Enhancement of HDL Based Memory Models

Major Project Report

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology
in
Electronics & Communication Engineering
(VLSI Design)

By

Patel Jinisha Arvindbhai
(14MECV15)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481
May - 2016

Enhancement of HDL Based Memory Models

Major Project Report

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology
in
Electronics & Communication Engineering
(VLSI Design)

By

Patel Jinisha Arvindbhai

(14MECV15)

Under the guidance of

External Project Guide:	**Internal Project Guide:**
Ms. Harpreet Kaur	**Dr. Amisha Naik**
Project Leader(FEM)	Professor (EC Dept.),
ST Microelectronics Pvt. Ltd.,	Institute of Technology,
Greate Noida.	Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
NIRMA University
Ahmedabad-382 481
May 2016

Declaration

This is to certify that

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

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

- Patel Jinisha A.

Disclaimer

NIRMA UNIVERSITY
INSTITUTE OF TECHNOLOGY
NAAC ACCREDITED 'A' GRADE

Certificate

This is to certify that the Major Project entitled **"Enhancement of HDL Based Memory Models "** submitted by **Patel Jinisha Arvindbhai (14MECV15)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design , Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. Amisha P. Naik	**Dr. N. M. Devashrayee**
Internal Guide	PG Coordinator (VLSI Design)
Dr. P. N. Tekwani	**Dr. P. N. Tekwani**
Head, EE Dept.	(i/c) Director, IT-NU
Date:	**Place: Ahmedabad**

Certificate

This is to certify that the Project entitled ”**Enhancement of HDL Based Memory Models**” submitted by **Patel Jinisha Arvindbhai (14MECV15)**, towards the submission of the Project for requirements for the degree of Master of Technology in VLSI Design, NIRMA University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Ms. Harpreet Kaur
Project Leader,
ST Microelectronics India Pvt. Ltd.,
Greater NOIDA.

Acknowledgement

I would have never succeeded in completing my thesis without the cooperation, encouragement and help provided to me by various people. Firstly, my sincere thanks to the Front End Modeling team of DMIPS (Digital Modeling of IP Subsystems) group for their help during this training. Their wisdom, clarity of thought and support motivated me to bring this project to its present state.

I am highly indebted to my team Manager Ms. Deepshikha Singh, Group manager Mr. Anupam Jain and mentor Ms. Harpreet Kaur for providing necessary information regarding the project and also for their constant guidance, supervision, kind co-operation, and invaluable support in all aspects. My thanks and appreciations also go to my colleagues and team members in developing the project and for providing me with a lively and energetic work environment.

I would like to express my sincere gratitude to Dr. P. N. Tekwani (Director and Head of Department NIRMA University, Ahmedabad) for his continuous guidance and support. I would take this opportunity to thank Dr. N. M. Devashrayee (Professor and Program Coordinator, M.Tech - EC (VLSI Design)), Internal Guide Dr.Amisha Naik and all the faculties at Nirma University (VLSI Design), for their vision and relentless effort, support, and encouragement to provide me with this excellent opportunity to carry out my project work in such a highly renowned and esteemed organization, ST Microelectronics. I am equally thankful to ST Microelectronics for providing me the invaluable exposure to the industry and the current market trends.

- Jinisha Patel
14MECV15

Abstract

As process of fabrication technologies advances, chip complexity increases and the design flow becomes more iterative. Iterations in the design flow cost money, time and engineering resources that adversely affect the time to market and cost of the devices being designed.

This report deals with the development of a generic HDL models and describes the validation process and need for automation of validation environment of behavioral memory models.

The project deals with Development of SP, ROM, p-REG & DPREG behavioral models and the use of SV verification environment to validate those behavioral models and using Shell Scripts to automate the validation process.

The technology does affect the physical level of the design, but functionality does not change. The customers needs affect the model structure and the functionality, not the technology. Thus we always prefer that the given model works for the technology, not on that technology.

NOTE: Since the work done in this project is of confidential nature, more stress is given on concepts than on actual work done.

Contents

List of Figures

Abbreviation Notation and Nomenclature

ALU	Arithmetic Logic Unit
AMS	Analog & Mixed Signal
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BGA	Ball Grid Array (BGA)
CISC	Complex Instruction Set Computer
DIP	Dual In-line Package
DP	Dual Port
FEM	Front End Modelling
GDS	Graphic Data System
GLS	Gate Level Simulation
IP	Intellectual Property
MAS	Micro-Architectural Specification
MCM	Multi-Chip Modules
MIF	Map Into Format
MOS	Metal Oxide Semiconductor
NVRAM	Non-Volatile Random Access Memory
OTP	One Time Programmable
OVI	Open Verilog International
p-REG	pseudo Register
PCB	Printed Circuit Board
PGA	Pin Grid Array
QFP	Quad Flat Package
RISC	Reduced Instruction Set Computer
SoC	System on Chip
SP	Single Port
STA	Static Timing Analysis

Chapter 1

Introduction to VLSI

1.1 History and Evolution Of Integrated Circuits

The development of microelectronics spans a time which is even lesser than the average life expectancy of a human, and yet it has seen as many as four generations. Early 60s saw the low density fabrication processes classified under Small Scale Integration (SSI) in which transistor count was limited to about 10. This rapidly gave way to Medium Scale Integrationin the late 60s when around 100 transistors could be placed on a single chip.

It was the time when the cost of research began to decline and private firms started entering the competition in contrast to the earlier years where the main burden was borne by the military. Transistor-Transistor logic (TTL) offering higher integration densities outlasted other IC families like ECL and became the basis of the first integrated circuit revolution. It was the production of this family that gave impetus to semiconductor giants like Texas Instruments,Fairchildand National Semiconductors. Early seventies marked the growth of transistor count to about 1000 per chip called the Large Scale Integration.

By mid eighties, the transistor count on a single chip had already exceeded 1000 and hence came the age of Very Large Scale Integration or VLSI. Though many improvements have been made and the transistor count is still rising, further names of generations like ULSI are generally avoided. It was during this time when TTL lost the battle to MOS family owing to the same problems that had pushed vacuum tubes into negligence, power dissipation and the limit it imposed on the number of gates that could be placed on a single die.

1.2 VLSI Design Flow

The VLSI design cycle starts with a formal specification of a VLSI chip, follows a series of steps, and eventually produces a packaged chip. A typical design cycle may be represented by the flow chart shown in Figure. Our emphasis is on the physical design step of the VLSI design cycle. However, to gain a global perspective, we briefly outline all the steps of the VLSI design cycle. [6]

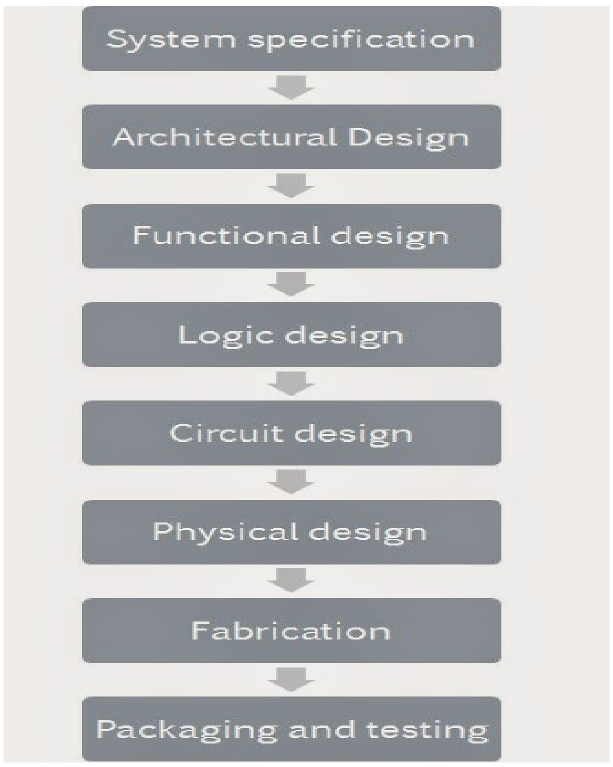


Figure 1.1: VLSI Design Flow

- **System Specification:**
  The first step of any design process is to lay down the specifications of the system. System specification is a high level representation of the system. The factors to be considered in this process include: performance, functionality, and physical dimensions (size of the die (chip)). The fabrication technology and design techniques are also considered.
  The specification of a system is a compromise between market requirements, technology and economical viability. The end results are specifications for the size, speed, power, and functionality of the VLSI system.

- **Architectural Design:**
  The basic architecture of the system is designed in this step. This includes, such decisions as RISC (Reduced Instruction Set Computer) versus CISC (Complex Instruction Set Computer), number of ALUs, Floating Point units, number and structure of pipelines, and size of caches among others.
  The outcome of architectural design is a Micro-Architectural Specification (MAS). While MAS is a textual (English like) description, architects can accurately predict the performance, power and die size of the design based on such a description.

- **Behavioral or Functional Design:**
  In this step, main functional units of the system are identified. This also identifies the interconnect requirements between the units. The area, power, and other parameters of each unit are estimated. The behavioral aspects of the system are considered without implementation specific information. For example, it may specify that a multiplication is required, but exactly in which mode such multiplication may be executed is not specified. We may use a variety of multiplication hardware depending on the speed and word size requirements. The key idea is to specify behavior, in terms of input, output and timing of each unit, without specifying its internal structure.

The outcome of functional design is usually a timing diagram or other relationships between units. This information leads to improvement of the overall design process and reduction of the complexity of subsequent phases. Functional or behavioral design provides quick emulation of the system and allows fast debugging of the full system. Behavioral design is largely a manual step with little or no automation help available.

- **Logic Design:**

  In this step the control flow, word widths, register allocation, arithmetic operations, and logic operations of the design that represent the functional design are derived and tested.

  This description is called Register Transfer Level (RTL) description. RTL is expressed in a Hardware Description Language (HDL), such as VHDL or Verilog. This description can be used in simulation and verification. This description consists of Boolean expressions and timing information. The Boolean expressions are minimized to achieve the smallest logic design which conforms to the functional design. This logic design of the system is simulated and tested to verify its correctness. In some special cases, logic design can be automated using high level synthesis tools. These tools produce a RTL description from a behavioral description of the design.

- **Circuit Design:**

  The purpose of circuit design is to develop a circuit representation based on the logic design. The Boolean expressions are converted into a circuit representation by taking into consideration the speed and power requirements of the original design. Circuit Simulation is used to verify the correctness and timing of each component.

  The circuit design is usually expressed in a detailed circuit diagram. This diagram shows the circuit elements (cells, macros, gates, transistors) and interconnection between these elements. This representation is also called a netlist.

Tools used to manually enter such description are called schematic capture tools. In many cases, a netlist can be created automatically from logic (RTL) description by using logic synthesis tools.

- **Physical Design:**

  In this step the circuit representation (or netlist) is converted into a geometric representation. As stated earlier, this geometric representation of a circuit is called a layout. Layout is created by converting each logic component (cells, macros, gates, transistors) into a geometric representation (specific shapes in multiple layers), which perform the intended logic function of the corresponding component. Connections between different components are also expressed as geometric patterns typically lines in multiple layers.

  The exact details of the layout also depend on design rules, which are guidelines based on the limitations of the fabrication process and the electrical properties of the fabrication materials. Physical design is a very complex process and therefore it is usually broken down into various sub-steps. Various verification and validation checks are performed on the layout during physical design. In many cases, physical design can be completely or partially automated and layout can be generated directly from netlist by Layout Synthesis tools. Layout synthesis tools, while fast, do have an area and performance penalty, which limit their use to some designs. Manual layout, while slow and manually intensive, does have better area and performance as compared to synthesized layout. However this advantage may dissipate as larger and larger designs may undermine human capability to comprehend and obtain globally optimized solutions.

- **Fabrication:**

  After layout and verification, the design is ready for fabrication. Since layout data is typically sent to fabrication on a tape, the event of release of data is called Tape Out.Layout data is converted (or fractured) into photo-lithographic masks, one for each layer. Masks identify spaces on the wafer, where certain materials need to be deposited, diffused or even removed. Silicon crystals are grown and sliced to produce wafers. Extremely small dimensions of VLSI devices require that the wafers be polished to near perfection. The fabrication process consists of several steps involving deposition, and diffusion of various materials on the wafer. During each step one mask is used. Several dozen masks may be used to complete the fabrication process.

A large wafer is 20 cm (8 inch) in diameter and can be used to produce hundreds of chips, depending of the size of the chip. Before the chip is mass produced, a prototype is made and tested. Industry is rapidly moving towards a 30 cm (12 inch) wafer allowing even more chips per wafer leading to lower cost per chip.

- **Packaging, Testing and Debugging:**
  Finally, the wafer is fabricated and diced into individual chips in a fabrication facility. Each chip is then packaged and tested to ensure that it meets all the design specifications and that it functions properly. Chips used in Printed Circuit Boards (PCBs) are packaged in Dual In-line Package (DIP), Pin Grid Array (PGA), Ball Grid Array (BGA), and Quad Flat Package (QFP). Chips used in Multi-Chip Modules (MCM) are not packaged, since MCMs use bare or naked chips.

The chip design includes different types of processing steps to finish the entire flow. For anyone who just started his carrier as a VLSI engineer has to understand all the steps of the VLSI design flow to become good in his area of operations. There are different types of design procedures.

1.2.1 Front End Design

- **Design entry:** It describes the RTL (Register Transfer Level) logics in HDLs. For this, we use any of the hardware description languages (HDLs) such as verilog and VHDL. This design specification contains all the details which all

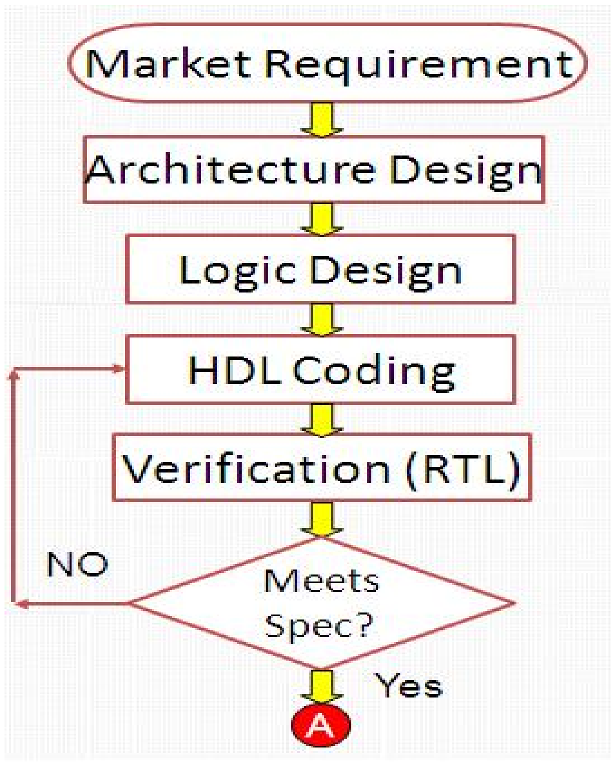


Figure 1.2: Front End Design Flow

are required for the design architecture, RTL block diagram, clock frequency, frequency domain details, waveforms, port details etc.

- **Logic Synthesis:** The RTL logic written is synthesized to get the gate level netlist. This process can be done with the help of EDA tools. The code written can be implemented on an FPGA board only if, it is synthesizable.

- **Gate level simulation:** The gate level simulation of the logic is very important in the verification. The functional check, timing checks and the Power analysis checks are included in the verification.

1.2.2 Back End Design

- **Schematic entry:** Using the cadence schematic editor in icms, you can create and extract the logic design you needed.

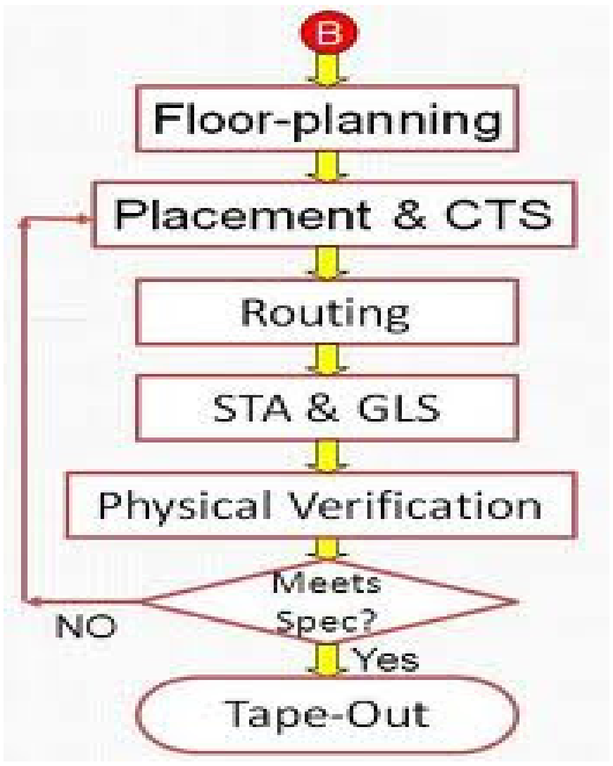


Figure 1.3: Back End Design Flow

- **Pre layout simulation:** The logic design is then verified, before doing its layout. The netlist of the schematic is generated and simulated using any of the tools such as Cadence ultrasim or Synopsys hspice. The working of each circuit can be checked using the simulation results.

- **Layout Design:** After the schematic is simulated and verified, the corresponding mask layers of the circuit should be created which can be done using the Layout Editor by cadence. The layout design includes the floor planning, placement and routing.
  The locations of each schematic component are decided in the floor planning and are placed accordingly. While routing, the interconnections are done between the components.

- **Extracted simulation:** The layout design should be extracted with the parasitics and simulate the system for performance using hspice or ultrasim. The parasitics can be either resistance or capacitances which are produced because of the interconnecting wires using for the routing. The parasitic components may affect the circuit performance badly. The parasitics cannot be avoided but can be reduced by proper routing. Finally the routed netlist that is called as GDS-II will be sent to the foundry and the chip will be manufactured as per the technology requirement.

Chapter 2

Verilog : Hardware Description Language

2.1 Introduction

In electronics, HDL is a language from a class of computer language for formal description of electronic circuit. It can describe circuit operation, its design and tests to verify its operation at any level. It also allows for the synthesis of a HDL description into a netlist (a specification of physical electronic components and how they are connected together), which can then be placed and routed to produce the set of masks used to create an integrated circuit. VHDL and VERILOG are popular HDLs.

2.1.1 History

Verilog was created by Prabhu Goel and Phil Moorby between late 1983 and early 1984. Although the history of the Verilog HDL goes back to the 1980s, when a company called Gateway Design Automation developed a logic simulator, Verilog-XL, and with it a hardware description language.

Cadence Design Systems acquired Gateway in 1989, and with it the rights to the language and the simulator. In 1990, Cadence put the language (but not the simulator) into the public domain, with the intention that it should become a standard, non-proprietary language. The Verilog HDL is now maintained by a non profit making organisation, Accellera, which was formed from the merger of Open Verilog International (OVI) and VHDL International. OVI had the task of taking the language through the IEEE standardisation procedure.
In December 1995 Verilog HDL became IEEE Std. 1364-1995. A significantly revised version was published in 2001: IEEE Std. 1364-2001. There was a further revision in 2005 but this only added a few minor changes. [5]
Verilog was one of the first modern hardware description languages to be invented. It is one of the two most common Hardware Description Languages (HDL) used by integrated circuit (IC) designers.

2.1.2 Levels of Abstraction

HDLs allows the design to be simulated earlier in the design cycle in order to experiment with different architectures or correct errors. Designs described in HDL are easy to design and debug, technology-independent, and are usually more readable than schematics, particularly for large circuits. Verilog can be used to describe designs at four levels of abstraction:

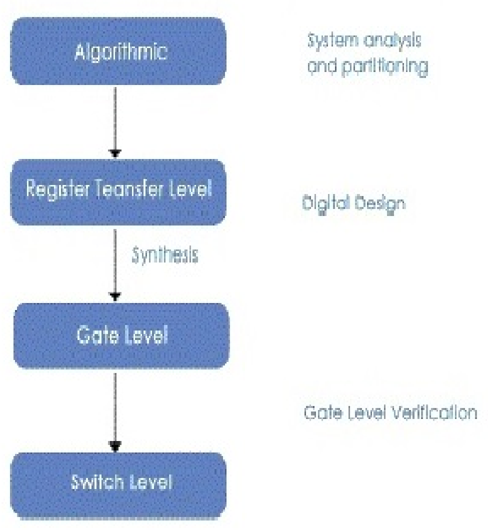


Figure 2.1: Verilog: Levels of Abstraction

- Algorithmic level (much like c code with if-else, case and loop statements).
- Register transfer level (RTL uses registers connected by Boolean equations).
- Gate level (interconnected AND, OR etc.).
- Switch level (MOS transistors)

More recently Verilog is used as an input for synthesis programs that generate a gate-level description (a netlist) for the circuit. Some Verilog constructs are not synthesizable. Also the way the code is written will greatly affect the size and speed of the synthesized circuit. Most readers will want to synthesize their circuits, so non-synthesizable constructs are used only for test benches. These are program modules used to generate I/O needed to simulate the rest of the design. The words not synthesizable is used for examples and constructs as needed that do not synthesize.

2.1.3 Benefits

- Early design verification via high level design verification
- Evaluation of alternative architectures
- Top-down design (w/synthesis)
- Reduced risk to project due to design errors
- Design capture (w/synthesis & independence of implementation media)
- Reduced design/development time & cost (w/synthesis)
- Base line testing of lower level design representations - example: gate level or register level design
- Ability to manage/develop complex designs
- Hardware/software co-design
- Documentation of design (depends on quality of designer comments)

2.1.4 Designer concerns

- Loss of control of detailed design
- Synthesis is inefficient
- Quality of synthesis varies between synthesis tools
- Synthesized logic does not perform the same as the HDL
- Learning curve associated with HDLs & synthesis tools

2.2 Design Flow using Verilog

The diagram below summarises the high level design flow for an FPGA or ASIC. In a practical design situation, each step described in the below sections may be split into several smaller steps, and parts of the design flow will be iterated as errors are uncovered.

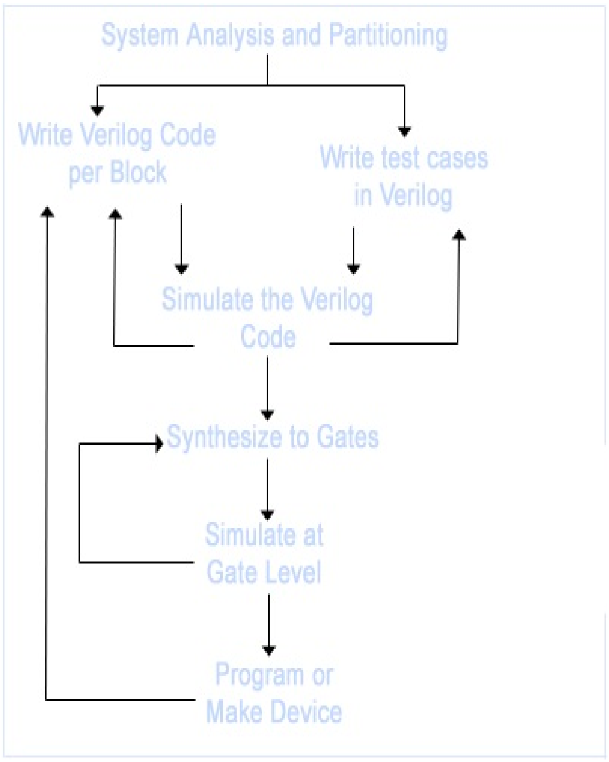

System Analysis and Partitioning
Write Verilog Code per Block
Write test cases in Verilog
Simulate the Verilog Code
Synthesize to Gates
Simulate at Gate Level
Program or Make Device

Figure 2.2: Desin Flow Using Verilog

2.2.1 System-level Verification

As a first step, Verilog may be used to model and simulate aspects of the complete system containing one or more ASICs or FPGAs. This may be a fully functional description of the system allowing the specification to be validated prior to commencing detailed design. Alternatively, this may be a partial description that abstracts certain properties of the system, such as a performance model to detect system performance bottle-necks.

2.2.2 RTL design and testbench creation

Once the overall system architecture and partitioning is stable, the detailed design of each FPGA or ASIC can commence. This starts by capturing the design in Verilog at the register transfer level, and capturing a set of test cases in Verilog.

These two tasks are complementary, and are sometimes performed by different design teams in isolation to ensure that the specification is correctly interpreted. The RTL Verilog should be synthesizable if automatic logic synthesis is to be used. Test case generation is a major task that requires a disciplined approach and much engineering ingenuity: the quality of the final FPGA or ASIC depends on the coverage of these test cases.
For today's large, complex designs, verification can be a real bottleneck. This provides another motivation for System Verilog - it has features for expediting testbench development. See the System Verilog section of Knowhow for more details.

2.2.3 RTL verification

The RTL Verilog is then simulated to validate the functionality against the specification. RTL simulation is usually one or two orders of magnitude faster than gate level simulation, and experience has shown that this speed-up is best exploited by doing more simulation, not spending less time on simulation. In practice it is common to spend 70-80% of the design cycle writing and simulating Verilog at and above the register transfer level, and 20-30% of the time synthesizing and verifying the gates.

2.2.4 Look-ahead Synthesis

Although some exploratory synthesis will be done early on in the design process, to provide accurate speed and area data to aid in the evaluation of architectural decisions and to check the engineer's understanding of how the Verilog will be synthesized, the main synthesis production run is deferred until functional simulation is complete. It is pointless to invest a lot of time and effort in synthesis until the functionality of the design is validated.

2.3 Scope of Verilog

Verilog can be used at different levels of abstraction as we have already seen. But how useful are these different levels of abstraction when it comes to using Verilog?

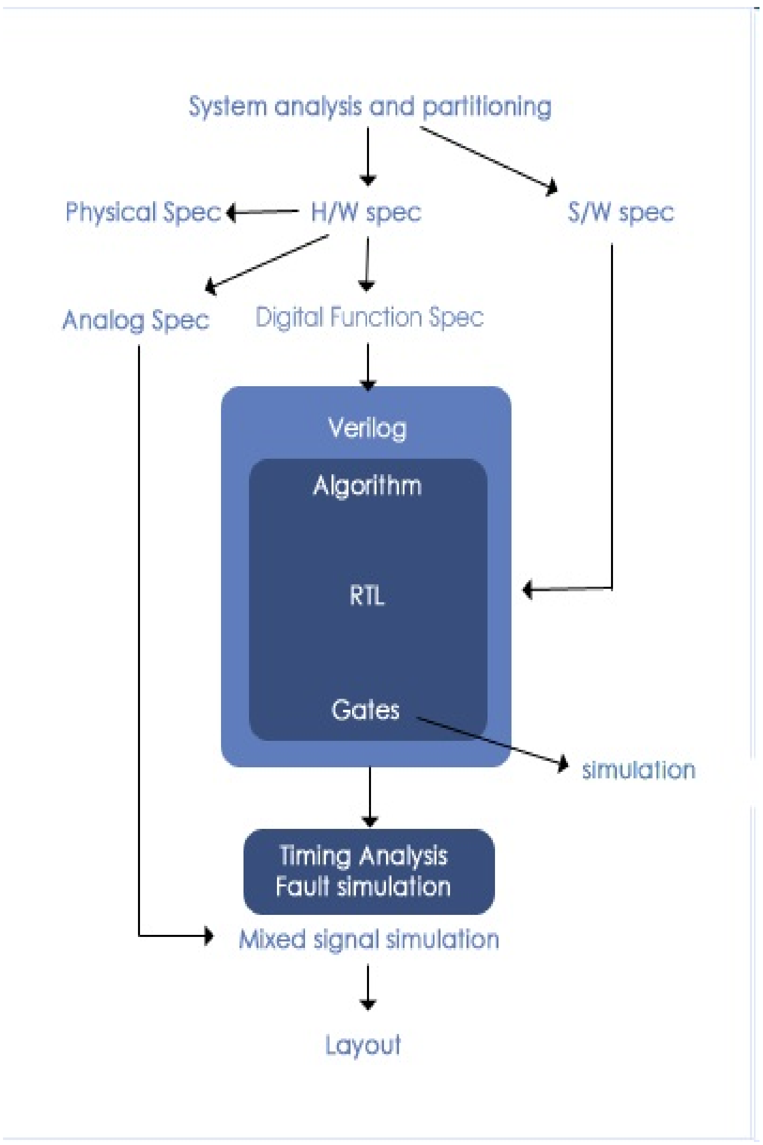


Figure 2.3: Scope of Verilog

– **Design process**

  The diagram below shows a very simplified view of the electronic system design process incorporating Verilog. The central portion of the diagram shows the parts of the design process which will be impacted by Verilog.

– **System level**

  Verilog is not ideally suited for abstract system-level simulation, prior to the hardware-software split. This is to some extent addressed by SystemVerilog. Unlike VHDL, which has support for user-defined types and overloaded operators which allow the designer to abstract his work into the domain of the problem, Verilog restricts the designer to working with pre-defined system functions and tasks for stochastic simulation and can be used for modelling performance, throughput and queueing but only in so far as those built-in langauge features allow. Designers occasionally use the stochastic level of abstraction for this phase of the design process.

– **Digital**

  Verilog is suitable for use today in the digital hardware design process, from functional simulation, manual design and logic synthesis down to gate-level simulation. Verilog tools provide an integrated design environment in this area.

  Verilog is also suited for specialized implementation-level design verification tools such as fault simulation, switch level simulation and worst case timing simulation. Verilog can be used to simulate gate level fanout loading effects and routing delays through the import of SDF files.

  The RTL level of abstraction is used for functional simulation prior to synthesis. The gate level of abstraction exists post-synthesis but this level of abstraction is not often created by the designer, it is a level of abstraction adopted by the EDA tools (synthesis and timing analysis, for example).

– **Analog**

  Because of Verilog's flexibility as a programming language, it has been stretched to handle analog simulation in limited cases. There is a draft standard Verilog-AMS that addresses analog and mixed signal simulation.

Chapter 3

Logic Synthesis

3.1 Introduction

The introduction of logic synthesis for HDLs pushed HDLs from the background into the foreground of digital design. Synthesis tools compiled HDL source files (written in a constrained format called RTL) into a manufacturable netlist description in terms of gates and transistors. Writing synthesizable RTL files required practice and discipline on the part of the designer; compared to a traditional schematic layout, synthesized RTL netlists were almost always larger in area and slower in performance.
How do you write good synthesisable Verilog code to give you the hardware you want? Synthesis is a broad term often used to describe very different tools. Synthesis can include silicon compilers and function generators used by ASIC vendors to produce regular RAM and ROM type structures. This is best suited to gate arrays and programmable devices such FPGAs.
Synthesis is not a panacea! It is vital to tackle High Level Design using Verilog with realistic expectations of synthesis. The definition of Verilog for simulation is cast in stone and enshrined in the Language Reference Manual. Other tools which use Verilog, such as synthesis, will make their own interpretation of the

Verilog language. There is an IEEE standard for Verilog synthesis (IEEE Std. 1364.1-2002) but no vendor adheres strictly to it.
It is not sufficient that the Verilog is functionally correct; it must be written in such a way that it directs the synthesis tool to generate good hardware, and moreover, the Verilog must be matched to the idiosyncrasies of the particular synthesis tool being used. We shall tackle some of these idiosyncracies in this Verilog tutorial.
There are currently three kinds of synthesis:

– behavioural synthesis
– high-level synthesis
– RTL synthesis

There is some overlap between these three synthesis domains. We will concentrate on RTL synthesis, which is by far the most common. The essence of RTL code is that operations described in Verilog are tied to particular clock cycles. The synthesised netlist exhibits the same clock-by-clock cycle behaviour, allowing the RTL testbench to be easily re-used for gate-level simulation.

3.2 Synthesis tools targeting ASICs

– Design Compiler by Synopsys
– Encounter RTL Compiler by Cadence Design Systems
– BuildGates, an older product by Cadence Design Systems, humorously named after Bill Gates
– HDL Designer by Mentor Graphics
– TalusDesign by Magma Design Automation
– RealTime Designer by Oasys Design Systems
– BooleDozer: Logic synthesis tool by IBM (internal IBM EDA tool)

3.3 Synthesis tools targeting FPGAs

- XST (delivered within ISE) by Xilinx
- Quartus II integrated Synthesis by Altera
- IspLever by Lattice Semiconductor
- Encounter RTL Compiler by Cadence Design Systems
- LeonardoSpectrum and Precision (RTL / Physical) by Mentor Graphics
- Synplify (PRO / Premier) by Synopsys
- BlastFPGA by Magma Design Automation

Chapter 4

Memory

4.1 INTRODUCTION TO MEMORIES

A device or an electrical circuit used to store a single bit (0 or 1) is called a memory cell. Examples of memory cell are flip flop, a charged capacitor etc. Semiconductor memories are capable of storing large amount of Digital information. The amount of memory required in a particular system depends on the type of application but the no. of transistor required for storage of data are always much larger than the no of transistors used for logic operations & other purposes.
The ever increasing demand of high storage capacity has driven the fabrication technology & memory development towards more compact design rules and consequently towards higher data storage densities. The memory data storage capacity of a single chip doubles almost after every two years. The number of data bits stored per unit area is one of the key criteria that determine the overall storage capacity & hence the memory cost per bit. Another important issue is the memory access time i.e. the time taken to store or retrieve data in the memory array. The access time determines the memory speed. Static & Dynamic Power consumption of the memory array is a significant factor to

be considered in the design because of the increasing demand of the low power applications.

4.2 TYPES OF MEMORY

The Semiconductor memories are classified according to the type of data storage and data access.

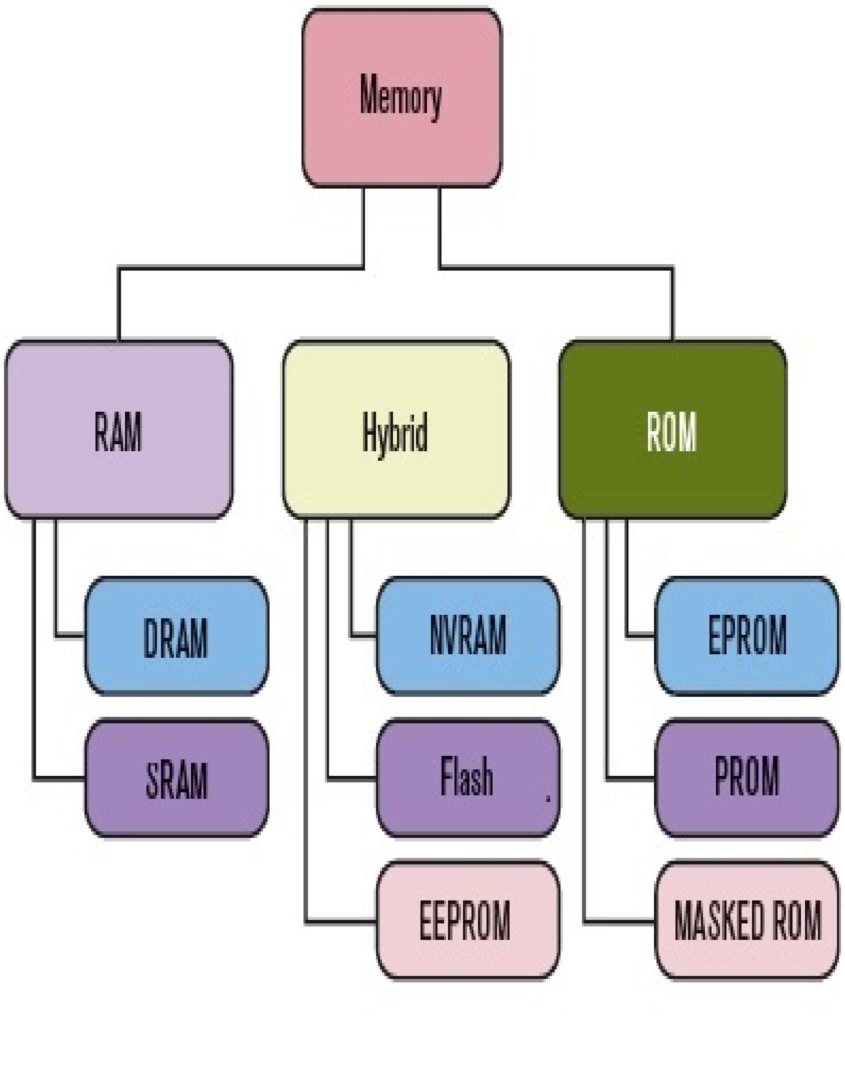


Figure 4.1: Types of Memory

4.2.1 ROM

These are read only memory . ROM are non volatile memories that is the stored data is not lost even when the power supply is not switched off and refresh operation is not required. Memories in the ROM family are distinguished by the methods used to write new data to them (usually called programming), and the number of times they can be rewritten. This classification reflects the evolution of ROM devices from hardwired to programmable to erasable-and-programmable. The ROM are classified into three main categories:

a. PROM

   These are programmable ROM in which data is written electrically after the chip is fabricated. The device programmer writes data to the device one word at a time by applying an electrical charge to the input pins of the chip. Once a PROM has been programmed in this way, its contents can never be changed. If the code or data stored in the PROM must be changed, the current device must be discarded. As a result, PROMs are also known as one-time programmable (OTP) devices.

b. EPROM

   An EPROM (erasable-and-programmable ROM) is programmed in exactly the same manner as a PROM. However, EPROMs can be erased and reprogrammed repeatedly. To erase an EPROM, you simply expose the device to a strong source of ultraviolet light. (A window in the top of the device allows the light to reach the silicon.) By doing this, you essentially reset the entire chip to its initial–unprogrammed–state. Though more expensive than PROMs, their ability to be reprogrammed makes EPROMs an essential part of the software development and testing process.

c. MASKED ROM

   These are ROM in which data is written during chip fabrication by using a photo mask. The contents of the ROM had to be specified before chip production, so the actual data could be used to arrange the transistors inside the chip. These are also known as hard wired memories. The primary advantage of a masked ROM is its low production cost. Unfortunately, the cost is low only when large quantities of the same ROM are required.

4.2.2 RAM

The Read/Write memory is commonly known as Random Access Memory (RAM). Read/Write(R/W) memory must permit the modification (writing) of data bits stored in the memory array, as well as their retrieval (reading) on demand. .Unlike sequential access memory any cell can be accessed with nearly equal access time. The stored data is volatile i.e. the stored data is lost when the power supply is switched off. RAMs are classified into two main categories :

a. DRAM

   Dynamic Random Access memory (DRAM) cells consist of capacitor to store binary information , 1(high voltage) or 0(low voltage) ,and transistor to access the capacitor . Cell information (voltage) is degraded mostly due to the junction leakage current at the storage node. Therefore, a cell data must be read and re-written periodically (refresh operation) even when memory arrays are not access. The entire memory area is divided into several blocks where the row & column decoders are shared by neighboring blocks. Each memory block is sliced into sub-blocks, where each sub block has its data line control circuits. The number of cells per word & bit lines is determined by the trade-off between the chip size & performance. The Chip size is smaller if more cells can share the same word line & bit lines. However the performance is degraded in this case as row & column address decoders must drive a large load. In dynamic RAM cell data is stored as charge on a capacitor & the presence or absence of stored charge determines the value of the stored bit. Since the data stored as charge on a capacitor cannot be retained indefinitely because the leakage currents eventually remove or modify the stored data. Thus all the DRAM cells requires a periodic refreshing so that the unwanted modifications due to leakage can be prevented. The usage of a capacitor as the primary storage device generally enables the DRAM cell to be realized in a much smaller area as compared to the typical SRAM cell.

b. SRAM

Static Random access memory (SRAM) is used when we require high speed because SRAM devices offer extremely fast access times (approximately four times faster than DRAM) but are much more expensive to produce. Generally, SRAM is used only where access speed is extremely important. Rest all about SRAM is explained in next chapter

4.2.3 HYBRID

As memory technology has matured in recent years, the line between RAM and ROM has blurred. Now, several types of memory combine features of both. These devices do not belong to either group and can be collectively referred to as hybrid memory devices. Hybrid memories can be read and written as desired, like RAM, but maintain their contents without electrical power, just like ROM. Two of the hybrid devices, EEPROM and flash, are descendants of ROM devices. These are typically used to store code. The third hybrid, NVRAM, is a modified version of SRAM. NVRAM usually holds persistent data.

a. EEPROM

EEPROMs are electrically-erasable-and-programmable. Internally, they are similar to EPROMs, but the erase operation is accomplished electrically, rather than by exposure to ultraviolet light. Any byte within an

EEPROM may be erased and rewritten. Once written, the new data will remain in the device forever or at least until it is electrically erased. The primary trade off for this improved functionality is higher cost, though write cycles are also significantly longer than writes to a RAM. So you wouldn't want to use an EEPROM for your main system memory.

b. FLASH RAM

Flash memory combines the best features of the memory devices described thus far. Flash memory devices are high density, low cost, nonvolatile, fast (to read, but not to write), and electrically reprogrammable. These advantages are overwhelming and, as a direct result, the use of flash memory has increased dramatically in embedded systems. From a software viewpoint, flash and EEPROM technologies are very similar. The major difference is that flash devices can only be erased one sector at a time, not byte-by-byte. Typical sector sizes are in the range 256 bytes to 16KB. Despite this disadvantage, flash is much more popular than EEPROM and is rapidly displacing many of the ROM devices as well.

c. NVRAM

The third member of the hybrid memory class is NVRAM (non-volatile RAM). Nonvolatility is also a characteristic of the ROM and hybrid memories discussed previously. However, an NVRAM is physically very different from those devices. An NVRAM is usually just an SRAM with a battery backup. When the power is turned on, the NVRAM operates just like any other SRAM. When the power is turned off, the NVRAM draws just enough power from the battery to retain its data. NVRAM is fairly common in embedded systems. However, it is expensive–even more expensive than SRAM, because of the battery–so its applications are typically limited to the storage of a few hundred bytes of system-critical information that can't be stored in any better way.

Chapter 5

Why do we need Models.?

We have studies about mathematical models or statistical models of some systems or some theory. But why do we use those models was a big question that time. So here I explain you the reason.
Generally speaking, models are simplified descriptions of reality that strip away all of its complexity except for a few features thought to be critical to the understanding of the phenomenon under study. Mathematical models are such descriptions translated into a very precise language which, unlike natural human languages, does not allow for any double (or triple) meanings. The great strength of mathematics is that, once we have framed a problem in mathematical language, we can deduce precisely what are the consequences of the assumptions we made – no more, no less.
Models can be used for a variety of purposes: a compact description of the system structure, an investigation into the logical coherence of the proposed explanation, and derivation of specific predictions from theory that can be tested with data. Depending on the purpose, we can develop different models for the same empirical system.
Similarly, in VLSI industries, HDL Models are used for logical/functional coherence of the proposed explanation and derivation of specific predictions that can

be tested with data.Also, note that, HDL Models are the Replication of your Design functionality. Models should be developed such that its functionality should act like more closed to Silicon.
In ST Microelectronic, my work is to make these Models with provided design functionalities and specifications. I have done the Modelling of Memory IPs with Verilog HDL.

Chapter 6

HDL Based Memory Models

In ST Microelectronics, I updated a previous version of model with adding some functionalities and some strategies. We will go through these Functionalities thoroughly.

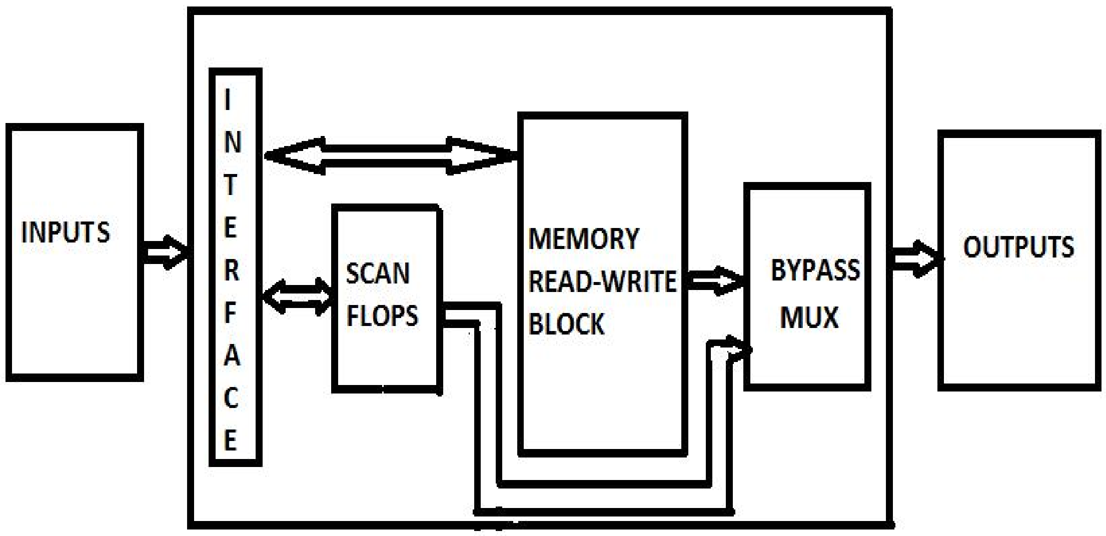


Figure 6.1: Memory Model Block Diagram

The functionalities which were supported in Memory models are,

- Read/write
- Redundancy
- Masking
- E-switch functionality
- Power supply checks
- Margin Controls
- Retention Logic

6.1 Read/Write

Static Random access memory (SRAM) is used when we require high speed because SRAM devices offer extremely fast access times (approximately four times faster than DRAM) but are much more expensive to produce. Generally, SRAM is used only where access speed is extremely important. The timing diagrams of read cycle and write cycles are shown below:

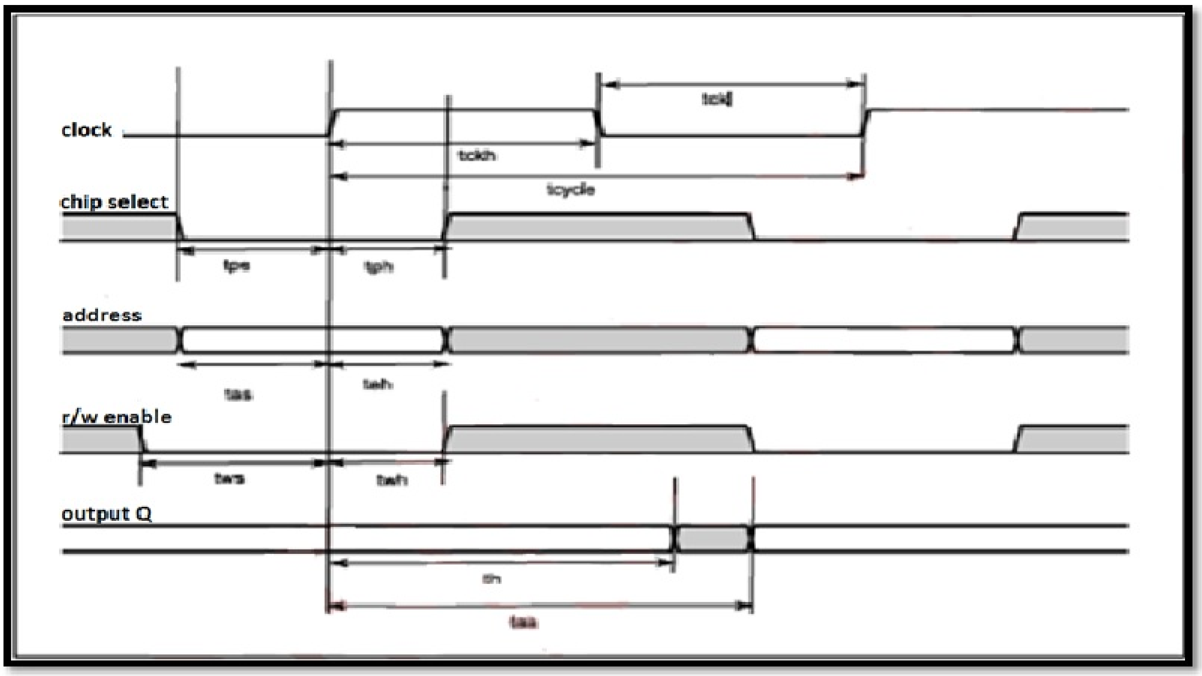


Figure 6.2: Read cycle

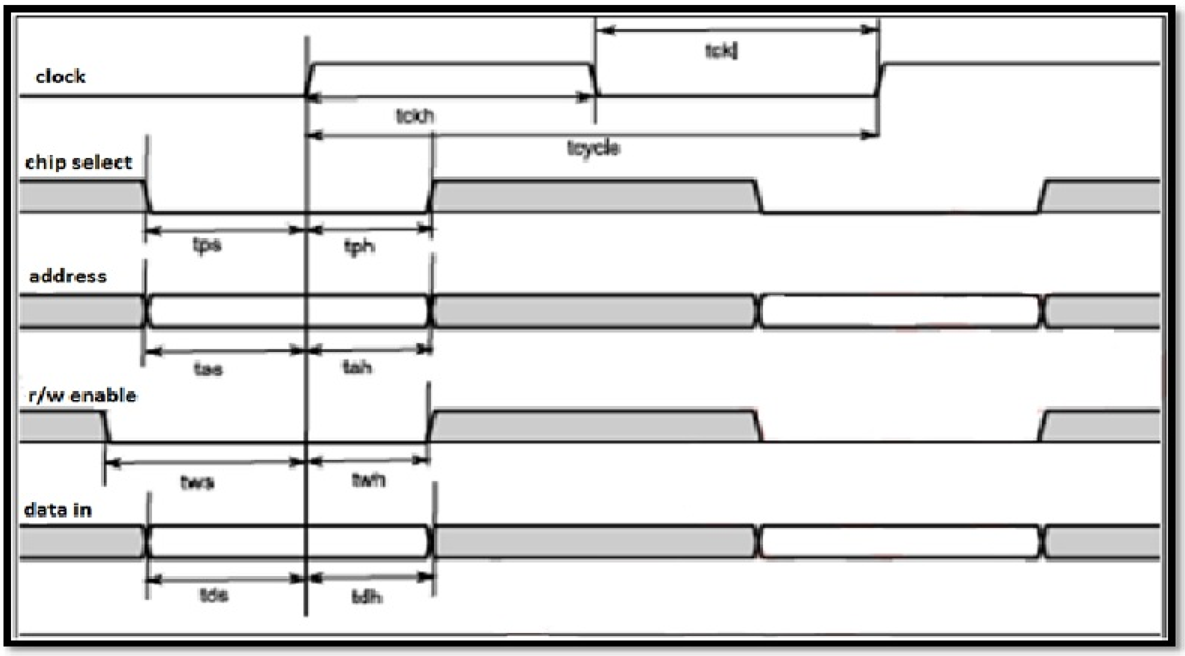


Figure 6.3: Write cycle without mask

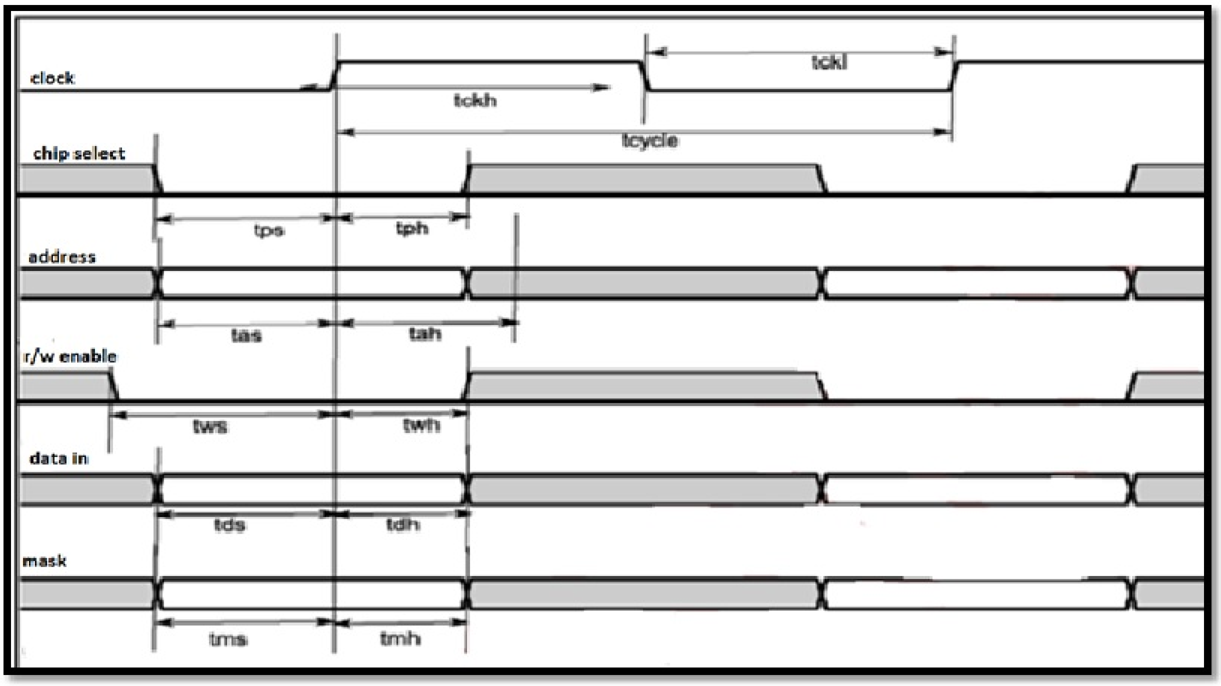


Figure 6.4: Write cycle with mask

6.2 Redundancy

In large memory arrays (where in SRAM is being used), there are possibilities of corruption of memory arrays /memory cells. If any memory cell is corrupted in between the array, the current technology replaces the entire array which doubles the cost and reduces the throughput. To overcome this problem redundancy technique is proposed.

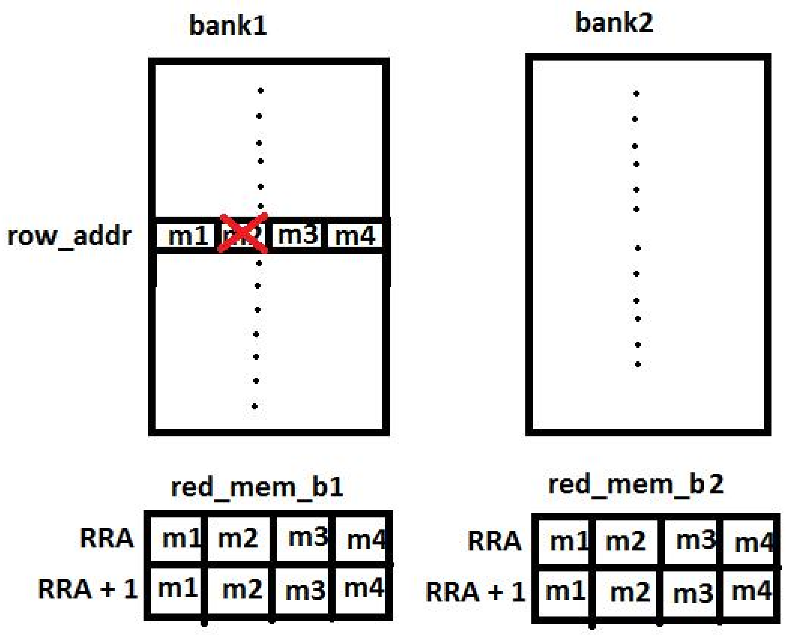


Figure 6.5: Redundancy operation of Model

The basic idea of the technique is to replace the row or column containing the corrupted bit cell with a redundant row or column. So the memory array will be built along with the redundant blocks (rows or column). But one should see to it that the total cost of SRAM array with redundant blocks and additional features should not exceed the double the cost of a single SRAM array.

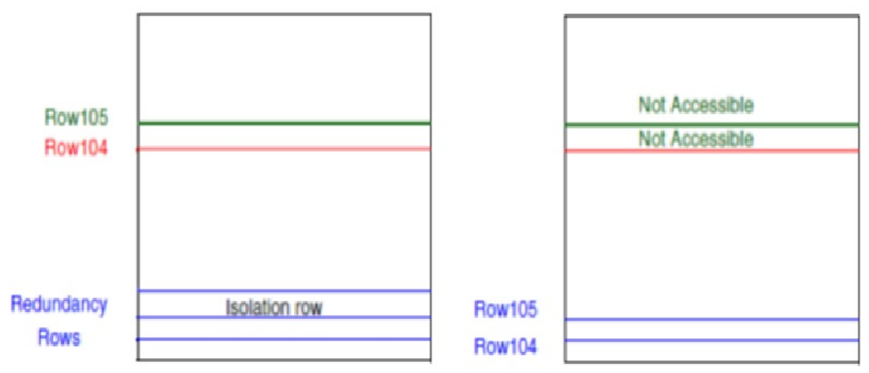


Figure 6.6: Redundancy Concept

For example,
the above fig 6.6 shows the cut with a single failing row 104 on left, and the same cut after repair, on the right. Note that row 105 is also replaced along with row 104. Thus, if during testing, a failure is found on row number 104, to repair the memory redundancy repair address should be put as 104 (binary 1101000) on redundancy repair address, and redundancy repair address enable should be pulled high. This ensures that the failing row is never accessed; redundancy row replaces it instead.

6.3 Masking

The Bit Mask feature enables a user to preserve the contents of particular bits/bytes in a word when a new Write operation is initiated. When a memory is generated with Mask feature, then every data input is accompanied by a mask input. When mask input is high during a Write operation, the corresponding bit/byte of the memory location is not updated with the new data input. When Mask pin is asserted, the corresponding bit/byte does not enter into Write mode and dynamic power consumption is reduced.

For example, consider the following case for memory size 16x8, when the address is 5 and my previous value at that memory location be 8b 1111 1000, then the value that should go to the memory location 5 during write operation with masking is obtained below:

Mem[5]	8'b 1111 1000
Data	8'b 0101 0110
Mask	8'b 1011 1101
Value	8'b 1111 1010

Thus, instead of data hex 56 my value at that memory will go as hex FA.

6.4 E-switch

The Eswitch option enables embedded power gating in the compiler. By default, the option is set to No. When the Eswitch option is set to Retention, both array and periphery can be switched Off or On independently. Separate switch controls are available for turning the periphery and array off, with two external pins each for array and periphery to control a small and a large internal switch. The small switch is turned On before the large switch. This ensures low peak currents so that adjoining circuits sharing the same power mesh are not impacted when memory wake up is in progress.

6.5 Power Supply

A designer may decide to use separate substrate supplies in Split Bulk Supply mode. For the periphery, these two are different power supplies, that is, substrate power is vddsmp and source power is vddmp. Ground is split into gndm and gndsm (substrate) for periphery. The substrate supply of memory array is connected to gndm.

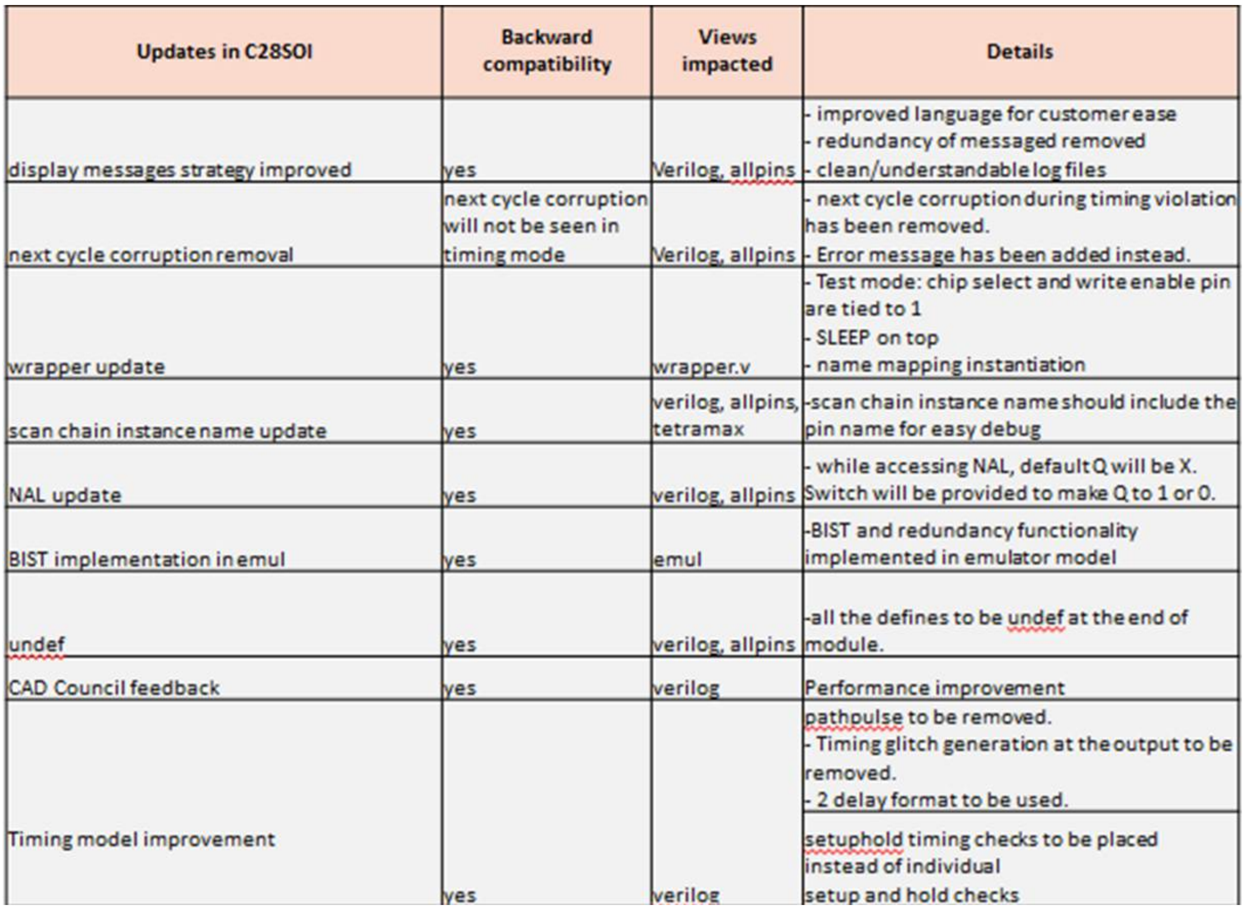
Split Bulk power supply option instances have separate substrate supplies to enable operation in Body Bias mode. Forward body bias is used to improve memory performance for slow lots.

Chapter 7

Work Flow of HDL Modeling

7.1 Updates

Customer reports some updation in terms of,structure of memory models or model specifications & functionality. Figure 7.1 shows the updates from the customers. With reference to these listed updates, previosly defined models are edited in terms of verilog langauge or supported views.The previously defined models are enhanced with the new specs or functionality or structure.After that the updated model is being verified on SV verification environment.



Updates in C28SOI	Backward compatibility	Views impacted	Details
display messages strategy improved	yes	Verilog, allpins	- improved language for customer ease - redundancy of messaged removed - clean/understandable log files
next cycle corruption removal	next cycle corruption will not be seen in timing mode	Verilog, allpins	- next cycle corruption during timing violation has been removed. - Error message has been added instead.
wrapper update	yes	wrapper.v	- Test mode: chip select and write enable pin are tied to 1 - SLEEP on top - name mapping instantiation
scan chain instance name update	yes	verilog, allpins, tetramax	-scan chain instance name should include the pin name for easy debug
NAL update	yes	verilog, allpins	- while accessing NAL, default Q will be X. Switch will be provided to make Q to 1 or 0.
BIST implementation in emul	yes	emul	-BIST and redundancy functionality implemented in emulator model
undef	yes	verilog, allpins	-all the defines to be undef at the end of module.
CAD Council feedback	yes	verilog	Performance improvement
Timing model improvement	yes	verilog	pathpulse to be removed. - Timing glitch generation at the output to be removed. - 2 delay format to be used. setuphold timing checks to be placed instead of individual setup and hold checks

Figure 7.1: List of updates

7.2 Bug finding

In Modelling, we first write the HDL code. That HDL code is written with considering all the functionalities that the model should represent or the customer wants in the model/IPs.

When the HDL code is written, we do take care of

- Some instances and its co-dependence
- Priority / preference of occurring of events
- Dependence of one functionality on others
- Modes to be included in HDL model

- Groups of timing checks i.e. the group of timing checks are described in terms of stability time windows and the group to check clock and control signals and are described in terms of the difference in time between two events.

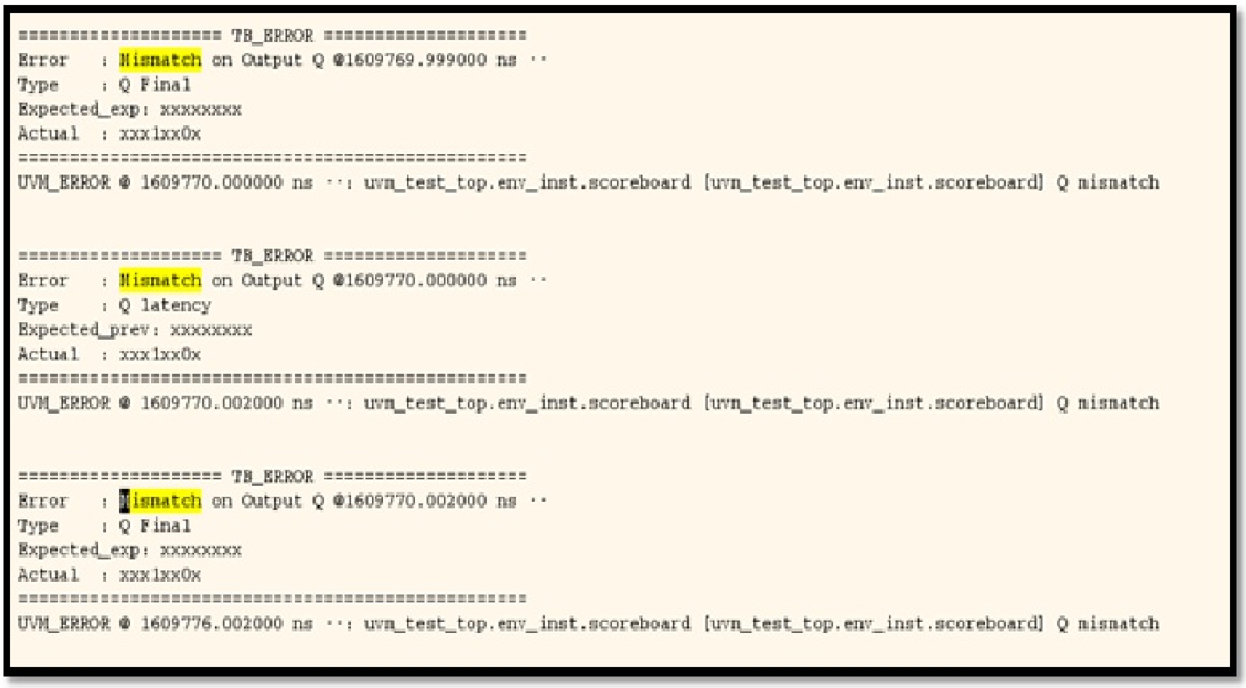
- Supply Checks and priority of supply togglings priority

- Task handling

After considering all the above points, the Model Developer should verify whether the model is in sync with the design specifications or not. For that "Verification" of the written RTL is done.

In ST Microelectronics, we use an "SV ENVIRONMENT" that is preparatory of the Verification Team to verify the Model. This environment is updated with the design specifications. So, when the model is run on the environment, it compares the expected golden response with the model output responds. When the mismatch occurs, we can say that, there is a BUG present in the HDL model (actually its a Verilog code only). Finding the "Presence of the BUG" is easy but finding "where and why the BUG is...!" is difficult. After the mismatch comes in the log file, we open up the Simulation window of the respected case. With putting all the functionality in a human mind, the developer should check the mismatch that the simulation tool shows us.

In the below figures 7.2 and 7.3, the screenshot of the log file and the waveform window at that time instant are shown respectively.



```
==================== TB_ERROR ====================
Error    : Mismatch on Output Q @1609769.999000 ns ..
Type     : Q Final
Expected_exp: xxxxxxxx
Actual   : xxx1xx0x
==================================================
UVM_ERROR @ 1609770.000000 ns ..: uvm_test_top.env_inst.scoreboard [uvm_test_top.env_inst.scoreboard] Q mismatch

==================== TB_ERROR ====================
Error    : Mismatch on Output Q @1609770.000000 ns ..
Type     : Q latency
Expected_prev: xxxxxxxx
Actual   : xxx1xx0x
==================================================
UVM_ERROR @ 1609770.002000 ns ..: uvm_test_top.env_inst.scoreboard [uvm_test_top.env_inst.scoreboard] Q mismatch

==================== TB_ERROR ====================
Error    : Mismatch on Output Q @1609770.002000 ns ..
Type     : Q Final
Expected_exp: xxxxxxxx
Actual   : xxx1xx0x
==================================================
UVM_ERROR @ 1609776.002000 ns ..: uvm_test_top.env_inst.scoreboard [uvm_test_top.env_inst.scoreboard] Q mismatch
```

Figure 7.2: Log file view

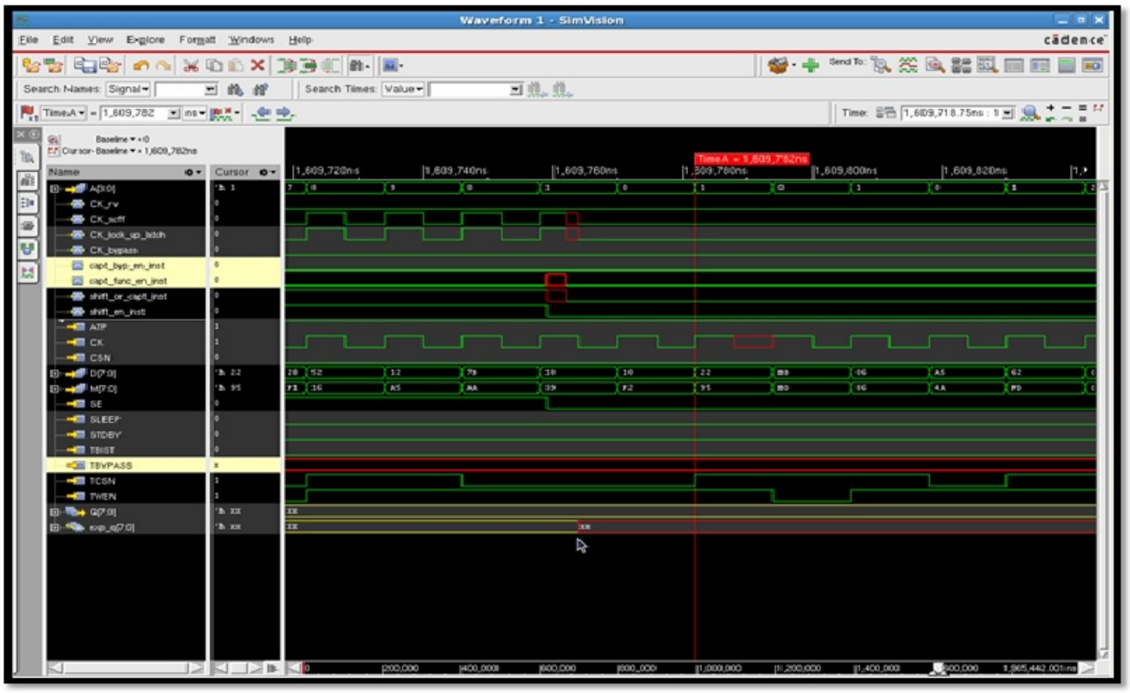
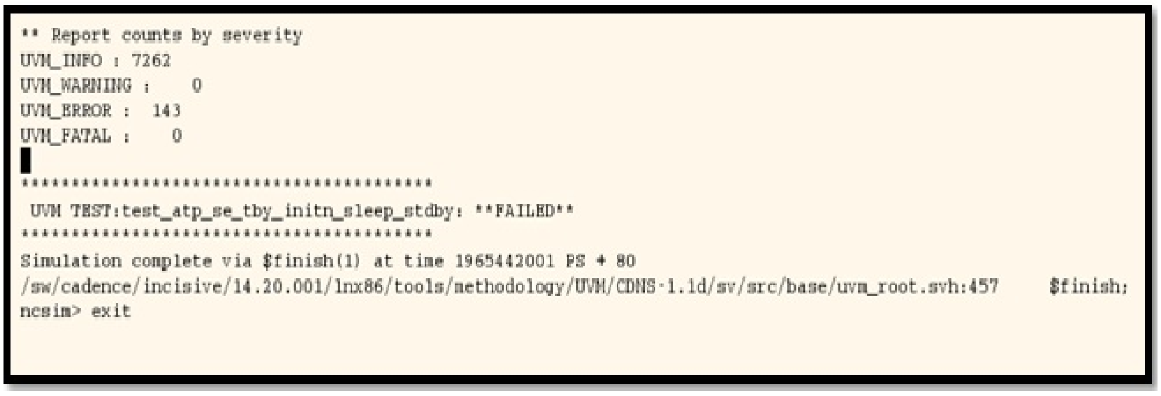


Figure 7.3: Waveform view

In test case atp_se_tby_initn_sleep_stdby there are 143 errors and thus that test case failed in the SV environment run. Generated report:



```
** Report counts by severity
UVM_INFO : 7262
UVM_WARNING :    0
UVM_ERROR :  143
UVM_FATAL :    0
█
****************************************
 UVM TEST:test_atp_se_tby_initn_sleep_stdby: **FAILED**
****************************************
Simulation complete via $finish(1) at time 1965442001 PS + 80
/sw/cadence/incisive/14.20.001/lnx86/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457     $finish;
ncsim> exit
```

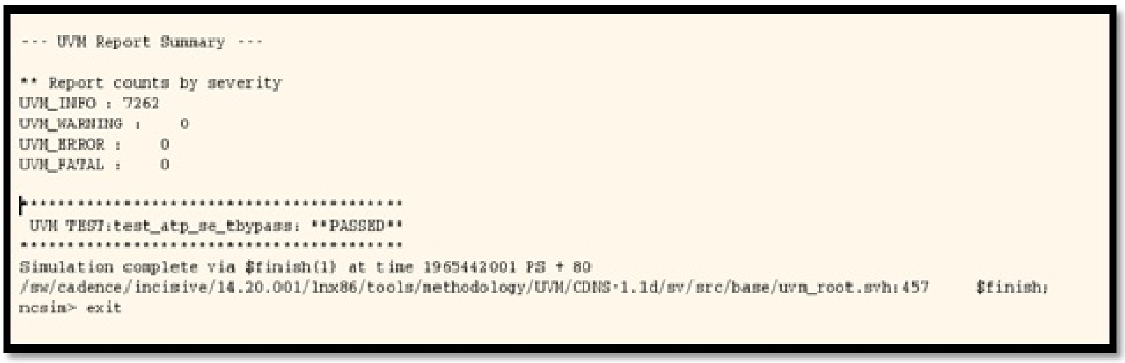
Figure 7.4: Failed testcase

7.3 Debugging

When we find the presence of the bug, and where the big is, we need to do the DEBUGGING process on our Verilog code.

So the developer again goes to the flow of the code and with respect to the mismatch occurred, the test case conditions and the defined functionality, debugging of the code is done.

Here in the below fig 7.5 the screenshot of the log file, for the same test case.



```
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO : 7262
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0

****************************************
 UVM TEST:test_atp_se_tbypass: **PASSED**
****************************************
Simulation complete via $finish(1) at time 1965442001 PS + 80
/sw/cadence/incisive/14.20.001/lnx86/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457     $finish;
ncsim> exit
```

Figure 7.5: Passed testcase

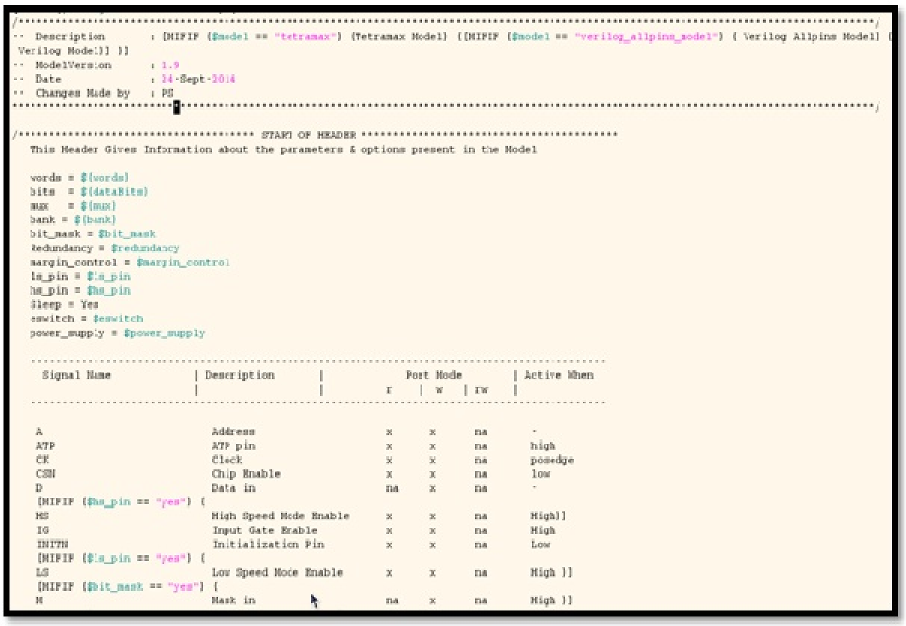
"After passing all the BUG FINDING & DEBUGGING the failed code becomes the successive one".

7.4 Model Template File Update

The model developer writes a code for a small memory cut i.e for the lesser memory size so that the length of the code do not make the bug finding and debugging process a tedious one. Thus, the first cut I have referred was of 16x8 in which Address is 16 and each address have 8 bits of contain.
But after the HDL code becomes the successive code, i.e if it matches the design specific functionalities for a smaller cut, then that flow of that HDL code should be made generic so that it gives the same successive approach for the larger cuts too.
Also, the handling of functionalities should be in the hand of user/customer. It should not be a hard coded one. If a customer wants a memory code in which the Redundancy functionality should not be present, then it becomes unworthy to remove all the redundancy related terms from the hard-coded HDL code.
Thus, a Template file is written in MIF codes, that will handle the functionalities as well as the terms of HDL related to it, like, the number of repeated instances of a sub-module.
MIF - Map Into Format file, is a file extension for an interchange format file format. MIF files contain representations of the text and layout construction in grouped statements. Due to its text nature, an MIF file can be read and parsed easily.
In ST Microelectronics, I have updated the MIF file i.e a template file of the updated code. After updating it, that file is also compiled and checked whether there is any syntax error or not! And then a script that will contain the information of the required cuts functionalities & features like, the memory cut size, address bits, data bits, name of the compiler supported, the name of the .v file,

model type i.e allpins model (contains the supply pins on the input ports) or Verilog model, the presence of the required functionalities & also the absence of the non-required functionalities.

Here, is screen shot of the MIF File,

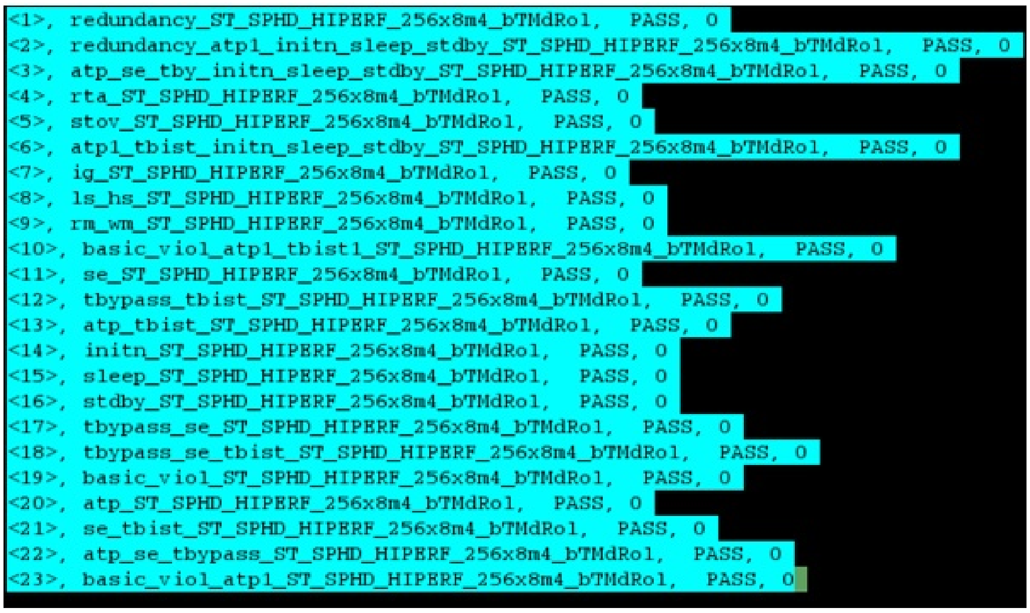


```
/*****************************************************************************************************************/
--  Description        : [MIFIF ($model == "tetramax") (Tetramax Model] [[MIFIF ($model == "verilog_allpins_model") ( Verilog Allpins Model] [
 Verilog Model]] ]]
--  ModelVersion       : 1.9
--  Date               : 24-Sept-2014
--  Changes Made by    : PS
*****************************************************************************************************************/

/************************************** START OF HEADER ******************************************
   This Header Gives Information about the parameters & options present in the Model

   words = ${words}
   bits  = ${dataBits}
   mux   = ${mux}
   bank = ${bank}
   bit_mask = $bit_mask
   Redundancy = $redundancy
   margin_control = $margin_control
   ls_pin = $ls_pin
   hs_pin = $hs_pin
   Sleep = Yes
   eswitch = $eswitch
   power_supply = $power_supply

   ------------------------------------------------------------------------------------------------
    Signal Name              | Description         |              Port Mode         | Active When
                             |                     |         r     |  w    | rw      |
   ------------------------------------------------------------------------------------------------

    A                          Address                   x       x       na        -
    ATP                        ATP pin                   x       x       na        high
    CK                         Clock                     x       x       na        posedge
    CSN                        Chip Enable               x       x       na        low
    D                          Data in                   na      x       na        -
    [MIFIF ($hs_pin == "yes") (
    HS                         High Speed Mode Enable    x       x       na        High]]
    IG                         Input Gate Enable         x       x       na        High
    INITN                      Initialization Pin        x       x       na        Low
    [MIFIF ($ls_pin == "yes") (
    LS                         Low Speed Mode Enable     x       x       na        High ]]
    [MIFIF ($bit_mask == "yes"] (
    M                          Mask in                   na      x       na        High ]]
```

Figure 7.6: Template view

When the script is run successfully the output will be the HDL code for that memory cut. But the generated file is also being verified at the developer side. Sometime it is also possible that the generated file also have some errors that will be seen in the verification end.

If the developed code is checked for all the possible cuts and the possible modes i.e. timing mode or functional mode, the MIF file will be considered as the next updated MODEL of the Memory. Thus it is important that the generated cut should pass all the test cases, below fig. 7.7 shows the report of a Timing mode in which file initialization is done without Fault.



```
<1>, redundancy_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<2>, redundancy_atp1_initn_sleep_stdby_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<3>, atp_se_tby_initn_sleep_stdby_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<4>, rta_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<5>, stov_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<6>, atp1_tbist_initn_sleep_stdby_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<7>, ig_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<8>, ls_hs_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<9>, rm_wm_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<10>, basic_viol_atp1_tbist1_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<11>, se_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<12>, tbypass_tbist_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<13>, atp_tbist_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<14>, initn_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<15>, sleep_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<16>, stdby_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<17>, tbypass_se_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<18>, tbypass_se_tbist_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<19>, basic_viol_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<20>, atp_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<21>, se_tbist_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<22>, atp_se_tbypass_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
<23>, basic_viol_atp1_ST_SPHD_HIPERF_256x8m4_bTMdRol,  PASS, 0
```

Figure 7.7: Final Report with all passed test cases

Thus, the updated model is ready to release.

Chapter 8

Work done during the project

- Understanding of Behavioral Modeling of Single Port Memory previously defined models.
- Simulation on NCSim, QuestaSIM and VCSMX simulators.
- Writing synthesible verilog RTL code for emulator models.
- Spyglass tool for RTL checks.
- Design_Compiler for synthesis.
- TetraMAX Automatic Test Patterns Generation.
- Above mentioned steps for, ROM model, pseudo-REG models, and Dual Port REG models.
- Emulation on VELOCE tool for SP, ROM, pREG & DP emulator-models.

Chapter 9

msmCutGen: Auto-Cut Generation Tool

9.1 Introduction

As we have seen before, updation of memory involves some steps to be followed:

- Template updation
- Cut generation
- Verification
- Validation

The cut generation step converts the MIF formatted file into the actual verilog coded files. along with this verilog view, other supported views are also geneated. The generation step involves, many deliverable products which provides the respective background supports.
The compilers supported for a perticular memory model, are submitted on UPT(Unicad Project Tracking). Today Cut generation process is error prone due to many manual interventions. Also, it was Difficult for newcomers. Some-

times it introduce bugs in products due to missing cuts/missing compilers. and because of these errors, its run time was large due to iterations.

steps involved in Old process:

- Look up / findout compilers supported and PRIMARY names
- manually create dirs for all compilers supported by a model product. (sometimes not all compilers are created.)
- Download products required for the primaries are listed
- Decide on the cutlist and get it reviewed by PL (Risky)
- Generate Cuts (manual generation commands. + environment- variable settings can be sometimes be incorrect.)

Thus, a script was developed to automate the process. and steps involved in new process:

- User only gives the Model name and version on which he is working
- Automatic lookup of PRIMARIES and compilers supported
- Auto Dir structure created for all compilers
- Auto cutlist generated, with controllable combinations.

9.2 Flow of msmCutGen

The flow if msmCutGen is shown in figure 9.1

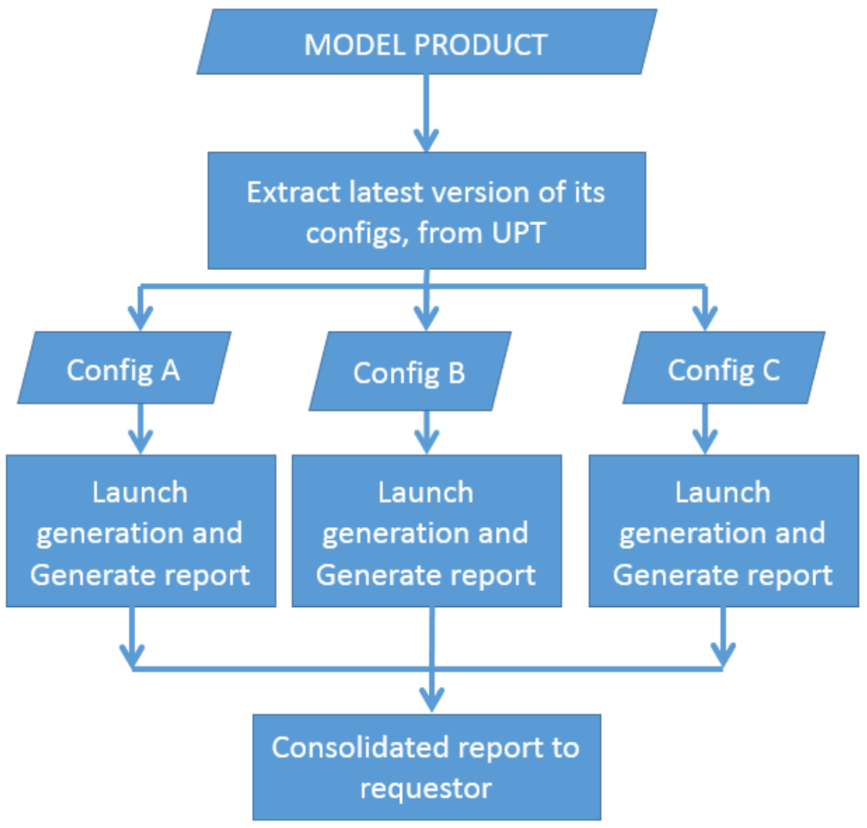

MODEL PRODUCT
Extract latest version of its configs, from UPT
Config A
Config B
Config C
Launch generation and Generate report
Launch generation and Generate report
Launch generation and Generate report
Consolidated report to requestor

Figure 9.1: msmCutGen Flow

9.3 Features

The msmCutGenkit offers :

- Extraction of latest version of compilers supported by given model product which user wants to validate from UPT.
- Allows user to add compilers which are not on UPT but wants to run the generation for it.
- Allows to select the compilers for which generation should run.
- Allows to select the compilers for which generation should run.
- Allows to give already existing ucdprod, .ucdprod and ugnGuiSetupDB paths as an input for any of the compilers.

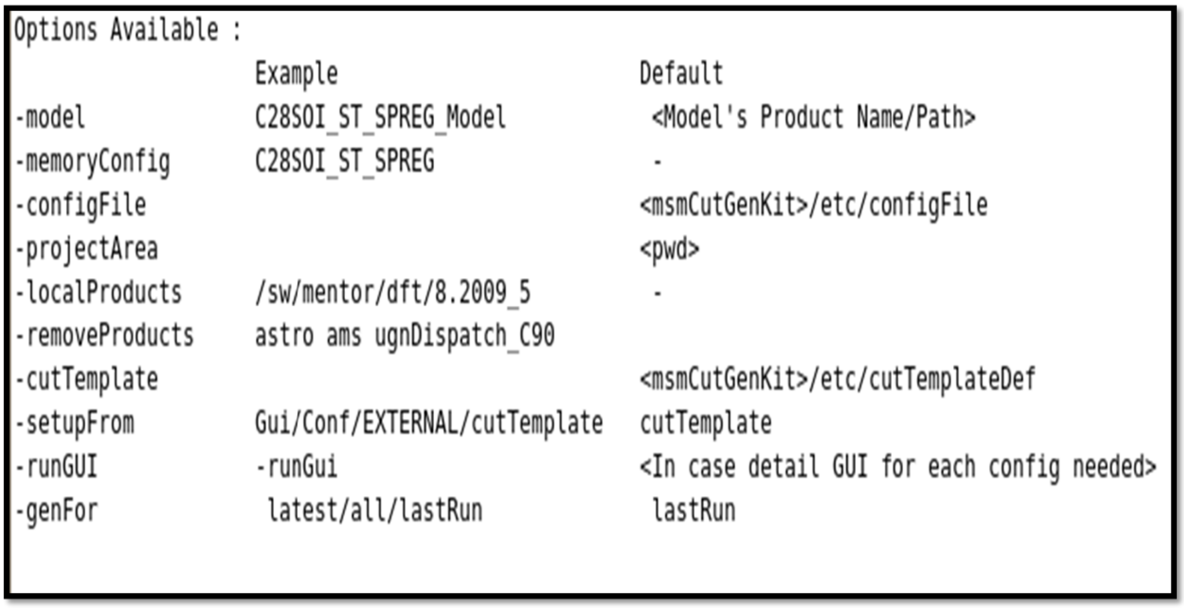
– Exhaustive cutlist generation for compilers based on given cut template.

– Runs the cut generation in complete batch mode.

– Generates summarised and detailed report for cut generation done.

9.4 Procedure

Provided options with msmCutGen command are as shown in below fig. 9.2



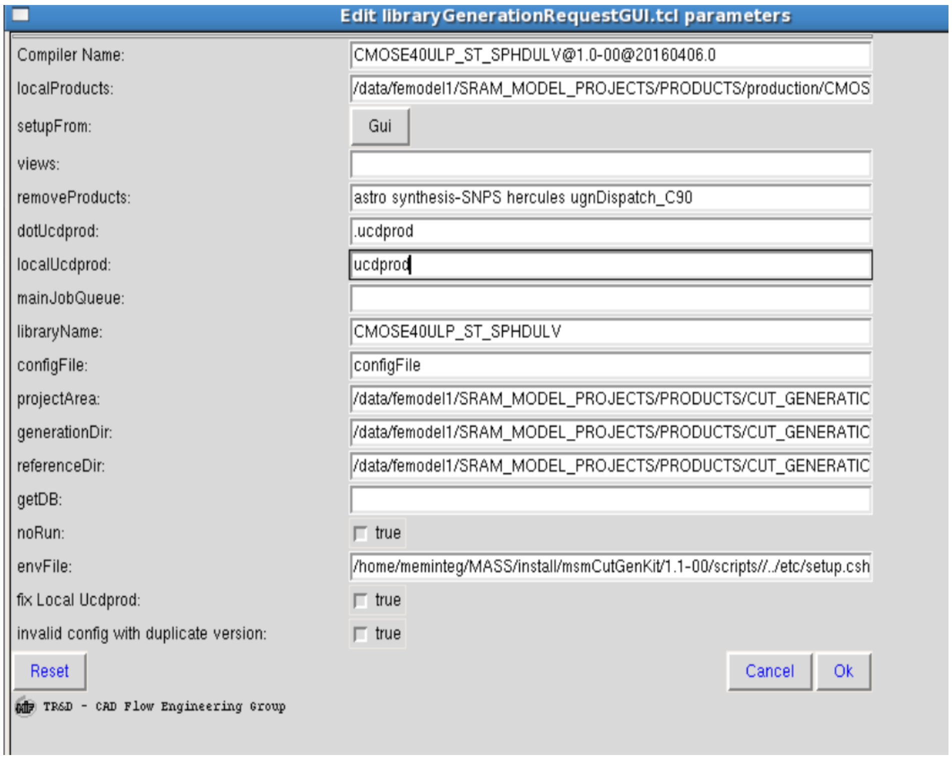
```
Options Available :
                   Example                         Default
-model             C28SOI_ST_SPREG_Model            <Model's Product Name/Path>
-memoryConfig      C28SOI_ST_SPREG                  -
-configFile                                        <msmCutGenKit>/etc/configFile
-projectArea                                       <pwd>
-localProducts     /sw/mentor/dft/8.2009_5          -
-removeProducts    astro ams ugnDispatch_C90
-cutTemplate                                       <msmCutGenKit>/etc/cutTemplateDef
-setupFrom         Gui/Conf/EXTERNAL/cutTemplate   cutTemplate
-runGUI            -runGui                         <In case detail GUI for each config needed>
-genFor             latest/all/lastRun              lastRun
```

Figure 9.2: msmCutGen: Options

– Extracts the latest version of compilers from UPT supported by given model product.

– Pops up a GUI window for selection of configs on which generation should run.

– If user wants to give already existing ucdprod, .ucdprod and ugnGuiSetupDB paths as an input for any of the selected compilers then another GUI window as shown in below fig. 9.3 pops up to enter the inputs.



Edit libraryGenerationRequestGUI.tcl parameters

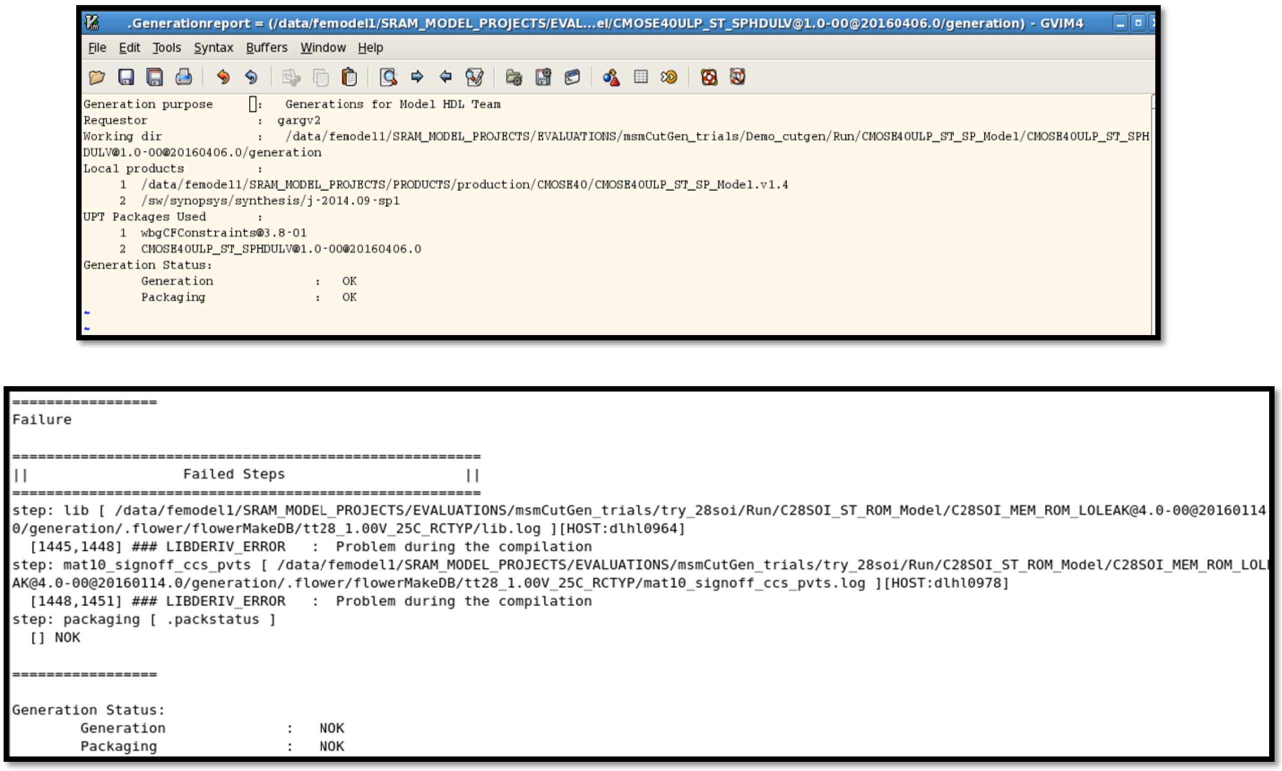
Compiler Name:	CMOSE40ULP_ST_SPHDULV@1.0-00@20160406.0
localProducts:	/data/femodel1/SRAM_MODEL_PROJECTS/PRODUCTS/production/CMOS
setupFrom:	Gui
views:	
removeProducts:	astro synthesis-SNPS hercules ugnDispatch_C90
dotUcdprod:	.ucdprod
localUcdprod:	ucdprod
mainJobQueue:	
libraryName:	CMOSE40ULP_ST_SPHDULV
configFile:	configFile
projectArea:	/data/femodel1/SRAM_MODEL_PROJECTS/PRODUCTS/CUT_GENERATIC
generationDir:	/data/femodel1/SRAM_MODEL_PROJECTS/PRODUCTS/CUT_GENERATIC
referenceDir:	/data/femodel1/SRAM_MODEL_PROJECTS/PRODUCTS/CUT_GENERATIC
getDB:	
noRun:	☐ true
envFile:	/home/meminteg/MASS/install/msmCutGenKit/1.1-00/scripts//../etc/setup.csh
fix Local Ucdprod:	☐ true
invalid config with duplicate version:	☐ true

Figure 9.3: runGUI window

– After having complete inputs of all the compilers, then each of the compilers generation is launched on LSF in parallel to each other.

– In mid of the whole process, for ugnGuiSetupDB if cut template is given, then exhaustive cutlist is also generated automatically.

– Generates the report of generations launched and mails to the user.


.Generationreport = (/data/femodel1/SRAM_MODEL_PROJECTS/EVAL...el/CMOSE40ULP_ST_SPHDULV@1.0-00@20160406.0/generation) - GVIM4
File
Edit
Tools
Syntax
Buffers
Window
Help

```
Generation purpose      :   Generations for Model HDL Team
Requestor               :  gargv2
Working dir             :   /data/femodel1/SRAM_MODEL_PROJECTS/EVALUATIONS/msmCutGen_trials/Demo_cutgen/Run/CMOSE40ULP_ST_SP_Model/CMOSE40ULP_ST_SPH
DULV@1.0-00@20160406.0/generation
Local products          :
     1  /data/femodel1/SRAM_MODEL_PROJECTS/PRODUCTS/production/CMOSE40/CMOSE40ULP_ST_SP_Model.v1.4
     2  /sw/synopsys/synthesis/j-2014.09-sp1
UPT Packages Used       :
     1  wbgCFConstraints@3.8-01
     2  CMOSE40ULP_ST_SPHDULV@1.0-00@20160406.0
Generation Status:
        Generation              :   OK
        Packaging               :   OK
~
~
```

```
=================
Failure

=========================================================
||                   Failed Steps                      ||
=========================================================
step: lib [ /data/femodel1/SRAM_MODEL_PROJECTS/EVALUATIONS/msmCutGen_trials/try_28soi/Run/C28SOI_ST_ROM_Model/C28SOI_MEM_ROM_LOLEAK@4.0-00@20160114
0/generation/.flower/flowerMakeDB/tt28_1.00V_25C_RCTYP/lib.log ][HOST:dlhl0964]
  [1445,1448] ### LIBDERIV_ERROR   :  Problem during the compilation
step: mat10_signoff_ccs_pvts [ /data/femodel1/SRAM_MODEL_PROJECTS/EVALUATIONS/msmCutGen_trials/try_28soi/Run/C28SOI_ST_ROM_Model/C28SOI_MEM_ROM_LOL
AK@4.0-00@20160114.0/generation/.flower/flowerMakeDB/tt28_1.00V_25C_RCTYP/mat10_signoff_ccs_pvts.log ][HOST:dlhl0978]
  [1448,1451] ### LIBDERIV_ERROR   :  Problem during the compilation
step: packaging [ .packstatus ]
  [] NOK

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

Generation Status:
        Generation              :   NOK
        Packaging               :   NOK
```

Figure 9.4: Generated reports

Chapter 10

Conclusion

Front-end Modeling is an important activity for all projects. Hence, this needs to be continuously updated to deliver state-of-the-art solutions to our customers. The need for modeling to resemble the actual hardware design keeps on increasing with time, as designers want to know exactly how a particular IP will behave when it is plugged into a SoC (System on Chip).

Moreover, nowadays with shrinking technology, the numbers of views delivered for same IP are also increasing. Earlier it was mainly to do with functional and timing model. Now it includes ATPG views, emulation view, Equivalence Checker view etc. Hence, Front-End modeling becomes all the more complex.

The working Memory model is released to customer and the customer can generate the required size and the different views can also be generated.

References

[1] Samir Palnitkar, ”Verilog HDL: A Guide to Digital Design and Synthesis” Second edition, Prentice Hall PTR, February 21, 2003

[2] *http : //only − vlsi.blogspot.in/*

[3] H. Yalcin, M. Mortazavi, R. Palermo, C. Bamji, and K. Sakallah, ”Functional timing analysis for IP characterization” , In DAC, pages 731-736, 1999.

[4] *www.vlsi − expert.com*

[5] *https : //www.doulos.com/knowhow/verilog_designers_guide/*

[6] *http : //vlsibyjim.blogspot.in/*

[7] Product specs of C28SOI SP Model of STMicroelectronics.

[8] Product specs of C28SOI ROM Model of STMicroelectronics.

[9] Product specs of CMOSE40 DPREG Model of STMicroelectronics.