports two execution threads per core, with each thread functioning as a separate architectural Processing Element (PE) and maintaining a complete copy of the architectural state. This simultaneous multithreading allows the Cortex A65AE to issue and execute instructions from both threads concurrently within the same cycle. Software perceives a thread on the Cortex A65AE, in the same way, it views a core in a traditional single-threaded, multi-core processor, enabling existing software for single-core and multi-core systems to run unmodified on the Cortex A65AE Architecture.

TABLE OF CONTENTS

List of Figures

LIST OF ABBREVIATIONS

- IP: Intellectual Property
- CPU: Central Processing Unit
- SoC: System on Chip
- IC: Integrated Circuit
- DV: Design Verification
- RTL: Register Transfer Logic
- HDL: Hardware Description Language
- AMBA: Advanced Microprocessor bus architecture
- SV: System Verilog
- OOPS: Object-oriented programming
- DUT: Design Under Test
- UVM: Universal Verification Methodology
- ADAS: Advanced Driver Assistance System
- ECU: Electronic controller units
- TCM: Tightly coupled memory.
- FIFO: First in First Out
- RAL: Register Abstraction layer
- VIP: Verification IP
- XML: Extensible Markup Language

Chapter 1
INTRODUCTION

The design of a SoC is extremely complicated and producing a bug-free design is a challenging undertaking in and of itself. As a result, defects are detected during the early stages of semiconductor design verifications. Verification is a procedure in which we verify the design's functional correctness about the specified specifications. It determines whether the suggested design is functionally sound.

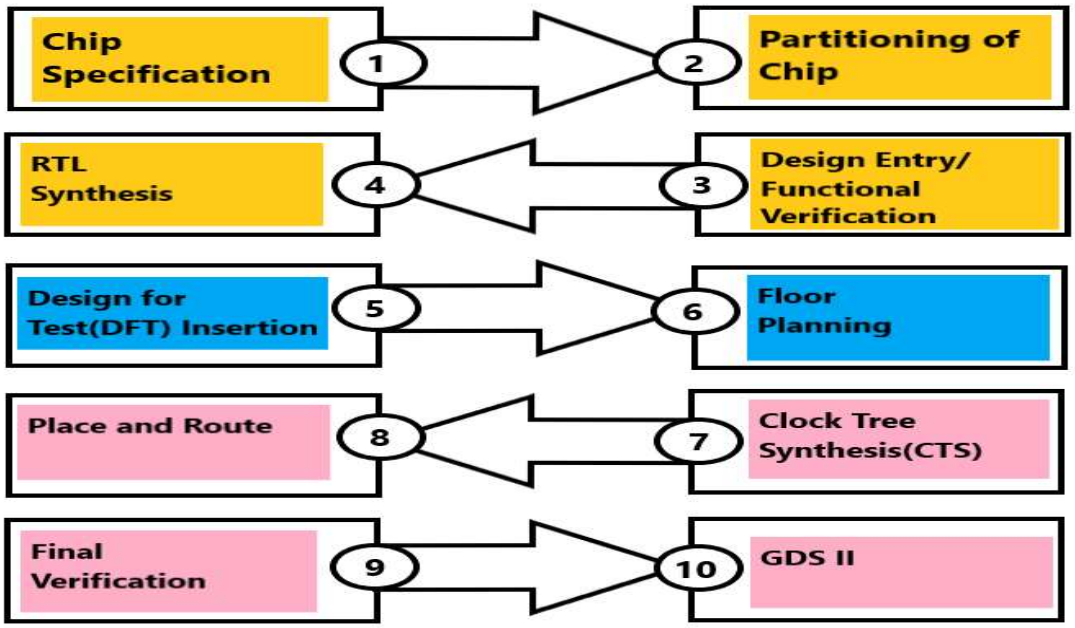

Figure 1.1 Design Cycle

Figure 1.1 shows various steps involved in the design cycle including verification. At the entry level, functional verification confirms the design's functioning and behavior. At this point, the RTL and verification teams enter the picture, with verification engineers attempting to check the validity of the RTL code using test benches and coverage metrics. Expression, block, and toggle coverage are all included in the coverage. Using code, strive to attain a coverage of greater than 97 percent.

1.1 Intellectual Property (IP)

Intellectual property, or IP, is the fundamental building block of a SOC. IP is a unit of semiconductorsthat can be utilized to license various manufacturers in different sorts of chips. Many functions are combined into a single chip in IC design. Pre-designed IP blocks play a vital part in this type of sophisticated architecture. Most SoCs have standard CPUs and functionality, allowing IPs to be reused across multiple architectures. Soft IP cores and Hard IP coresare the two types of IP cores that are now

available. The HDL languages are used to create these cores. Soft cores can be altered in the backend flow, however hard IP cores cannot be customized for different types of technology. Ethernet MAC IP and AMBA bus protocol IP are examples ofSoft IP cores.

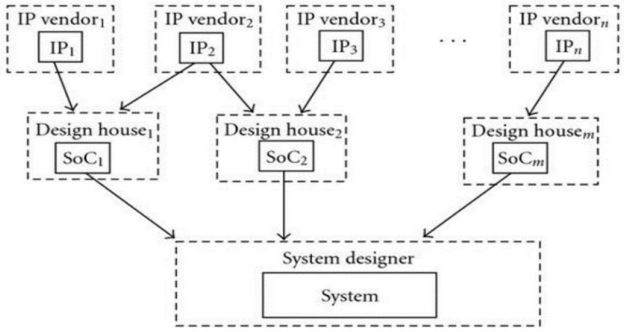

Figure 1.2 Reusing of IP in SOC.

Source of Above Image:- https://www.researchgate.net/figure/a-Components-on-a-system-on-chip-b-IP-reuse-in-SoC-environment_fig2_220174590

Figure 1.2 shows how IPs are reused in complex SoCs. The demand for IP verification has arisen because of increased training in IP design. It is like a design patent. It comes with preprogrammed functional blocks that may be directly inserted into the testbench for design verification.

1.2 Verification and Levels of Verification

Verification is a critical difficulty for organizations that build IP and SoC products to meet client needs. The complexity of modern standards poses several verification issues that can only be overcome by employing advanced verification techniques and optimizing reuse. During the verification stages of chip design, over 70% of the time is spent. All flaws and errors must be fixed before moving on to the costly manufacturing process.

Levels of verification are:

1.2.1 IP Level:

The RTL code of IPs is designed in a generic manner that incorporates parameters, allowing the IP's functionality to be directly modified using these parameters. The primary objective of the IP verification engineer is to validate every function of the IP and confirm the accuracy of the IPdesign.

1.2.2 Subsystem Level:

The main objective at this stage is to verify all essential subsystems before their integration into the primary SOC. A subsystem refers to a platform comprising a processor and bus masters, and the verification process focuses on confirming the interactions between the master and slave components.

1.2.3 SoC Level:

All non-critical or peripheral IPs are integrated into one single code SoC RTL, and thisRTL code is verified by SoC verification engineers.

The demand for IP verification has arisen because of increased training in IP design. It is like adesign patent. It comes with pre-programmed functional blocks that may be directly inserted into the testbench for design verification.

1.3 Types of IP Level Verification

1.3.1 Direct Testing Verification:

In this type of verification, the specification list named the Block Guide (BG) of the IP is read and according to the Bock Guide, and the verification plan is made. And with the help of a verification plan targeted test cases have been developed to check the functionality of the IP. This ensures steady growth to 100% coverage as shown in Fig 1.3.

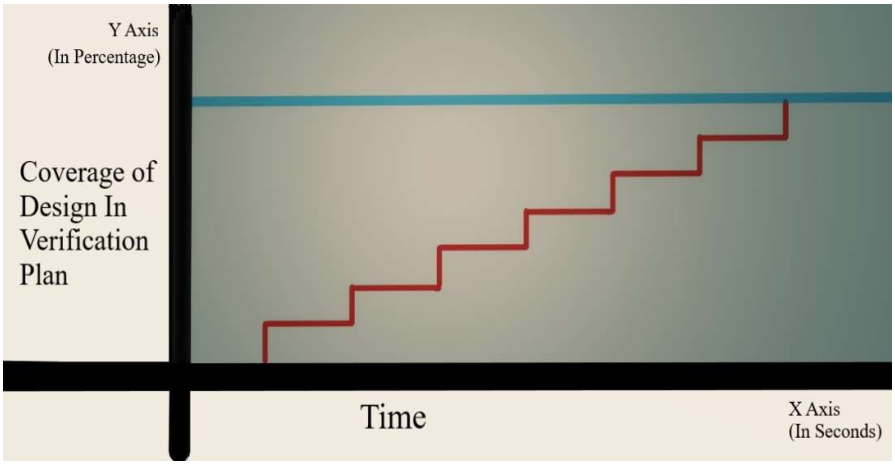

Figure 1.3 Coverage vs time (direct stimuli verification)

1.3.2 Constrained Random Stimulus Verification:

In this randomization technique is used. It is a technique by which test cases take differentvalues. This is done using the random seed values generated randomly using the simulator. According to the seed value various variables are set for the test. The same test will run using different values.

1.3.3 Formal Verification:

In this form of verification, tools generate the input, and the outputs are validated against assertions written in SVA, a subset of System Verilog. However, formal verification has some limitations. As IP complexity increases, assertion-based verification struggles to keep up, and debugging assertions in complex systems can be challenging.

1.4 Verification Cycle

Verification cycles generally involve four phases:

Development: In development, there are verification plans, architecture, testbench, and sequence development.

Simulation: compilation, elaboration, and waveform generation come in this stage.Debugging: at the transaction level, signal level, etc.

Coverage: functional, code, and SVA coverage comes at this stage that feedback to the first stage.

Figure 1.4 shows the verification cycle Every Phase plays an important role in reducing time and developing and improving it. For increasing throughput and accuracy reduction of time is important.

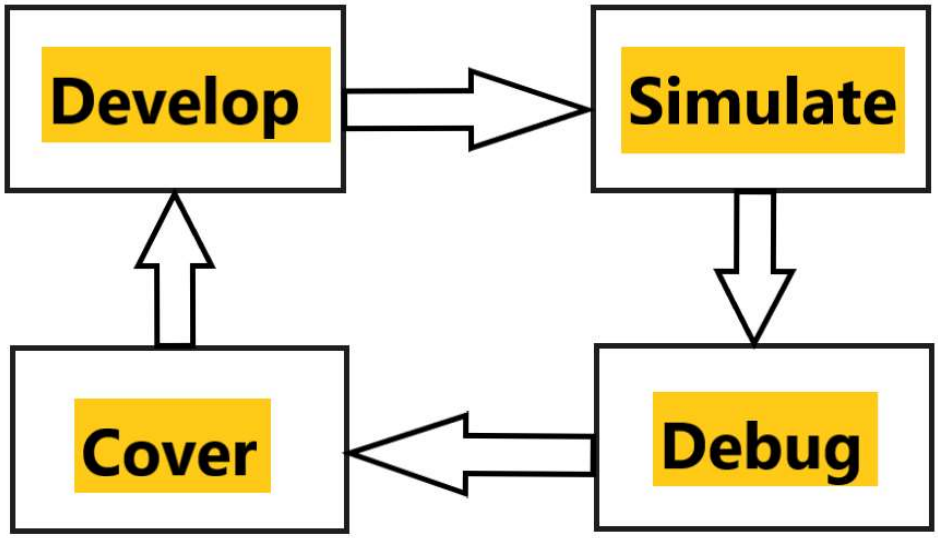

Figure 1.4 Design Verification Loop

1.5 Verification Challenges

Due to increasing complexity and market demands additional pressure comes from certifying engineers to complete complex projects in a short period. Several challenges to be addressed to deal with the growing difficulty of completing a project. These challenges are:

1.6 Production

verification production is defined as the capability to manage complex projects in a short period. These production advantages for design engineers have been made available by transistor-level, gate-level, RT, and system-level design techniques. The same type of production benefits should be provided by certification engineers to allow them to deal with ever-increasing devices. Such Product Benefits are achieved by delivering high output rates both in terms of verification resources and validated block blocks.

1.7 Efficiency

Effectiveness of determining how much human involvement is necessary to do the verification process. With the increasing complexity of machinery, it is advisable to reduce manual intervention or manipulation to as little as possible. The effectiveness of authentication increases automatically with the verification and establishment of paid authentication services, which through appropriate authentication, leads to a reduction in manual intervention.

1.8 Reusability

The ability to reuse an existing verification environment, or parts of an existing verification environment, for brand-new projects or further iterations of an identical assignment is referred to as verification reusability. By creating a modular structure for the verification environment and defining modules as parts that can be reused in different activities, reusability is addressed. Better verification architecture documentation, as well as the usage of programming and code maintenance techniques, may result in further benefits.

1.9 Completeness

The goal of verification completeness is to encompass as much design functionality as is practical. Gaining more time to focus on strengthening verification completeness will be made possible by improving verification productivity, efficiency, and reuse. It is also possible to increase completeness by adopting verification approaches that focus on this issue and tools that increase the transparency of the verification process.

1.6 Description of Cortex-A65AE Architecture

1.6.1 ARM CORTEX A65AE

STMicroelectronics is utilizing the ARM Cortex A65AE core in one of their projects. This specialized version of the Cortex A65 is tailored for the automotive industry, with 'AE' standing for Automotive Enabled. It incorporates additional safety features not present in the standard A65 core. The Cortex A65 is a high-throughput, multithreaded core designed for high-performance applications, based on the ARMv8-A architecture, and it supports a Split-Lock mode of operation. Unlike the R series cores, which operate solely in Lockstep mode, the Cortex A65AE can operate in three modes: Lock, Split, and Hybrid. In Split mode, each core within a pair functions independently. In Lock mode, one core served as a redundant copy of the primary core. In Hybrid mode, the cores work independently as in Split mode, while the external shared unit of the cluster runs in lockstep, like Lock mode.

1.6.2 About the Cortex A65AE Architecture

To achieve a balance between power efficiency and performance, ARM has developed a new architecture called Dynamic IQ, which is an enhancement of the traditional Big-LITTLE architecture [14]. In the Big-LITTLE architecture, processing units are typically organized into two separate clusters: one consisting of high-performance "big" cores and the other comprising power-efficient "little" cores. Each cluster has its own L2 cache and can operate at different frequencies. The high-performance "big" cores are designed for demanding tasks, while the "little" cores prioritize power efficiency. An L3 cache is shared between both clusters. This architecture is commonly used in application cores for smartphones and computers. Resource-intensive tasks such as gaming are allocated to the "big" cores, whereas routine activities like web browsing and typing are handled by the "little" cores. Table 3.1 outlines the differences between the "big" and "little" cores.

Table 1.6.1: Differences between Big and Little Cores

Big Cores	Little Cores
High Performance	High Efficiency
Long Pipelines	Short Pipelines
Out of order execution	In order execution
Register renaming	No renaming
Fewer configuration options	More configurability

For automotive applications, ARM introduced an enhanced architecture named Dynamic IQ to advance the capabilities of the Big-LITTLE framework. In this new architecture, both "big" and "little" cores are integrated within the same cluster. Unlike the traditional Big-LITTLE design, each core in the Dynamic IQ architecture is equipped with its own L1 and L2 caches. The L3 cache, however, is shared among all the cores within the cluster. Additionally, Dynamic IQ allows each core to operate at its clock frequency, as opposed to the Big-LITTLE architecture where the entire cluster runs at a uniform clock frequency.

1.6.3 ARM CORTEX R52

ARM Cortex-R is a family of 64-bit and 32-bit processor cores optimized for safety-critical and hard real-time applications [15]. These are high-performance cores. They are like the A series but with additional features that make them more tolerant of faults and suitable for safety-critical applications. The Cortex-R is suitable for use in computer-controlled systems where very low latency and a high level of safety are required. An example of a hard real-time, safety-critical application would be a modern electronic braking system in an automobile. The system not only needs to be fast and responsive to a plethora of sensor data input but is also responsible for human safety. The failure of such a system could lead to severe injury or loss of life. Cortex R52 is the processor core used by STMicroelectronics on most projects for real-time operation.

1.6.3.1 Some of the safety features included in Cortex R52 are:

• Tightly coupled memory (un-cached memory with guaranteed fast access time)
• Increased exception handling in hardware
• Hardware division instructions
• Memory protection unit (MPU)
• Deterministic interrupt handling as well as fast non-maskable interrupts
• ECC on L1 cache and buses
• Dual-core lockstep for CPU fault tolerance

1.6.4 The Cortex-A65 Architecture supports

- ❖ The Armv8.2-A extension.
- ❖ The extension for Reliability, Availability, and Serviceability (RAS).
- ❖ The Armv8.3-A extension introduced Load Acquire (LDAPR) instructions.
- ❖ Dot Product support instructions were added in the Armv8.4-A extension.

- The Armv8.5-A extension introduced the PSTATE Speculative Store Bypass Safe (SSBS) bit, which provides software mitigation for Spectre Variant 4.

The Cortex A65 architecture does not support the AArch32 execution state of the Armv8-A architecture. As a Simultaneous Multithreading (SMT) core, the Cortex A65 architecture allows each core to support two execution threads. Each thread acts as an independent architectural Processing Element (PE) with a complete copy of the architectural state. This simultaneous multithreading capability enables the Cortex A65 architecture to issue and execute instructions from both threads concurrently within the same cycle. Software perceives a thread on the Cortex A65 architecture similarly to how it views a core in a traditional, single-threaded, multi-core processor. Consequently, existing software for single-core and multi-core systems can run without modification on the Cortex A65 architecture.

In a DSU cluster, each Cortex A65 core includes a dedicated Level 1 (L1) instruction cache, a dedicated L1 data cache, and an optional private Level 2 (L2) cache per core.

1.6.4.1 Features: -

The Cortex A65 architecture is equipped with several notable features:

1.6.4.2 Core features:

- Implements the Armv8.2-A A64 instruction set in its entirety.
- Supports AArch64 execution state at every exception level, from EL0 to EL3.
- Utilizes a superscalar, variable-length, out-of-order pipeline design.
- Capable of simultaneous multithreading, allowing two execution threads per core.
- Contains a Memory Management Unit (MMU).
- Compatible with Arm Trust Zone® technology.
- Supports a 44-bit physical address space.
- Offers an optional execution unit for Advanced SIMD and floating-point operations.
- Includes an optional Cryptographic Extension, provided the Advanced SIMD and floating-point execution unit is present.
- Equipped with a Generic Interrupt Controller (GIC) CPU interface for integration with an external distributor.
- Features a Generic Timer interface that supports a 64-bit count input from an external system counter.

1.6.4.3 Cache features

- Separate L1 caches dedicated to data and instructions.
- An optional, unified private L2 cache.
- Error Correcting Code (ECC) or parity cache protection is applied to all RAM instances for both L1 and L2 caches.

1.6.4.4 Debug features

- Extension for Reliability, Availability, and Serviceability (RAS).
- Debug logic based on Armv8.2-A architecture.
- Performance Monitor Unit (PMU) for performance tracking.
- Embedded Trace Macro-cell (ETM) that supports tracing of instructions exclusively.

1.6.4.5 Implementation options

The Cortex A65 architecture provides extensive flexibility. Options available at build-time allow the architecture to be tailored to meet functional needs while optimizing for minimal area and power usage. Each core is identically configured at build-time in configurations with multiple cores, except for the possible inclusion and size of the L2 cache.

The following table outlines the various implementation options for a single core.

***Note: The implemented features serve as shared resources for both threads. The chosen cache options are dynamically allocated based on usage.**

Table: 1.6.2 Implementation Options for an Architecture

Feature	Range of options	Notes
L1 instruction cache size	• 32KB • 64KB	-
L1 data cache size	• 32KB • 64KB	-
L2 cache	• Included • Not included	-
L2 cache size	• 64KB • 128KB • 256KB	-
Advanced SIMD and floating-point support	• Included • Not included	There is no option to implement floating-point without Advanced SIMD.
Cryptographic Extension	• Included • Not included	Requires Advanced SIMD and floating-point support.
CoreSight *Embedded Logic Analyzer* (ELA)	• Optional support	Support for integrating CoreSight ELA-500. CoreSight is a range of debug components available as a separately licensable product.
Dot Product support instructions	• Included	Support for the Armv8.4-A `SDOT` and `UDOT` instructions. Requires Advanced SIMD and floating-point support.

1.6.4.6 Supported standards and specifications: -

The Cortex A65 architecture is based on the Armv8-A architecture and includes several extensions. It supports a range of standards for interconnects, interrupts, timers, debugging, and tracing.

Table: 1.6.3 Compliance with standards and specifications

Architecture specification or standard	Version	Notes
Arm architecture	Armv8-A	• AArch64 Execution state only • All Exception levels, EL0-EL3 • A64 instruction set
Arm architecture extensions	• Armv8.1-A extensions • Armv8.2-A extensions • Advanced SIMD and floating-point support • Cryptographic Extension • RAS Extension • Armv8.3-A LDAPR instructions • Armv8.4-A Dot Product support instructions • Armv8.5-A extensions	• You cannot implement floating-point without Advanced SIMD. • You cannot implement the Cryptographic Extension without the Advanced SIMD and floating-point support. • The Cortex-A65 core implements the LDAPR instructions introduced in the Armv8.3-A extensions. • From the Armv8.4-A extensions, the Cortex-A65 core only implements the UDOT and SDOTinstructions. • The Cortex-A65 core implements the PSTATE SSBS (Speculative Store Bypass Safe) bit that supports software mitigation for Spectre Variant 4 introduced in the Armv8.5-A extension.
Generic Interrupt Controller	GICv4	Backwards compatible with GICv3
Performance Monitor Unit	PMUv3	-
Debug	Armv8-A	With support for the debug features added by the Armv8.2-A extensions
CoreSight	CoreSightv3	-
Embedded Trace Macrocell	ETMv4.2	• Instruction trace only. Data trace is not supported. • System register access to the ETM is not supported.

1.6.4.7 Test features

The Cortex A65 core includes test signals that support Automatic Test Pattern Generation (ATPG) and Memory Built-In Self-Test (MBIST) for evaluating the core's logic and memory arrays.

1.6.4.8 Design tasks

The Cortex A65 architecture is provided as a synthesizable Register Transfer Level (RTL) description in Verilog HDL. To utilize the Cortex A65 architecture, it must be implemented, integrated, and programmed. Different parties can perform these tasks, making implementation and integration decisions that influence the core's behavior and features.

1.6.4.9 Implementation

The implementer is responsible for configuring and synthesizing the RTL to create a hard macro-cell, which involves integrating RAMs into the design.

1.6.4.10 Integration

The integrator's role is to connect the macro-cell to a system-on-chip (SoC), which includes linking it to the memory system and peripherals.

1.6.4.11 Programming

In the final stage, the system programmer creates the software needed to configure and initialize the core, as well as to test the application software.

1.6.5 Factors Influencing the Operation of the Final Device

1.6.5.1 Build configuration.

The implementer decides on options that determine how the RTL source files are pre-processed. These options generally include or exclude logic, affecting aspects such as area, maximum frequency, and features of the resulting macro-cell.

1.6.5.2 Configuration inputs

The integrator configures certain features of the core by setting specific values to inputs. These settings influence the initial behavior of the core before any software configuration and may also limit the options available to the software.

1.6.5.3 Software configuration

The programmer sets up the core by writing values into registers, which affects how the core operates.

1.7 Cache Memory Using Verilog HDL

The Verilog Hardware Description Language is used for designing cache memory, incorporating both direct-mapped and set-associative cache structures. The set-associative cache designs feature two-way, four-way, and eight-way setups. In this architecture, a controller unit is responsible for adjusting the mapping technique. Additionally, power optimization is achieved by disabling unused cache memory blocks, enhancing access speed.

1.7.1 Basic Cache Memory

In modern computers, the speed of the memory is crucial and highly sought after. Cache memory significantly enhances processing speed. Figure 3.1 illustrates the block diagram of Cache Memory, positioned between the processor and main memory. It plays a pivotal role in enhancing CPU performance.

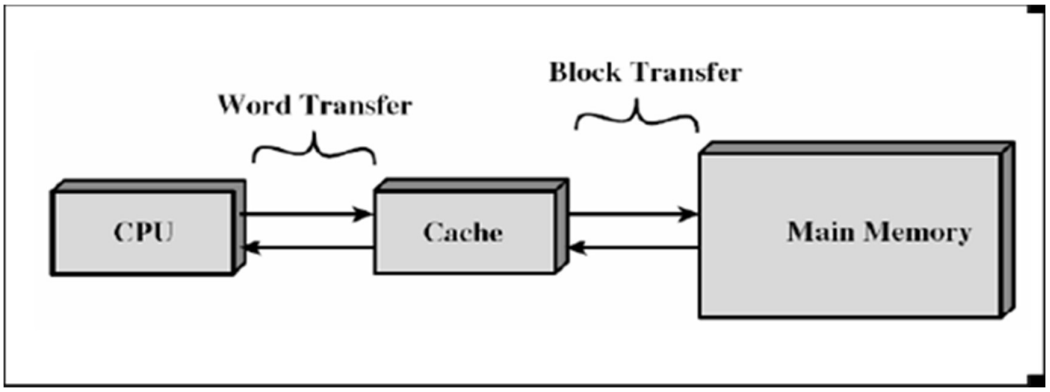

Figure 1.7.1 shows a block diagram of basic cache memory

Source of Above Image:- https://www.researchgate.net/figure/Position-of-cache_fig1_319523795 [accessed 12 July 2024]

To address the speed difference between the CPU and main memory, cache memory is implemented. This is crucial because the CPU processes data much faster than the main memory can transfer. The principle of locality of reference involves preloading data into the cache, enabling quicker data access and retrieval.

Different mapping techniques are implemented to boost CPU performance by ensuring efficient cache data access. In set-associative mapping, a cache set selection bit is used to access specific cache memory sets. This design employs a random cache replacement

policy, where, upon a cache miss, the corresponding cache line is replaced with data from the new address block in the main memory.

1.7.2 Main Memory Design

To interface a 512 MB main memory with a cache memory using a block size of 64 bytes:

- The total number of lines in the main memory is calculated by dividing the main memory size by the block size, resulting in 8,388,608 lines.
- Addressing main memory lines from 0 to 8,388,607 requires 23 bits for the main memory index, corresponding to CPU address bits [28:6].
- The remaining 6 address bits [5:0] function as offset bits.

This chapter provides an in-depth look at the cache memory architecture, utilizing various mapping techniques managed by a cache way controller unit. Section I covers the fundamental operations of cache memory. Section II discusses different cache mapping techniques. Section III explains the implementation of the cache way controller unit.

1.7.3 CACHE MAPPING TECHNIQUES

Cache mapping techniques encompass various strategies to associate main memory with cache memory:

A. Direct mapping
B. Two-way mapping
C. Four-way mapping
D. Eight-way mapping.

A. Direct Mapping: -

In the direct mapping technique, mapping 512 MB of main memory with a block size of 64 bytes to a 512 KB cache memory requires 29 bits from the CPU address.

Address 29 bits

A[28,19]	A[18,6]	A[5,0]
Tag	**Cache Index**	**Offset**

Figure 3.2 illustrates the number of address bits used in direct cache memory addressing

The diagram in Fig. 3.2 illustrates the address configuration used in Direct Mapped Cache Memory. The offset bits [5:0] specify the byte address within each cache line, while the

index bits [18:6] identify the cache line itself. The tag and cache index collectively represent the corresponding main memory address.

This diagram outlines the structure of a direct-mapped cache with a capacity of 512 KB and a line size of 64 bytes. A decoder selects each cache line, which stores the address associated with its tag. Upon a cache hit, the valid bit is activated, allowing the CPU to access data from the cache line at the specified byte offset.

During a read operation, the CPU compares its address bits [28:19] with the cache line's tag bits. If there is a match, indicating a cache hit, the valid bit remains set, and the requested data is promptly retrieved from the cache. In the event of a cache miss, where no match is found, the valid bit is reset, the dirty bit may be set, and the CPU retrieves the required data directly from the main memory. Concurrently, the cache updates its tag bits and stores the fetched data block from main memory into the appropriate cache line.

B. Two-way mapping

In two-way mapping, 512 Kbyte Cache memory is divided into two equal sizes of 256 Kbyte each memory. It is like direct mapping with two cache memory associative sets.

Address 29 bits

A[28,18]	A[17,6]	A[5,0]
Tag	**Set Index**	**Offset**

Figure 3.3 Details the address bit configuration for two-way cache memory

The address bits for the two-way cache memory are illustrated in Figure 3.4. The main memory can be assigned to either of the two cache sets based on the cache set selection bit from the CPU address. Like direct mapping, data is written to one of the two cache sets according to the cache selection bit. In this method, the tag bits of the cache lines are fewer compared to direct mapping, as the cache is split into two equal parts of 256 Kbytes each.

The tag bits are compared with address bits [28:18]; if they match, it results in a hit; otherwise, it is a miss, like direct mapping. In this case, only one cache set of 256 Kbytes is enabled for each CPU address, reducing the search time compared to direct mapping due to the smaller cache set size.

C. Four-way mapping

In four-way mapping, the 512 Kbyte cache memory is divided into four equal cache sets, each 128 Kbytes size. The address bits for the four-way cache memory are shown in Figure 7 and follow a

similar structure to two-way mapping but with four sets instead of two. Each cache set is selected using the CPU address bits [18:17].

Address 29 bits

A[28,17]	A[16,6]	A[5,0]
Tag	**Set Index**	**Offset**

Figure 3.4 Displays the address bit setup for four-way cache memory

The read-and-write operations function similarly to those in two-way mapping. If the CPU address bits [28:17] match the stored cache tag bits, a hit occurs and data is read from the cache memory. If there is a miss, data is retrieved from the main memory, and the cache memory is updated with the missed memory block simultaneously.

Here, in this configuration, only one 128 Kbyte cache set is enabled for each CPU address, resulting in faster search speeds compared to two-way mapping due to the smaller cache set size.

D. Eight Mapping

In eight-way mapping, the 512 Kbyte cache memory is divided into eight equal cache sets, each 64 Kbytes in size. The address bits for the eight-way cache memory are illustrated in Figure 3.5.

Address 29 bits

A[28,16]	A[15,6]	A[5,0]
Tag	**Set Index**	**Offset**

Figure 3.5 Describes the address bit allocation for eight-way cache memory

Here, in this setup, only one 64 Kbyte cache set is enabled for each CPU address, resulting in faster search speeds compared to two- and four-way mapping due to the smaller cache set size.

1.7.4 IMPLEMENTATION

It depicts the cache way controller unit. This unit contains two control bits that allow for the adjustment of mapping techniques based on the designer's requirements.

- When the way control bit is set to "00," the cache operates as direct mapping, utilizing the entire 512 Kbyte capacity and functioning according to direct mapping principles.
- If the way control bit is set to "01," the cache functions as two-way mapping. The CPU can access any one of the two 256 Kbyte cache sets based on the CPU address bit.
- I am Setting the way control bit to "10" which results in the cache operating as four-way mapping. The CPU can access any one of the four 128 Kbyte cache sets based on CPU address bits [18:17].
- When the way control bit is set to "11," the cache operates as eight-way mapping. The CPU can access any one of the eight 64 Kbyte cache sets based on CPU address bits [18:16].

Chapter 2
Literature Review

1.] S. Venkataraman et al.

This paper presents a methodology for verifying the performance of the ARM CORTEX®-A65 CORE using System Verilog Assertions (SVA). The authors developed a suite of SVA templates specifically tailored for assessing NIC400 performance metrics such as throughput, latency, and power consumption. Applied to a complex SoC design, this methodology effectively identified performance issues through SVA-based verification.

Focused on the ARM CORTEX®-A65 ARCHITECTURE, a critical Network Interconnect Component (NIC) within ARM SoCs, the study utilizes System Verilog assertions to enhance verification efficiency. The authors demonstrate that these assertions facilitate early error detection in design, thereby reducing verification cycles and improving overall design quality. Such verification is essential to ensure that the CORTEX®-A65 ARCHITECTURE meets stringent performance criteria and operates reliably.

Employing System Verilog assertions, the authors verified the NIC-400's performance by specifying crucial design properties. These assertions were rigorously evaluated during simulation to pinpoint potential design flaws. The verification approach combined directed and random testing strategies, underscoring the effectiveness of System Verilog assertions in verifying the CORTEX®-A65 ARCHITECTURE's correctness and performance.

In conclusion, the research highlights how System Verilog assertions significantly enhance the verification process for complex designs like the CORTEX®-A65 ARCHITECTURE. The authors advocate for the widespread adoption of System Verilog assertions to streamline verification efforts and improve overall design robustness and performance.

The research emphasizes the effectiveness of System Verilog assertions in optimizing the verification process for the CORTEX®-A65 ARCHITECTURE. It highlights how these assertions are pivotal in detecting errors early, thus reducing verification timelines and enhancing overall design quality. Furthermore, the authors demonstrate that System Verilog assertions play a critical role, in augmenting verification coverage and ensuring thorough testing of essential scenarios.

In conclusion, the study underscores the substantial advantages of integrating System Verilog assertions into similar design verification endeavors. The authors strongly recommend adopting System Verilog assertions for future design verification practices.

Overall, the paper presents a detailed study of the performance verification of the ARM CORTEX® -A65 ARCHITECTURE using System Verilog assertions. The paper provides valuable insights for designers and verification engineers working on similar designs in the future.

[2.] T. Bera et al.

The ARM CORTEX® -A65 ARCHITECTURE Interconnect Fabric is a critical component of ARM-based System-on-Chip (SoC) designs. In this paper, the authors present a performance analysis of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric. They investigate the effect of various parameters, such as the number of processors, the memory bandwidth, and the cache size, on the performance of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric. The authors use simulation-based experiments to evaluate the performance of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric under different scenarios.

The ARM CORTEX® -A65 ARCHITECTURE Interconnect Fabric is a Network Interconnect Component designed to provide high-performance interconnect between processing elements in ARM-based SoCs. The NIC400 is designed to support AMBA AXI4, ACE, and CHI protocols, and it provides a few features, such as Quality of Service (QoS) and power management. The performance of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric is critical to ensure that the SoC meets the performance requirements and functions correctly.

Performance Analysis of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric: The authors perform a performance analysis of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric using simulation-based experiments. They investigate the effect of various parameters on the performance of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric. The authors vary the number of processors, the memory bandwidth, and the cache size to evaluate the performance of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric under different scenarios.

The authors present a detailed analysis of the performance of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric. They show that the performance of the CORTEX® -A65 ARCHITECTURE Interconnect Fabric depends on various parameters, such as the number of processors, the memory bandwidth, and the cache size. The authors demonstrate that increasing the

number of processors can improve the performance of the CORTEX® -A65 CORE Interconnect Fabric up to a certain limit. They also show that increasing the memory bandwidth and the cache size can improve the performance of the CORTEX® -A65 CORE Interconnect Fabric.

In conclusion, the authors present a performance analysis of the ARM CORTEX® -A65 CORE Interconnect Fabric.
They investigate the effect of various parameters, such as the number of 22 processors, the memory bandwidth, and the cache size, on the performance of the CORTEX® -A65 CORE Interconnect Fabric. The authors demonstrate that the performance of the CORTEX® -A65 CORE Interconnect Fabric depends on various parameters, and they provide recommendations to optimize the performance of the CORTEX® -A65 CORE Interconnect Fabric. The paper provides valuable insights for designers and verification engineers working on ARM-based SoC designs in the future.

Overall, the paper presents a comprehensive performance analysis of the ARM Core Link NIC400 Interconnect Fabric. The authors investigate the effect of various parameters on the performance of the CORTEX® -A65 CORE Interconnect Fabric and provide recommendations to optimize its performance. The paper provides valuable insights for designers and verification engineers working on ARM-based SoC designs in the future.

[3.] S. Mehta et al.

In this paper, the authors present a UVM-based verification environment to verify the performance of the CORTEX® -A65 CORE Interconnect Fabric. The verification environment is designed to perform performance verification at both the transaction and cycle-accurate levels. The authors demonstrate the effectiveness of the verification environment by verifying the performance of the CORTEX® -A65 CORE Interconnect Fabric under different scenarios.

Introduction: The ARM Core Link CORTEX® -A65 CORE Interconnect Fabric is a Network Interconnect Component designed to provide high-performance interconnect between processing elements in ARM-based SoCs. The performance of the CORTEX® -A65 CORE Interconnect Fabric is critical to ensure that the SoC meets the performance requirements and functions correctly. In this paper, the authors present a UVM-based verification environment to verify the performance of the CORTEX® -A65 CORE Interconnect Fabric. UVM-Based Verification Environment: The authors present a UVM-based verification environment to verify the performance of the CORTEX® -A65 CORE Interconnect Fabric. The verification environment is designed to perform performance verification at both the transaction and cycle-accurate levels. The verification environment is

composed of a sequence generator, a scoreboard, and a coverage collector. The sequence generator generates a stimulus to test the performance of the CORTEX® -A65 CORE Interconnect Fabric, while the scoreboard checks the correctness of the responses. The coverage collector collects coverage data to ensure that all possible scenarios have been tested.

Results: The authors demonstrate the effectiveness of the verification environment by verifying the performance of the CORTEX® -A65 CORE Interconnect Fabric under different scenarios. They show that the 23-verification environment can detect performance bottlenecks and ensure that the performance of the CORTEX® -A65 CORE Interconnect Fabric meets the design requirements. The authors also show that the verification environment can be used to validate the correctness of the QoS and power management features of the CORTEX® -A65 CORE Interconnect Fabric.

Conclusion: In conclusion, the authors present a UVM-based verification environment to verify the performance of the ARM Core Link CORTEX® -A65 CORE Interconnect Fabric. The verification environment is designed to perform performance verification at both the transaction and cycle-accurate levels. The authors demonstrate the effectiveness of the verification environment by verifying the performance of the CORTEX® -A65 CORE Interconnect Fabric under different scenarios. The paper provides valuable insights for verification engineers working on ARM-based SoC designs in the future.

Overall, the paper presents a comprehensive UVM-based verification environment to verify the performance of the ARM Core Link CORTEX® -A65 CORE Interconnect Fabric. The verification environment is designed to perform performance verification at both the transaction and cycle-accurate levels and is demonstrated to be effective in detecting performance bottlenecks and ensuring that the performance of the CORTEX® -A65 CORE Interconnect Fabric meets the design requirements. The paper provides valuable insights for verification engineers working on ARM-based SoC designs in the future.

[4.] S. Saha et al.

The ARM Core Link CORTEX® -A65 CORE Interconnect Fabric is a cache-coherent interconnect designed to provide high-performance interconnects between processing elements in ARM-based SoCs. Formal verification is an important technique used to ensure the correctness of such complex interconnect designs. In this paper, the authors present a formal verification approach to verify the correctness of the ARM Core Link CORTEX® -A65 CORE Interconnect Fabric.

Formal Verification Approach: The authors present a formal verification approach to verify the correctness of the ARM Core Link CORTEX® -A65 CORE Interconnect Fabric. The approach is based on a combination of bounded model checking (BMC) and induction-based techniques. The approach involves creating a formal model of the CORTEX® -A65 CORE Interconnect Fabric and then using the model checker to prove the correctness of the design. The authors also use induction-based techniques to check the correctness of the model by checking a set of inductive invariants.

The authors demonstrate the effectiveness of the formal verification approach by verifying the correctness of the ARM Core Link CORTEX® -A65 CORE Interconnect Fabric under different scenarios. They show that the approach can detect design errors and ensure that the interconnect design meets the correctness requirements. The authors also show that the approach can be used to verify the correctness of the QoS and power management features of the CORTEX® -A65 CORE Interconnect Fabric. In conclusion, the authors present a formal verification approach to verify the correctness of the ARM Core Link CORTEX® -A65 CORE Interconnect Fabric.

In conclusion, the authors present a formal verification approach to verify the correctness of the ARM Core Link CORTEX® -A65 CORE Interconnect Fabric. The approach is based on a combination of BMC and induction-based techniques and is demonstrated to be effective in detecting design errors and ensuring the correctness of the interconnect design. The paper provides valuable insights for verification engineers working on ARM-based SoC designs in the future.

Overall, the paper presents a comprehensive formal verification approach to verify the correctness of the ARM Core Link CORTEX® -A65 CORE Interconnect Fabric. The approach is based on a combination of BMC and induction-based techniques and is demonstrated to be effective in detecting design errors and ensuring the correctness of the interconnect design. The paper provides valuable insights for verification engineers working on ARM-based SoC designs in the future.

[5.] Morgan Kaufmann, Bruce Wile et al.

The article provides a detailed and comprehensive overview of the process of functional verification in the context of system-on-chip (SoC) design.

The primary focus of this article was to address the challenges faced by engineers in verifying the correctness and functionality of complex SoC designs. It covers the complete verification cycle, from planning and specification to coverage-driven verification closure. It starts by introducing the fundamentals of functional verification and outlines the key concepts and methodologies used in the

industry. It explains the importance of verification planning and provides insights into creating effective verification strategies.

While emphasizes the need for building reusable verification environments and discusses various verification languages and methodologies such as Verilog, VHDL, System Verilog, and Universal Verification Methodology (UVM). The author also delves into the different levels of abstraction in verification, including module-level, block-level, and system-level verification.

Coverage-driven verification, which is crucial for ensuring the thoroughness of verification efforts, is given significant attention in the book. While explaining various coverage metrics and techniques for measuring and increasing coverage, including code coverage, functional coverage, and assertion-based verification.

The book also addresses other important topics such as constrained-random stimulus generation, directed testing, and assertion-based verification. It provides practical guidance on designing and implementing test benches, writing effective test cases, and debugging verification failures. Throughout the book, Wile emphasizes the need for a systematic and disciplined approach to functional verification. He provides numerous real-world examples, case studies, and best practices to illustrate the concepts and techniques discussed.

In summary, "Comprehensive Functional Verification: The Complete Industry Cycle (Systems on Silicon)" is a comprehensive guidebook that covers all aspects of functional verification in SoC design. It serves as a valuable resource for engineers involved in verifying complex integrated circuits, offering practical insights and methodologies to improve verification efficiency and effectiveness.

[6.] D.S.A.S.S. Kolay et al.

The article explores the use of System-on-Chip (SoC) as a platform for Intellectual Property (IP) reuse, as well as the challenges and considerations associated with IP infringement and protection. The author begins by highlighting the increasing significance of IP reuse in the field of SoC design. SoC offers a platform where multiple IPs can be integrated to create complex systems, reducing development time and cost. Kolay discusses the benefits of IP reuse, including accelerated design cycles, improved functionality, and reduced risk.

The article then delves into the issue of IP infringement, which arises when unauthorized use or copying of protected IPs occurs. Kolay examines the legal and ethical aspects of IP infringement and emphasizes the importance of respecting IP rights. The author provides insights into the different types of IP infringement, such as direct copying, reverse engineering, and unauthorized modification. To address the challenges of IP infringement, the article discusses various techniques for IP protection. Kolay explores hardware-based protection mechanisms, such as encryption, watermarking, and tamper-resistant designs. Additionally, software-based protection methods, such as licensing and digital rights management (DRM), are also discussed.

The author acknowledges that while IP protection is crucial, it can sometimes hinder IP reuse and collaboration. Therefore, a balanced approach is needed to ensure both protection and efficient reuse of IPs. Kolay proposes the need for standardized IP protection schemes that provide adequate security while maintaining ease of integration and interoperability.

In summary, "SoC: A Real Platform for IP Reuse, IP Infringement, and IP Protection" discusses the role of SoC in facilitating IP reuse and the challenges associated with IP infringement and protection. The article highlights the importance of IP rights and explores various hardware and software-based protection mechanisms. It calls for a balanced approach that promotes both IP protection and efficient IP reuse in the field of SoC design. Regenerate response

[7.] C.T.G. Spear et al.

The article provides a comprehensive guide to System Verilog, a hardware description and verification language widely used in the semiconductor industry.

The article's primary focus is on the application of System Verilog for the functional verification of digital designs. It covers the language features and methodologies that enable engineers to verify the correctness and functionality of complex designs effectively.

It begins with an introduction to the basics of System Verilog, including its syntax, data types, and procedural constructs. It explains how System Verilog extends the capabilities of Verilog for verification purposes, allowing engineers to write more efficient and reusable testbenches. Spear explores various verification methodologies enabled by System Verilog, including constrained random stimulus generation, coverage-driven verification, and assertion-based verification. He explains the concepts of classes and objects in System Verilog and how they can be used to build reusable and scalable verification environments.

It also covers important topics such as functional coverage, which ensures that all aspects of the design have been exercised during verification, and code coverage, which measures the completeness of the code tested. The author provides practical examples and techniques for achieving high coverage and effective verification closure.

Furthermore, Spear discusses advanced verification topics like constrained random testing, functional coverage modeling, and debugging techniques. He explains how System Verilog can be used to create self-checking testbenches, write assertions for formal verification, and perform dynamic and static checking of design properties.
Throughout the article, the author emphasizes the importance of verification planning and methodology to ensure efficient and thorough verification. He provides guidelines for creating effective test benches and designing tests that target different aspects of the design functionality. In summary, "System Verilog for Verification" is a comprehensive guidebook that covers the features and methodologies of System Verilog for functional verification in the semiconductor industry. It serves as a valuable resource for engineers involved in verifying complex digital designs, offering practical insights, examples, and best practices for effective verification using System Verilog.

[8.] M.A.B et al.
The book provides a comprehensive guide to the technologies and methodologies used in the functional design verification of Application-Specific Integrated Circuits (ASICs) and System-on-Chip (SoC) designs.

The primary focus of the book is to address the challenges faced by engineers in verifying the correctness and functionality of complex ASIC and SoC designs. It covers a wide range of topics related to functional design verification, providing insights into the various technologies and methodologies employed in the industry.

The book starts by introducing the fundamental concepts and principles of functional design verification, including the importance of verification planning, and establishing verification goals. It provides an overview of the verification flow and discusses the different stages involved in the verification process.

M.A.B. explores the various verification technologies and methodologies used in ASIC and SoC design verification. This includes simulation-based verification, formal verification, emulation, and

hardware acceleration. The author explains the strengths and limitations of each technique and guides when and how to effectively apply them.

The book also covers important topics such as constrained-random stimulus generation, assertion-based verification, and coverage-driven verification. It delves into the techniques for creating 28 effective test benches, writing test cases and ensuring thorough coverage of the design. Furthermore, the author discusses advanced verification topics such as low-power design verification, verification of complex protocols and interfaces, and hardware-software co-verification. M.A.B. provides practical examples and case studies to illustrate the application of these methodologies in real-world scenarios. Throughout the book, M.A.B. emphasizes the need for a systematic and disciplined approach to functional design verification. The author highlights the importance of verification methodology, test planning, and collaboration among the various stakeholders involved in the verification process.

In summary, "ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies" is a comprehensive guidebook that covers the technologies and methodologies used in the functional design verification of ASICs and SoCs. It serves as a valuable resource for engineers involved in verifying complex integrated circuits, offering practical insights, techniques, and best practices to improve verification efficiency and effectiveness.

[9.] K.S.Pooja et al.

The paper focuses on the verification of interconnection intellectual property (IP) used in automobile applications, utilizing System Verilog and the Universal Verification Methodology (UVM).

The objective of the paper is to address the challenges associated with verifying interconnection IPs in the context of automobile applications, where reliability and safety are critical. The authors propose the use of System Verilog and UVM as effective methodologies for designing and verifying such IPs. The paper begins by discussing the significance of interconnection IPs in automobile applications and the need for thorough verification to ensure their proper functionality. It highlights the importance of adhering to industry standards and requirements for safety and reliability.

The authors then introduce System Verilog, a hardware description and verification language, as a suitable tool for designing and verifying interconnection IPs. They explain the key features of System Verilog, such as its object-oriented capabilities, concurrency constructs, and enhanced verification methodologies.

Additionally, the paper discusses the Universal Verification Methodology (UVM) as a standardized framework for creating reusable and scalable verification environments. The authors explain how UVM facilitates testbench development, transaction-level modeling, and coverage-driven verification. The authors present a case study on the verification of an interconnection IP for an automobile application, utilizing System Verilog and UVM. They describe the design architecture, testbench 29 implementation, and verification strategies employed. The paper highlights the use of random stimulus generation, functional coverage, and assertion-based verification techniques.

The results and observations from the verification process are presented, demonstrating the effectiveness of the proposed methodologies in identifying and resolving design issues. The authors discuss the achieved functional coverage and provide insights into the verification closure process.

In conclusion, the paper showcases the verification of an interconnection IP for automobile applications using System Verilog and UVM. It highlights the significance of reliable and safe interconnection IPs in the automotive domain and emphasizes the importance of thorough verification. The authors demonstrate the benefits of System Verilog and UVM in designing and verifying such IPs, showcasing a case study as an example of their application.

[10.] Y.Yun, J.Kim, N.Kim and B.Min et al.

The paper discusses the limitations of the Universal Verification Methodology (UVM) and proposes alternative approaches for practical System-on-Chip (SoC) verification. The authors begin by acknowledging the growing complexity of SoC designs and the need for effective verification methodologies to ensure their correct functionality. They highlight the widely adopted UVM as a popular methodology but identify certain limitations that may hinder its practical application.

The paper discusses the challenges faced in SoC verification, including testbench complexity, long simulation times, and limited observability of internal signals. The authors argue that while UVM provides a structured framework for verification, it may not adequately address these challenges. To address these limitations, the authors propose alternative verification approaches. They suggest using mixed-level modeling, which involves combining high-level models with lower-level RTL (Register Transfer Level) models, to improve the observability and controllability of the design during verification.

Furthermore, the authors advocate for the adoption of virtual platforms, which allow the simulation of the entire SoC system including the software stack. They argue that virtual platforms enable more comprehensive and efficient verification by capturing system-level behaviors and interactions. The paper also emphasizes the importance of scenario-based verification, where realistic usage scenarios and test cases are created to validate the SoC's functionality under various conditions. The 30 authors discuss the benefits of this approach in uncovering corner-case issues that may not be detected through traditional directed testing.

In addition to the proposed alternative approaches, the paper highlights the significance of verification planning, coverage analysis, and debugging methodologies in achieving thorough verification of SoCs.

In summary, "Beyond UVM for Practical SoC Verification" challenges the limitations of the UVM methodology and suggests alternative approaches for practical SoC verification. The authors advocate for mixed-level modeling, virtual platforms, and scenario-based verification to address the challenges of testbench complexity, simulation times, and limited observability. The paper underscores the importance of verification planning, coverage analysis, and effective debugging techniques in achieving comprehensive SoC verification.

[11.] S. A. Saji and K. Sivasankaran et al.

The paper focuses on the development of a test suite specifically designed for the verification of System-on-Chip (SoC) interconnects.

The authors recognize the critical role of interconnects in SoC designs as they facilitate communication between different IP blocks and subsystems. They highlight the need for thorough verification of these interconnects to ensure their proper functionality and performance. The paper discusses the challenges associated with verifying SoC interconnects, including the complex nature of the design, the presence of various communication protocols, and the potential for timing and synchronization issues.

To address these challenges, the authors propose the development of a dedicated test suite for SoC interconnect verification. They describe the methodology and techniques used to create this test suite, which aims to comprehensively test the interconnect for various scenarios and use cases. The authors present the test suite architecture, which includes a set of test cases designed to cover different aspects

of interconnect functionality. They discuss the test case selection criteria, considering factors such as protocol compliance, performance evaluation, and stress testing.

Additionally, the paper addresses the test environment setup and the hardware and software components required for running the test suite. The authors provide insights into the test execution 31 process and discuss the metrics and measurements used to evaluate the interconnect's performance and quality.

The results and observations from the test suite are presented, showcasing the effectiveness of the proposed approach in detecting interconnect-related issues, such as data corruption, latency, and synchronization problems. The authors highlight the importance of coverage analysis and the identification of corner cases to ensure thorough verification.

In conclusion, the paper highlights the development of a dedicated test suite for SoC interconnect verification. It addresses the challenges associated with verifying these complex interconnects and provides a comprehensive methodology for creating and executing test cases. The authors emphasize the importance of thorough verification to ensure the proper functionality and performance of SoC interconnects in real-world scenarios.

[12.] H. Zhaohui et al.

This paper presents a verification flow that focuses on practicality and efficiency by leveraging the reuse of IP test cases and testbenches in System-on-Chip (SoC) verification.

The authors begin by highlighting the challenges faced in SoC verification due to the increasing complexity of designs and time-to-market pressures. They emphasize the need for an efficient verification flow that allows for the reuse of IP-level verification assets, such as test cases and testbenches, to minimize effort and improve productivity.

The paper introduces the proposed verification flow, which involves the creation of reusable IP-level test cases and testbenches that can be easily integrated into the overall SoC verification process. The authors discuss the importance of standardization and modularity in designing these assets for seamless reuse across multiple projects.

The authors delve into the methodology used to develop the IP test cases, emphasizing the need for comprehensive coverage and the ability to handle different IP configurations and modes of operation.

They discuss the benefits of a structured approach to test case development, including the use of specification-driven testing and constrained-random stimulus generation.
Furthermore, the paper addresses the creation of reusable IP test benches, which serve as the infrastructure for executing the test cases. The authors highlight the importance of scalability, flexibility, and maintainability in the design of these test benches. They discuss the use of modular and parameterized testbench architectures to enable efficient reuse and flexibility.

The paper also discusses the integration of IP-level testbenches into the overall SoC verification environment. The authors present a verification flow that demonstrates how the reusable IP testbenches can be seamlessly integrated, allowing for efficient verification at the SoC level. The benefits and advantages of the proposed verification flow are highlighted, including reduced development time, improved verification coverage, and enhanced productivity through the reuse of IP-level test cases and testbenches.

In conclusion, the paper presents a practical and efficient SoC verification flow that emphasizes the reuse of IP test cases and testbenches. The authors advocate for standardization, modularity, and scalability in the development of these assets, enabling their seamless integration into the overall verification environment. The proposed flow offers benefits such as reduced effort, improved coverage, and increased productivity in SoC verification projects.

4.1 Research Gap

Although the reviewed literature provides valuable insights into the performance verification of the ARM Core Link CORTEX® -A65 CORE and SoC verification methodologies, there are still some research gaps that can be addressed. These include:

1. Integration of formal verification with simulation-based approaches: While some papers discussed the use of formal verification for correctness validation, there is potential for further research on combining formal methods with simulation-based verification approaches to enhance verification efficiency and effectiveness.
2. Comprehensive analysis of the impact of interconnect parameters: Although some papers discussed the effect of parameters on the performance of the CORTEX® -A65 CORE Interconnect Fabric, a more comprehensive analysis considering a wider range of parameters and their interactions would provide deeper insights into design optimization.
3. Validation of QoS and power management features: While some papers briefly mentioned the validation of Quality of Service (QoS) and power management features, further research

can explore dedicated methodologies and techniques for their verification, ensuring their correctness and effectiveness in real-world scenarios.

4.2 Objective

The literature review aimed to examine existing research on the performance verification of the ARM Core-Link CORTEX®-A65 CORE and associated System-on-Chip (SoC) verification methodologies. The goal was to identify the methodologies utilized, the performance metrics assessed, and the efficacy of these verification strategies in detecting performance issues.

4.3 Methodology

The review involved a comprehensive search for relevant academic papers, articles, and books focusing on the performance verification of the ARM Core-Link CORTEX®-A65 CORE and SoC verification techniques. The selected documents were evaluated based on their methodologies, key findings, and insights. The analysis aimed to uncover common approaches, gaps in existing research, and potential future research directions.

4.4 Conclusion

The literature survey revealed that various methodologies and tools have been developed for the performance verification of the ARM Core-Link NIC-400. Techniques such as System Verilog Assertions, UVM-based verification, simulation, analytical modeling, and formal verification were employed in different studies to assess the NIC-400's performance. The findings indicated that the CORTEX®-A65 CORE can achieve high throughput with low latency across various traffic scenarios, and the verification methodologies were effective in identifying performance issues. The increasing use of formal verification is a notable trend in SoC performance verification, suggesting that future research could benefit from integrating formal methods with simulation-based verification approaches.

Chapter 3

SOFTWARE-DRIVEN VERIFICATION OF THE PLATFORM

When the processor core is the part of the system being verified, it is more convenient to write the testbench program in a language that the processor can easily understand. Thus, Software Driven Verification is used for such systems, as explained in section 4.2.2.

5.1 PLATFORM STRUCTURE

Whenever a company manufactures a product, it generally produces different variants which are similar in design for the most part with some variations to target different customer requirements. The part of the design, that is common among different projects, is called the platform. Similarly, STMicroelectronics designs many variants belonging to the same project family for different customers. All the Real-Time processor cores and main memories are classified under Platform. Subsystem-level verification is carried out for the platform, and it is a software-driven verification i.e., the processor core is involved in driving the test stimuli. A part of the functionality is also verified using Formal Verification techniques.

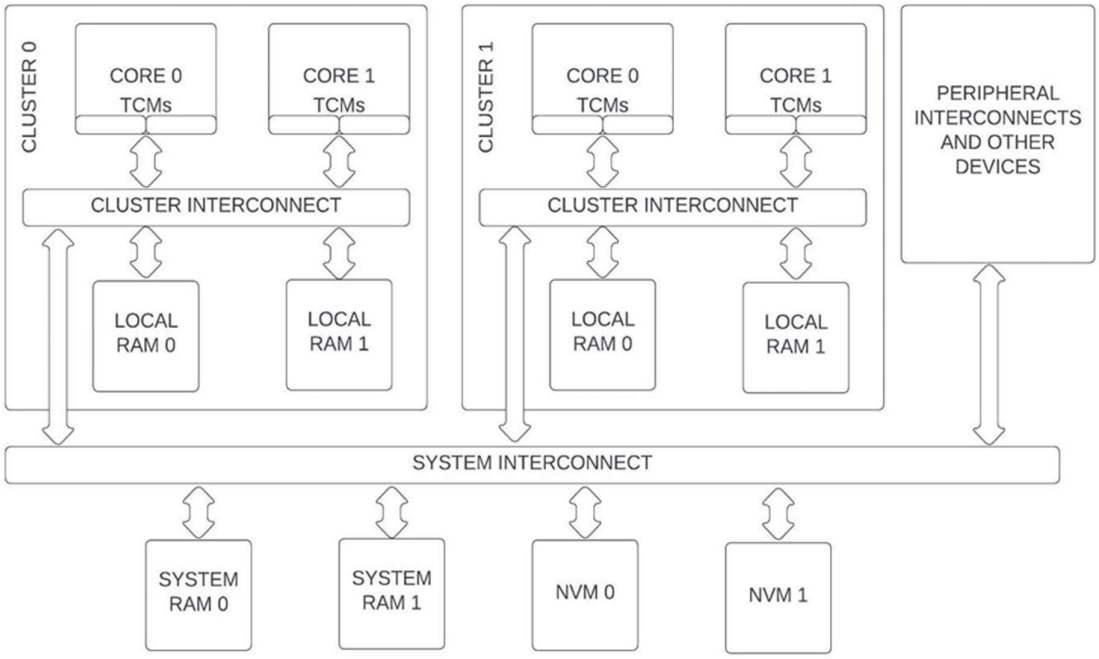

Figure 5.1 Structure of Platform

In the project named Stellar G5, the platform consists of two clusters containing RT52 processor cores. Tightly Coupled Memories (TCMs) and Local RAMs are also part of the 40-processor cluster. Two intra-cluster interconnects, one for each cluster, and a common inter-cluster interconnect are part of the platform. Several

Non-Volatile Memories (NVMs) and System RAMs are also part of the platform. Figure 6.1 shows the structure of the platform.

5.2 DATAFLOW PATHS ON THE PLATFORM

In Figure 5.1, the components that are placed above the interconnects act as masters, while the ones below the interconnects act as slaves. The masters can initiate the read or write transactions. The processor cores are the primary masters of the systems. The transaction is issued by the core. If the transaction address is local (for TCMs present inside the core cluster), the signal does not go to the interconnect and directly accesses the TCMs. Otherwise, the signal leaves the core and reaches the local interconnect. The interconnect then resolves the address and identifies the destination of the transaction. If the destination is among the local RAMs of the same cluster, the interconnect sends the signal directly to the RAM controller connected to it. Otherwise, the local interconnect passes on the signal to the system interconnect outside the cluster. The system interconnects again and checks the destination. If the target is connected directly to the system interconnect, like the System RAM or the Non-Volatile Memory, the signal is passed directly to the target. If the destination lies in another core cluster, the system interconnect further sends the signal to the local interconnect of that cluster. From there the local interconnect identifies the target and passes on the signal accordingly. At the target, the global address is converted back to the local address. There might also be a change in the communication protocol e.g., when an AXI signal reaches the peripheral bridge, it is converted into an IPS signal. IPS protocol is used to read or write to the configuration registers of various components on SOC.

5.3 VERIFICATION ACTIVITIES ON PLATFORM

The primary aim of platform verification is to ensure the proper signal flow from the cores to various slaves and to check the working of different memories present on the chip. Some of the tests that are performed are listed below.

5.3.1 Sanity Check: It is a fundamental test that must be run with every new release. This test simply ensures that all the components are present at their respective address locations as defined by the design files and they are accessible. If there is any fault with the connections or components placement in the design or port mismatch with the testbench, it will be caught at this point. A Tester can proceed with other tests only if the sanity test is clean.

5.3.2 Access Tests: Access tests are the next step before the transactions can be performed. Access test simply checks whether all the components are accessible to the core. There can be different 41 access modes, allowing access to various components under different circumstances. All these scenarios are checked.

5.3.3 Marching Test: The marching test performs 32-bit writes and reads on the target memory at different intervals. At first, the test writes zeroes to memory locations with an interval of 100 bytes. Then it reads back the written addresses and expects zero value. Then the test writes value 0xFFFFFFFF to different memory

locations at different intervals and reads back expecting all Fs. After this, the test writes a combination of 5's and A's. This is done to write consecutive bits as 1 and 0. In this test read transactions are performed only when all the writes at a particular value have been performed and not simultaneously.

5.3.4 Memory Fill Test: This is a more thorough version of the marching test. In this, a written transaction is performed and immediately read back. The write data values are 32-bit in size, and they are performed at all memory locations. Since the memory is byte addressable and 32 bits write and read can be performed at once, the write transactions are performed at a gap of 4 in memory addresses. This is because if we write a 32-bit value to a byte-addressable memory location M, the locations M+1, M+2, and M+3 also get written. So, the next 32-bit writes need to be performed at M+4, M+8, and so on. This test takes a long time to complete, because of its exhaustive nature.

5.3.5 Boundary Check Test: To prevent memories from overlapping, they are not assigned consecutively in the address space and there is some gap between them. This gap is known as reserved space. A boundary check test is performed to ensure that the memories span only up to the space they are designed to be. In this test, several writes are performed just outside the upper and lower boundary of the memory. For the test to pass, these writes should fail. If any of these write transactions is successful, it indicates that the memory exceeds its designated value. Thus, the test fails.

5.3.6 Reserved Space Check: All the unassigned address locations are known as reserved spaces. Transactions are randomly performed on these locations to make sure that no read/writable memory component exists in this space. Just like the boundary check test, all the transactions performed in this area should fail. Failure of the transactions to the reserved area is the passing criteria of the test. If any transaction is successful in the reserved space, the test is supposed to fail.

5.3.7 Restricted Access Test: Not all the components on the chip are supposed to be accessible via main processor cores. Especially the components related to safety and security are shielded by hardware protection to prevent any illegal access or hacking. This test to performed to ensure that such areas are inaccessible. If a restricted area is accessible to the core, the test results in failure.

Chapter 4

TOOLS AND METHODOLOGIES

6.1 EDA TOOLS

Several different kinds of EDA tools are used in the industry for RTL design and verification. STMicroelectronics uses the Cadence verification suite. The most used Cadence tools are Xcelium, Jasper Gold, Per spec, V manager, etc.

6.1.1 Cadence Xcelium

Cadence Xcelium is a very useful Logic Simulator tool that can be used for the verification of individual IPs or full SOC. It can simulate designs based on System Verilog, VHDL, System-C, UVM, mixed-signal, and low-power systems. It can bind with other related apps for increased flexibility and versatility. It can even use Machine Learning test compression and other features to speed up coverage and enable an early verification closure for IP and SoC designs [17]. Figure 6.1 shows the snippet of Cadence Xcelium Window.

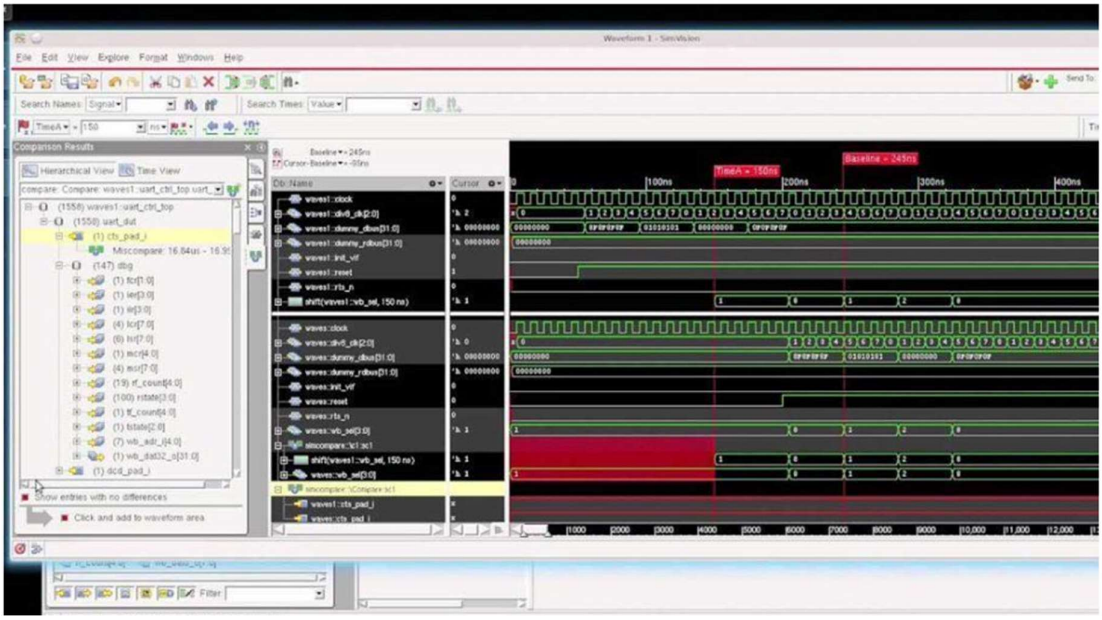

Figure 6.1 Cadence Xcelium Window

Xcelium allows the verification engineers to see the waveforms of the design signals and allows access to Verilog source code. The designers can also choose to see the schematic of the design to debug the errors. This is the primary tool for running SV and UVM-based test environments. It can be also used for reporting and plotting waveform in software-driven 24 verifications i.e. verification of tests written in C language. This is done through a porthole mechanism. The Porthole is explained further in section 6.2.2.

6.1.2 Cadence V-manager

V-manager is a powerful, scalable, and automated verification planning and management solution supporting multi-user, multi-engine, multi-projects, and multi-sites simultaneously [18]. Xcelium is used to run standalone tests, whereas V-manager is a tool that allows to create and maintains verification plans for many tests and launch regressions. It can be used for both (pre) and (post) silicon functional verification. In addition to running tests, it also provides coverage metrics and allows users to run split regressions if the number of tests is very large. In split regression, the regression is split into several parts and these parts run in parallel from several users' ends. The results are then merged as if it were a single regression. This process speeds up the verification process manyfold. The regressions are a kind of macro test. These do not provide as detailed information about individual tests as Xcelium but provide useful metrics about the test suite. Figure 4.2 shows a snippet of Cadence V-manager.

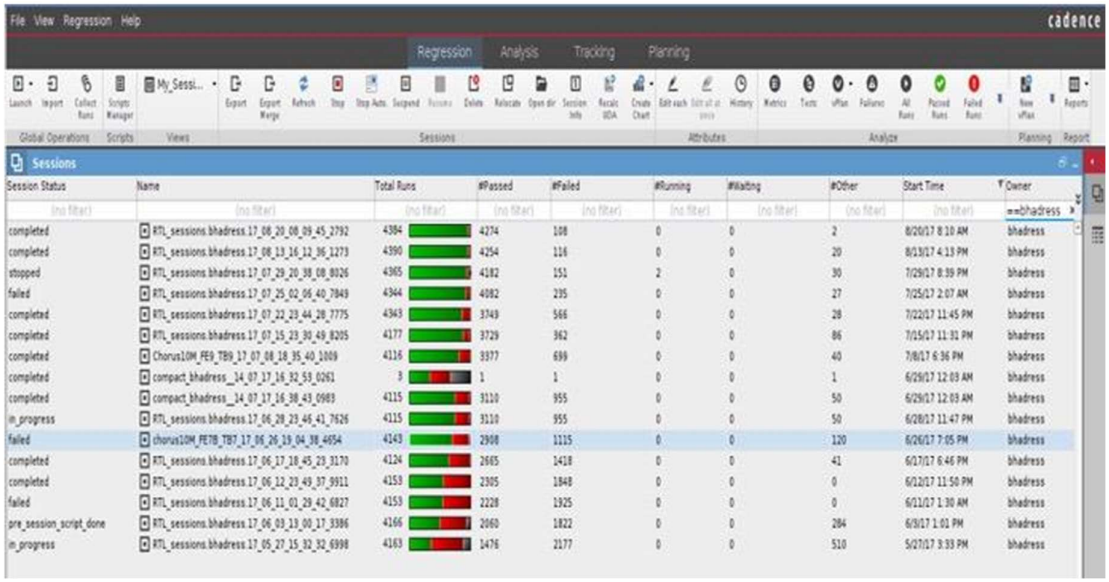

Figure 6.2 Cadence V-manager Window

V-manager provides several tabs to perform various tasks, like Analysis, Tracking, and Planning. The regression window shows all the present and past regressions and their status. This status can be analyzed in detail in the analysis window. The regression window shows the number of tests passing, failing, or running. Whereas the analysis window provides the names and paths of the tests along with their status.

Planning window can be used to manage existing verification plans or create new ones. The verification plan, also known as V-Plan, contains the list of tests and their paths to be executed for verifying the functionality of a particular part of the design. It may also contain other details and implementation notes about the tests. The plans can be reused for new projects simply by changing the test paths.

6.1.3 Cadence Jasper-Gold

Jasper is the premier electronic design automation (EDA) supplier of high-level formal functional verification software. Jasper-Gold formal and formal-assisted technology is integrated into the Cadence System Development Suite [19]. Jasper Gold tool takes assertion inputs from a text file in a particular format. Along

with that, it takes several other files as input for setup, forcing signals, etc. The main setup files are written in TCL language, while the other auxiliary files can be written in Python or Perl. Figure 6.3 shows a snapshot of the Jasper Gold tool window.

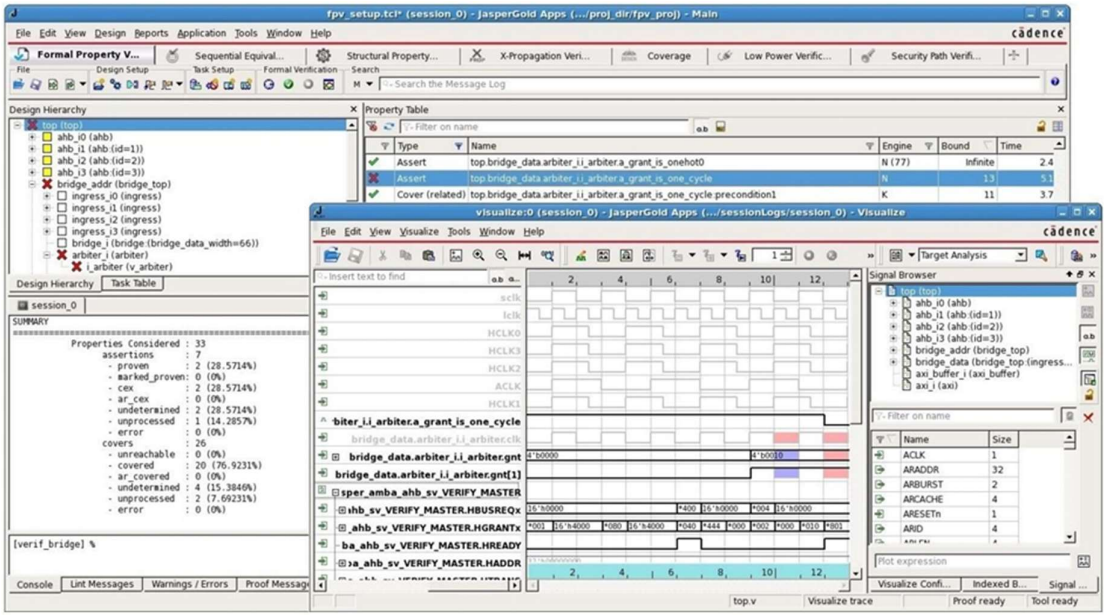

Figure 6.3 Cadence Jasper Gold Window

6.1.4 Cadence Prospect

The Prospect System Verifier is a model-based, goal-directed SoC verification product developed to meet verification challenges [20]. The Prospect usage flow includes the following steps:

1. Capture the SoC actions needed to create a desired use case, if not already captured.
2. Compose the desired use case. 26
3. Use the Prospect System Verifier to solve the abstract use case to create concrete use cases or scenarios.
4. The Prospect solution generates C tests for the concrete scenario mapped to a specific execution platform.
5. Run the tests on the targeted platform.
6. Debug the test and review coverage results.

Figure 6.4 shows the flowchart describing the working of the prospect tool.

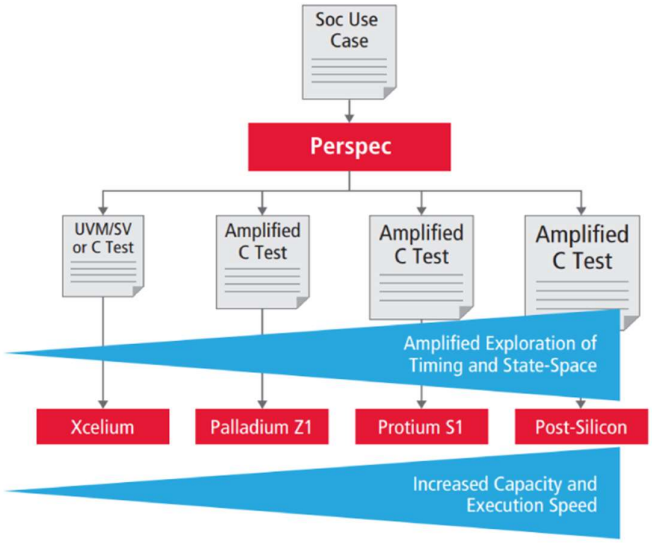

Figure 6.4 Cadence Per Spec Flowchart

Source of Above Image:- By:-

https://www.cadence.com/en_US/home/resources/datasheets/perspec-system-verifier-ds.html

6.2 VERIFICATION METHODOLOGIES

As mentioned in section 1.1, SoC verification is done at three levels. The Subsystem is the middle level between individual IP and full SOC. Verification at each level requires a slightly different kind of methodology as per changes in requirements. The first step is IP verification. The most common method for IP verification is Universal Verification Methodology (UVM) based testbenches.

6.2.1 Universal Verification Methodology (UVM)

UVM is one of the methodologies that were created from the need to automate verification. The Universal Verification Methodology is a collection of API and proven verification 27 guidelines written for System Verilog that help an engineer create an efficient verification environment. It is an open-source standard maintained by ACCELLERA [21] and can be freely acquired on their website. Figure 4.5 shows the structure of a UVM test bench.

The main UVM Components are as follows:

6.2.1.1 Interface: The Interface is designed to encapsulate and allow smooth connectivity between the highest-level and lowest-level components in the test bench. Interfacing also encourages the reuse of plans and designs. Interfaces are progressive structures that can contain additional interfacing and can be parameterized like a module according to system-level modeling and test bench needs. They also reduce the amount of code required to simulate port connections, making the interface more maintainable and readable.

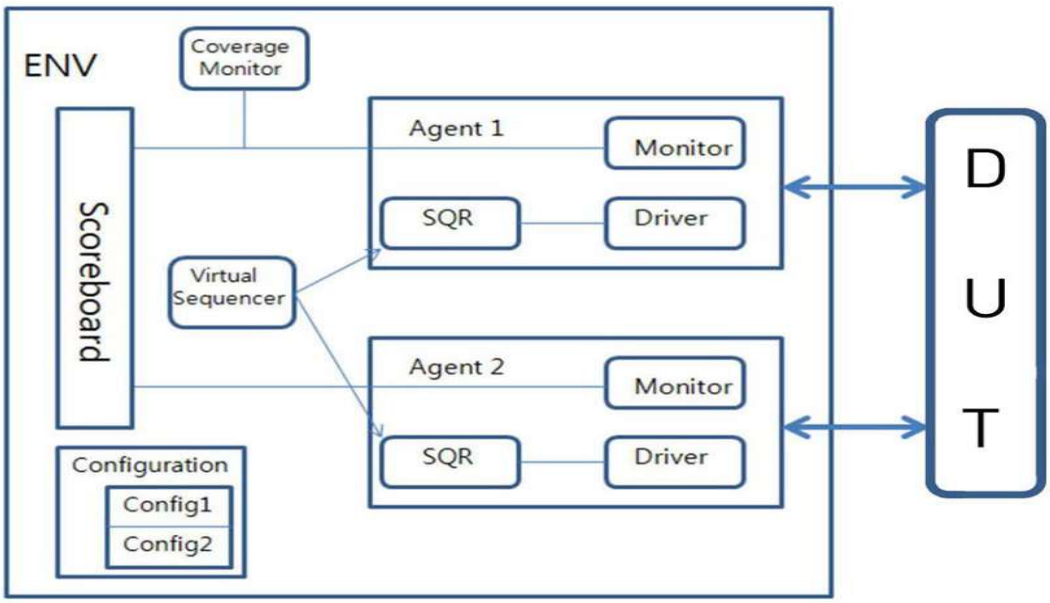

Figure 6.5 UVM Testbench Structure

Source of Above Image:- Functional verification of a safety class controller for NPPs using a UVM register model - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Diagram-of-the-Designed-Controller-Fig-2-Structure-of-a-UVM-Testbench_fig2_271715177 [accessed 12 July 2024]

6.2.1.2 UVM Sequence Item: The data fields in the sequence items are driven to the DUT. These are the subclasses of the UVM sequence item, which is itself a subclass of the UVM transaction. To generate a stimulus for the DUT, data fields in sequence items are modeled. The only rationale for constructing a user-defined sequence item that is derived from the UVM sequence item class is to take advantage of the base class's provided functions such as print, copy, and clone. These data fields are often randomized together with some restrictions. Therefore, they are specified as random and have constraints defined with them.

6.2.1.3 UVM Sequence: A sequence creates a succession of sequence items and transmits them to the driver through the sequencer. The UVM sequence class is extended to create sequences. A UVM sequence is derived from the UVM sequence item class and is parameterized with the type of sequence items. This determines the type of item sequence that will be sent or received to or from the driver.

6.2.1.4 UVM Sequencer: Between the sequences and the driver, the sequencer controls the flow of solicitation and responses. As a result, the driver and sequencer employ TLM interfaces to communicate among themselves. UVM sequencer and UVM driver define sequence item export and sequence item port, respectively, and the user connects them using the TLM connect method. The sequencer is created by extending the UVM sequencer class with the sequence item type as a parameter. It is a sophisticated stimulus generator

that generates and returns data items in response to the driver's commands. By altering the randomization weights, a sequencer can adapt to the present condition of the DUT and generate more valuable data items.

6.2.1.5 UVM Driver: The driver is a block that works with the DUT by transforming high-level code into Bus functional models (BFMs). It imitates the logic that powers the DUT. The driver can either request sequences from the sequencer (pull mode) or have sequences sent to it (push mode). Drivers work in pull mode by default, repeatedly pulling data items created by the sequencer and driving them to the DUT via an interface. UVM driver, which is further developed from the UVM component, is used to create the user-defined driver. The UVM driver is a parameterized class that includes the sequence item argument.

6.2.1.6 UVM Monitor: This is a stand-alone component that monitors all transactions between the DUT and the test bench. It samples the DUT interface, captures the signals received from the DUT, and delivers them to other components for additional analysis, such as scoreboards. The monitor throws an error if the communication does not follow the standards (protocol). It extracts signal-level data and converts it into sequence item format. Every signal UVC has its monitor, and each monitor takes data from its interfaces and sends it to the scoreboard via the analysis port. It is a passive entity that is derived from the UVM monitor class, which is derived from the UVM component class.

6.2.1.7 UVM Agent: The UVM agent, which is derived from the UVM component, is the foundation class for the user-specified agent. It encapsulates the driver, sequencer, and monitor. More than one agent can be found in a UVC. In a SoC-based example, different agents can be employed to target different protocols. It has a build and a connect stage, in which components such as drivers, sequencers, and monitors are produced and connected, but there is no run phase. Agents can be set to be active or passive. Active agents emulate devices and 29 drive DUT's ports. Passive agents are useful when no data items need to be extracted from the DUT and only monitoring of the DUT activity is required.

6.2.1.8 UVM Scoreboard: A scoreboard monitors the flow of exchange levels in and out of the DUT to ensure that it is operating as intended. It compares exchanges at the input of the plan under test to exchanges coming out of the DUT. For example, it could track packages in vs. packages out to verify whether all the packages placed into a communication device made it out. One of the most important confirmation components is the scoreboard. The scoreboard is a self-contained component that gathers data from numerous screens across the plan, applies the inputs to the self-contained demonstration, and calculates the expected yield. The genuine and expected yields are compared at this time.

6.2.1.9 UVM Environment: It is the top-level component where one or more agents can be found. It is inherited from the UVM component base class and extends the UVM env class. It may also have scoreboards for doing end-to-end transmission checks or comparisons to the reference models. It could also include other environments that were previously tested at the block level but are now being incorporated at the subsystem or SoC level.

Chapter 5
Testcases & results

5.1 Testcases

A test case refers to a set of inputs and expected outputs that are designed to verify the correct behavior of a particular aspect of the system under test. The purpose of a test case is to ensure that the system functions correctly and meets the specified requirements.

7.2 A test case typically consists of several components, including

7.2.1 Test inputs:

The data or commands that are used to drive the system under test. These inputs can include parameters, commands, and other stimuli that are required to exercise the system.
Expected outputs: The results or responses that are expected from the system under test in response to the given inputs. These outputs can include data, messages, signals, or other forms of feedback.
Test conditions: The specific conditions and assumptions that are necessary for the test case to be executed successfully. These conditions can include environmental factors, dependencies, and other variables that may affect the behavior of the system.

The goal of generating test cases in a system testbench is to provide comprehensive coverage of the system's functionality and to identify any defects or errors in the system's behavior. By creating test cases that cover a wide range of scenarios and edge cases, developers can ensure that the system is robust and reliable in all expected and unexpected conditions.

7.2.2 Results:

The cache way controller unit's design implementation has been simulated using Cadence NC-Sim. This section presents the simulation results for various mapping techniques utilized by the cache way controller unit.

7.2.3 Direct Mapping Cache:

Figure 7.1 illustrates the use of single cache memory with direct mapping, where the presence of data in the cache requires searching through all tag fields.

Figure 7.1 Cache in Direct Mapping.

7.2.4 Two-Way Cache Mapping:

In Figure 7.2, two distinct cache memories are used, each with a size of 256 Kbytes.

Figure 7.2 Two-Way Cache Mapping.

In the two-way mapping technique, when a cache miss occurs, the CPU accesses data from the main memory, as illustrated in Figure 7.3.

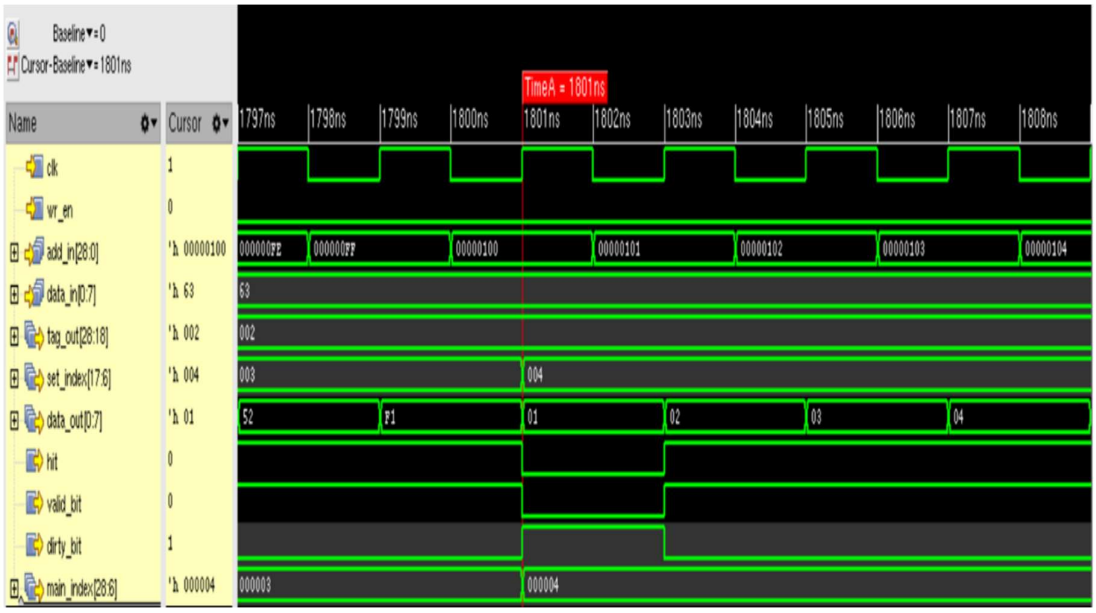

Figure 7.3 Cache Miss in a Two-Way Mapping.

During a cache miss, the cache memory is updated with a new main memory block, depicted in Figure 7.4.

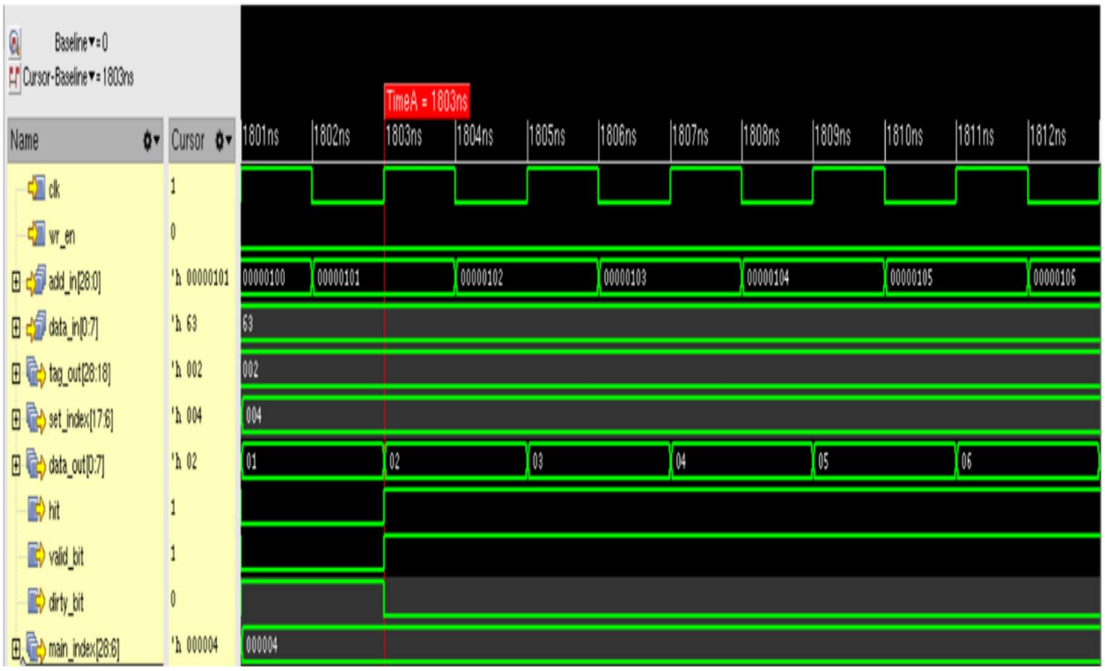

Figure 7.4 Cache Hit in a Two-Way Mapping.

7.2.5 Four-Way Cache Mapping:

Figure 7.5 illustrates the simulation of four separate cache memories, each with a cache set size of 128 Kbytes.

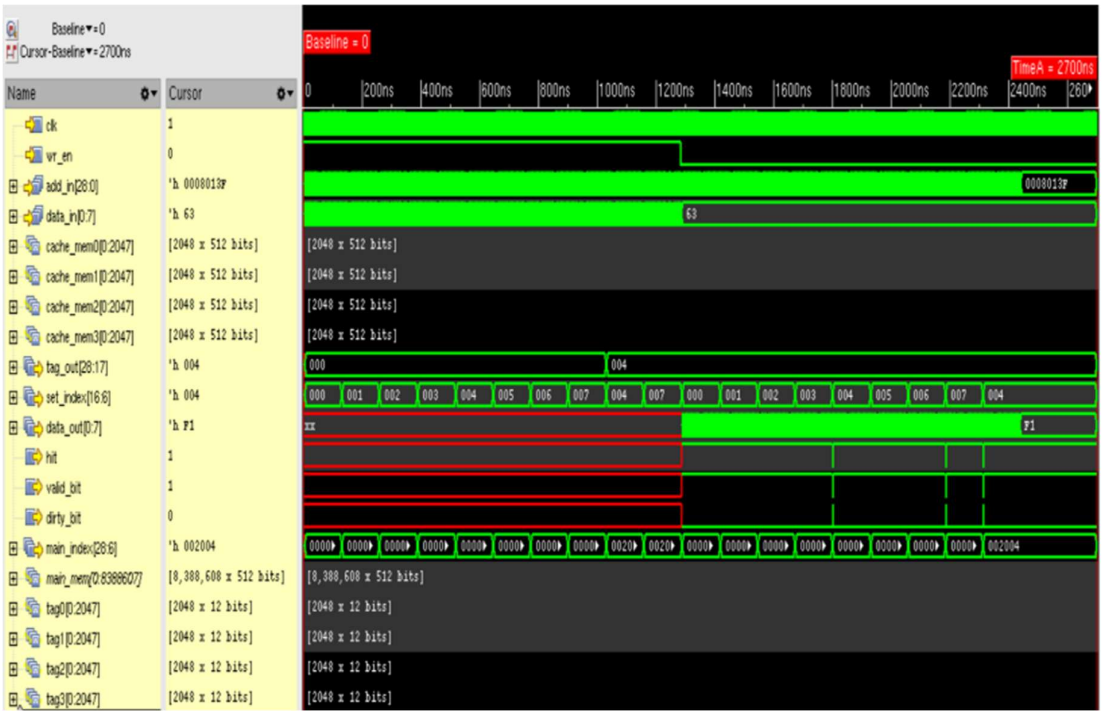

Figure 7.5 Mapping the Cache in a Four-Way Configuration.

7.2.6 Eight Way Cache Mapping:

Figure 7.6 depicts eight separate cache memories, each with a cache set size of 64 Kbytes.

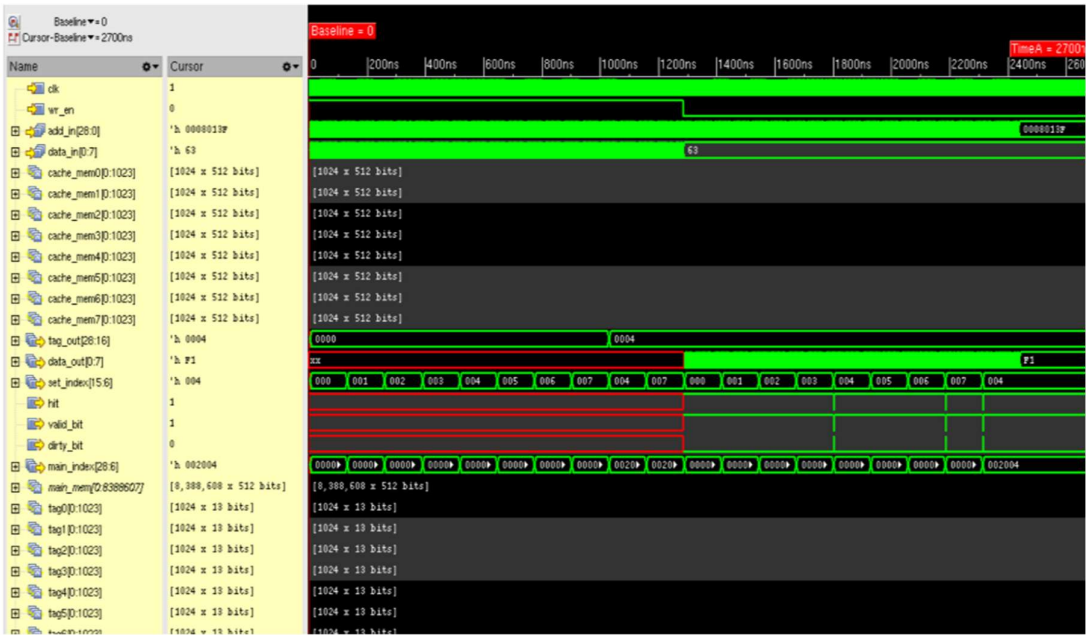

Figure 7.6 Mapping the Cache in an Eight-Way Configuration.

Chapter 6
Performance Analyzer

Performance analysis using a system testbench generator involves the use of simulation-based testing to evaluate the performance of a hardware design. A testbench generator is a tool that automatically generates a testbench that can be used to stimulate and verify the functionality of the design under test.

The performance analysis involves the measurement of various metrics such as latency, throughput, and resource utilization. Latency refers to the time it takes for a request to be processed and a response to be generated. Throughput is the rate at which data can be processed by the system. Resource utilization refers to the amount of hardware resources such as memory, CPU cycles, and bandwidth that are consumed by the design.

The system testbench generator typically consists of a set of pre-defined scenarios that can be executed to test the system under different conditions. These scenarios are based on real-world usage patterns and can be used to evaluate the performance of the system under varying workloads.

By using a system testbench generator, designers can identify performance bottlenecks and optimize the design for better performance. This can involve tweaking parameters such as clock speed, buffer sizes, and resource allocation to achieve optimal performance. The testbench generator can also be used to validate the performance of the design under different environmental conditions such as varying temperatures and operating conditions.

Overall, the use of a system testbench generator for performance analysis can help ensure that the hardware design meets the desired performance requirements and is capable of handling real-world workloads.

With increasing system complexity, ensuring the functional correctness of the SoC interconnect and its ability to meet the required latency and bandwidth demands for the target use cases is crucial. The time taken by data to travel from one point to another is referred to as latency, and it is influenced by the distance the data must traverse across the network. It is dependent on the physical distance that data must travel through the networks.

It is measured in nanoseconds (ns) in milliseconds (ms) or clock cycles. Bandwidth (or Throughput) is the quantity (rate) of data being sent and received within a unit of time. Bandwidth refers to the capacity of a communication channel, indicating how much data can be transmitted through it in each period. Essentially, a wider bandwidth allows for a greater volume of data to be transmitted simultaneously. It’s measured in Mbps or megabits per second.

Chapter 7

Conclusion & Future scope

7.1 Conclusion

- The project focused on reconfiguring cache memory to meet specific user requirements.
- In direct mapping, a 512 Mb main memory was mapped to a 512-Kbyte cache memory. In two-way set associative mapping, the 512-Kbyte cache memory was divided into two equal sets of 256 Kbytes each.
- Similarly, for 4-way set associative mapping, the cache was divided into four sets of 128 Kbytes each, and for 8-way set associative mapping, it was divided into eight sets of 64 Kbytes each.
- The key advantage of set-associative mapping is that it activates only one cache set at a time while keeping the others inactive. This enhances data access speed and reduces power consumption by placing the unused cache sets in standby mode.

7.2 Future scope

- This project concentrated on 8-bit (1-byte) data access design. Moving forward, the aim could be to implement 64-bit (8-byte) data access and create a reconfigurable 512-Kbyte cache memory using Verilog HDL. This would enhance data retrieval efficiency by allowing larger data transfers per clock cycle.
- Such advancements are expected to boost the efficiency and accuracy of the verification process by lessening the manual effort required for creating and managing test benches.
- Further development could involve refining the system testbench generator tool and exploring additional automation techniques for the verification and validation of complex designs.
- Incorporating formal verification techniques along with system testbench generation could also be considered to further enhance the accuracy and thoroughness of the verification process.

References

1.1. (Journal) Two Authors

[1] Y-Jin. Oh, and G-Yong. Song, “System-Level Verification Platform using System Verilog Layered Testbench & System-C OOP” International Journal of Control and Automation Vol.7, No.2 (2014), pp.221-230, (2014).

[2] Sur- Kolay Susmita, and Saha Debasri, "SoC: A Real Platform for IP Reuse, IP Infringement, and IP Protection," CAD for Gigascale SoC Design and Verification Solutions, Volume 2011, No.: 5, Page No.:1-10, (January 2011).

1.2. Conference Proceedings/ Edited Books

[3] Yellampalli Siva S., Manjunatha K. N., Kanagasabapati, Design of Reconfigurable Cache Memory Using Verilog HDL (2018 Third International Conference on Electrical, Electronics-Communication, Computer Technologies, and Optimization Techniques (ICEECCOT), pp14-15, (December 2018).

[4] Young-Nam Yun, “Beyond UVM for practical SoC verification”, SoC Design Conference (ISOCC), pp158-162, (2011).

1.3. Books

[5] M. A. B, “ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies”, Springer International Publishing, (2018), pp 346-369.

[6] J. G. W. R. Bruce Wile, “Comprehensive Functional Verification: The Complete Industry Cycle (Systems on Silicon)”, Morgan Kaufmann Publishers Inc., (2005). pp 136-159.

1.4. Citing a website

[7] Collins, L. (2019, November 27). Managing code coverage to achieve verification closure in low-power SoCs:- https://www.techdesignforums.com/practice/technique/managing-code-coverage-to-achieve-verification-closure-in-low-power-socs/

[8] V-Manager Verification Management. (n.d.). Cadence:- https://www.cadence.com/en_US/home/tools/system-design-and-verification/planning and management/vmanager-solution.html

[9] Vlsi4freshers.UVM:- https://www.vlsi4freshers.com/2020/04/uvm-testbench-architecture

[10] Documentation- Arm Developer. (n.d.). APB Interface:- https://developer.arm.com/documentation/ihi0024/c/Introduction/About-the-APB protocol

[11] Verification Guide: (2020, March 17) Verification Guide:- https://verificationguide.com

[12] Arm Cortex-A65 Core Technical Reference Manual:- By https://developer.arm.com/documentation/100439/0101/

[13] Figure 1.2 Reusing of IP in SOC. By:- https://www.researchgate.net/figure/a-Components-on-a-system-on-chip-b-IP-reuse-in-SoC-environment_fig2_220174590

[14] Figure 1.7.1 shows a block diagram of basic cache memory. By:- Architectures and Technologies of Cache Memory: A Survey - Scientific Figure on ResearchGate. Available from:-

https://www.researchgate.net/figure/Position-of-cache_fig1_319523795 [accessed 12 Jul 2024]

[15] Figure 6.4 Cadence Per spec Flowchart.
By:- https://www.cadence.com/en_US/home/resources/datasheets/perspec-system-verifier-ds.html

[16] Figure 6.5 UVM Testbench Structure. By:- Functional verification of a safety class controller for NPPs using a UVM register model - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Diagram-of-the-Designed-Controller-Fig-2-Structure-of-a-UVM-Testbench_fig2_271715177 [accessed 12 Jul 2024]

ACKNOWLEDGEMENT

I would like to extend my deepest gratitude to STMicroelectronics for their invaluable support and assistance throughout my M-Tech thesis. The data and resources they provided were instrumental in the successful completion of this research.

I am especially grateful to the entire team at STMicroelectronics in Greater Noida, where I completed my internship. The knowledge and experience I gained during my time there were fundamental to my work. I appreciate my colleagues' guidance, encouragement, and the conducive learning and growth environment they provided.

Specific Data Usage I acknowledge the use of data from STMicroelectronics in Tables 1.6.1, 1.6.2, and 1.6.3 of this Thesis. The availability and quality of this data significantly contributed to the depth and accuracy of my research findings. I sincerely thank STMicroelectronics for permitting me to utilize this valuable data in my thesis.

Additionally, I acknowledge the logistical and technical support provided by the Human Resources and IT departments at STMicroelectronics.

Lastly, I express my heartfelt thanks to my family and friends for their unwavering support and encouragement throughout this journey.

Vinay Tilwani

MTech Thesis (D5)

ORIGINALITY REPORT

14%	9%	7%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

4%

★ documentation-service.arm.com

Internet Source

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		