SoC Development Containing the Teststructures for SiVal of Memories/IO's/Efuses/Standard Cells

Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

In

Electronics & Communication Engineering

(VLSI Design)

By

Gaurav A. Lalani

(12MECV15)

Department of Electronics & Communication Engineering

Institute of Technology

Nirma University

Ahmedabad-382 481

Dec 2013

SoC Development Containing the Teststructures for SiVal of Memories/IO's/Efuses/Standard Cells

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

In

Electronics & Communication Engineering

(VLSI Design)

By

Gaurav A. Lalani

(12MECV15)

Under the Internal Guidance of

Dr. N.P.Gajjar

and

External Guidance of

Mr. Parvez Zaman

Department of Electronics & Communication Engineering

Institute of Technology

Nirma University

Ahmedabad-382 481

Declaration

This is to certify that

1. The thesis comprises of 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.

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

Gaurav A. Lalani

CERTIFICATE

This is to certify that the Major Project entitled **"SoC Development Containing the Teststructures for SiVal of Memories/IO's/Efuses/Standard Cells"** submitted by **Mr. Gaurav A. Lalani (12MECV15)**, towards the partial fulfillment of the requirements for the degree of **Master of Technology in VLSI Design** of **Nirma University of Science and Technology; Ahmedabad** is the record of work carried out by him under our supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for the 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.

External Guide

Mr. Parvez Zaman
Manager
TSO
Freescale India Pvt. Ltd.

Internal Guide

Dr. N.P.Gajjar
Sr. Associate Professor
VLSI Design
Nirma University

PG Co-ordinator

Dr. N. M. Devashrayee
VLSI Design

HOD

Dr. P. N. Tekwani
Professor, EE

Director

Dr. K Kotecha
Director, IT-NU

Date: _____

Place: Ahmedabad

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 TSO team, especially the Test Vehicle Group(TVG) team, 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 Group manager **Mr. Parvez Zaman** ,my immediate supervisors **Mr. Nitin Dhamija** and **Mr. Puneet Sharma** 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. Ketan Kotecha** (Director, Nirma University, Ahmedabad) for his continuous guidance, support and enthusiasm. I would take this opportunity to thank **Dr. P. N. Tekwani** (Head of Department, Electrical Engineering), **Dr. N. M. Devashrayee** (Professor and Program Coordinator, M.Tech - EC (VLSI Design)), Internal Guide **Dr. N.P.Gajjar** (Sr. Associate Professor, M.Tech - EC (VLSI Design)) 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, Freescale Semiconductor India Ltd. I am equally thankful to **Freescale Semiconductor** for providing me the invaluable exposure to the industry and the current market trends.

Finally, I would like to express my heartfelt thanks to my parents and colleagues for their blessings and for their constant love and support.

Gaurav Lalani A.

(12MECV15)

Abstract

Systems-on-chip (SoCs) has become reality now, driven by fast development of CMOS VLSI technologies. Complex system integration onto one single die introduces a set of various challenges and perspectives for industrial and academic institutions. Important issues to be addressed here are cost-effective technologies, efficient and application-tailored hardware/software architectures, and corresponding IP-based EDA methods. Due to exponentially increasing CMOS mask costs, essential aspects for the industry are now adaptivity of SoCs, which can be realized by integrating reconfigurable re-usable hardware parts on different granularities into configurable systems-on-chip (CSoCs).

SoC development involves combining different IP's in a single integrated circuit(chip). SoC development infers the advantages like lower cost,re-usability, high performance, high reliability, and reduced chip size. The design complexity and density of a SoC is very high. The design complexity can be reduced by partitioning the system into hardware and software based on functionality. Modelling at higher level of abstractions also helps in reducing the issue of high complexity to manageable level.

The SoC containing test-structures for memories, IO's, Efuse and standard cells is used as test chip for silicon validation. These test-structures containing reticle is used as a reference for development of such structures in a particular process node. The advancement to a particular lower process node will be carried out only if the reticle containing test-structures is successfully implemented. As from front-end point of view the verification of such test-structures is very important and should be thoroughly performed. Several testcases are written to verify the functionality of each module/structure.

Freescale Semiconductor At A Glance

- A world leader in providing the semiconductor solutions that help our customers improve quality of life for everyone, both today and in the future.
- Freescale Semiconductor, Inc. is an American company that produces and designs embedded hardware, with 17 billion semiconductor chips in use around the world. The company focuses on the automotive, consumer, industrial and networking markets with its product portfolio including microprocessors, microcontrollers, digital signal processors, digital signal controllers, sensors, RF power ICs and power management ICs. In addition, the company offers software and development tools to support product development.
- Among the world's largest semiconductor companies
- A leading technology innovator
- A pioneer and visionary leader in sustainability
- President and CEO: Gregg Lowe
- The company is headquartered in Austin, Texas with design, research and development, manufacturing and sales operations in more than 20 countries.
- Freescale is currently ranked 7th among the semiconductor sales leaders in the United States and is ranked 16th worldwide.

- The company also holds an extensive patent portfolio, including approximately 6,100 patent families.
- Freescale filed to go public on February 11, 2011 and completed its IPO on May 26, 2011. Freescale is traded on the New York Stock Exchange under the ticker symbol FSL.
- Freescale is a leader in embedded processing solutions for the automotive, consumer, industrial and networking markets. From microcontrollers and microprocessors to sensors, analog ICs and connectivity, our technologies are fueling the next great wave of innovation.
- Earlier known as MOTOROLA and later in 2004 as FREESCALE.

Contents

Declaration iii

Certificate iv

Acknowledgement v

Abstract vi

Freescale Semiconductor At A Glance vii

List of Figures xiii

1 Introduction 1

1.1 **Overview Of the Flow** 2

1.1.1 Specifications 2

1.1.2 IP Delivery 2

1.1.3 Top Level Netlist Delivery 2

1.1.4 Initial Constraints Delivery 2

1.1.5 Silicon Virtual Prototyping 3

1.1.6 Floorplanning 3

1.1.7 Chip Level Power Grid Design 3

1.1.8 RTL Verification 4

1.1.9 RTL Level Power Estimation 4

1.1.10 Physical Synthesis 4

1.1.11 Prelayout DFT 4
1.1.12 Prelayout STA 4
1.1.13 Clock Tree Synthesis 5
1.1.14 Block Level Routing 5
1.1.15 Top Level Routing 5
1.1.16 Extraction 5
1.1.17 Postlayout STA 6
1.1.18 Functional Noise Analysis 6
1.1.19 Power and IR Drop Analysis 6
1.1.20 Postlayout DFT 6
1.1.21 Postlayout Functional Verification 7
1.1.22 Physical Integration and Verification 7

2 Test Chip Architecture **9**

2.1 Test Chip IP's 9
2.2 Test Chip's Architectural Overview 10
2.3 Standard Cell Core 12
2.3.1 All Cell Block Description 12
2.4 Clock Generation and Control Register 14
2.4.1 Clock Control Register 14
2.5 Reset Generation 15
2.6 Ring Oscillator 16
2.7 Frequency Counter 18
2.8 Memory IP 19
2.9 Static RAM 19
2.9.1 Single Port Random Access Memory (SPRAM) 20
2.9.2 DUAL Port Static Random Access Memory (DPRAM) 21
2.9.2.1 DUAL Port Static Random Access Memory TYPE-1 21
2.9.2.2 DUAL Port Static Random Access Memory TYPE-2 22

3 Front End Integration **24**

3.1 Introduction 24

3.2 Deliverables to Front-end Integration 25

3.3 Deliverables from Front-end Integration 25

3.4 Flow 26

3.4.1 Phase 0 26

3.4.2 Phase 1 26

3.5 Functional Verification 28

3.5.1 Introduction 28

3.5.1.1 Deliverables to Verification Team 28

3.5.1.2 Deliverables from Verification Team 28

3.5.2 The Flow 28

3.5.2.1 Phase 0: 29

3.5.2.2 Phase 1: 29

3.5.2.3 Phase 2: 29

4 Synthesis **30**

4.1 Introduction 30

4.2 RC Synthesis Flow 30

4.2.1 Prerequisites for Synthesis with RC 30

4.2.2 What Tool does during Synthesis 31

4.2.3 Area-based Optimization 33

4.2.4 Timing-based Optimization 35

4.2.5 Special Care abouts and Recommendations 36

5 Configuring SDC **40**

5.1 Specifying Clocks 40

5.2 Clock Uncertainty 41

5.3 Clock Latency 41

5.4 Generated Clocks 42

5.5 Typical Clock Generation Scenario 44
5.6 Timing Path Groups 45
5.7 Virtual Clocks 45
5.8 Refining the Timing Analysis 46
5.8.1 Specifying Inactive Signals 47
5.8.2 Breaking Timing Arcs in Cells 48
5.8.3 False Paths 49
5.8.4 Multicycle Paths 50

6 Results **52**
6.1 Results 52

7 Conclusion **57**
7.1 Tasks handled 57
7.2 Conclusion 58

References **59**

List of Figures

1.1 SoC Design Flow 8

2.1 All Cell Core 12
2.2 Standard cell Layout - Inverter 13
2.3 Frequency Counter Block Diagram 18
2.4 SINGLE PORT SRAM 21
2.5 DUAL PORT RAM TYPE - 1 22
2.6 DUAL PORT RAM TYPE - 2 22

3.1 Front End integration/Verification Flow 27

5.1 Clock Latency 42
5.2 Clock distribution in a typical ASIC 44
5.3 Path Groups 45
5.4 Virtual Clock Example 46
5.5 Selecting Clock mode for timing analysis 48
5.6 disable_timing example 49
5.7 Multicycle path example 51

6.1 Top Level Instantiation 52
6.2 Core Module 53
6.3 Core Submodules 53
6.4 Clock Generator Block 54

6.5 All Cell Block 54
6.6 Memory Test and Repair Block 55
6.7 Memory Instance 55
6.8 Synthesis Summary Report 56

Chapter 1

Introduction

Over the past ten years, as integrated circuits became increasingly more complex and expensive, the industry began to embrace new design and reuse methodologies that are collectively referred to as system-on-chip (SoC) design. The reusable components, called intellectual property (IP) blocks or cores, are typically synthesizable register-transfer level (RTL) designs (often called soft cores) or layout level designs (often called hard cores). The concept of reuse can be carried out at the block, platform, or chip levels, and involves making the IP sufficiently general, configurable, or programmable, for use in a wide range of applications. Verification issues must be addressed when integrating reusable components.

The purpose of this section is to briefly explain the typical SoC design flow in vogue for 130 and 90nm high performance designs. Individual functional tasks are explained in greater detail in the sections that follow. This section identifies the various functional tasks and data transfer processes among them.

1.1 Overview Of the Flow

1.1.1 Specifications

A product is born out of necessity or requirement. A potential customer identifies a set of requirements, which, if and when endorsed by the marketing team (of the vendor), is called an L1 specification. The systems team then swings into action and begins creation of the Architecture or L2 specification which identifies IPs and protocols that should belong to the envisioned ASIC. Once the L2 specifications are frozen, the design or integration team creates the IC Design Specification Document or the L3 specification, which describes in detail the function of each IP and protocol. The L3 specification is considered as the Bible by every IP designer.

1.1.2 IP Delivery

An IP is essentially an independent unit catering to certain predefined functionality. Therefore, an IP can be reused in multiple SoCs that require the same functionality to be satisfied. An IP owner is responsible for delivering completely verified IPs (in form of RTL/gates) along with data such as stub netlists (consisting of input/output information in verilog format), timing exceptions, verification patterns, timing constraints etc.

1.1.3 Top Level Netlist Delivery

This activity is owned by the Frontend Integration team, that stitches together a verilog top level netlist based on the delivered stub netlists and IC architecture specifications.

1.1.4 Initial Constraints Delivery

The STA team releases initial timing constraints based on the Architecture and Design Specifications and the top level netlist. The constraints are verified for

bottlenecks through an established QTM flow prior to release. These constraints get refined as more accurate data is made available and the design progresses. Timing constraints drive synthesis, clock tree synthesis, placement, routing, floorplan etc. and is one of the most crucial deliverables from the STA team. The delivered constraints need to describe all the clocks in the design.

1.1.5 Silicon Virtual Prototyping

The Virtual Prototyping flow reads in very preliminary data and attempts to estimate issues that might arise later in the design cycle. Therefore, Virtual Prototyping is a very handy tool that can be used by the Physical Integration team to aid them in their floorplanning efforts. The STA team can also prototype the design to refine timing constraints and identify timing bottlenecks very early in the design cycle.

1.1.6 Floorplanning

Floorplanning can commence once a fairly comprehensive top level netlist is available. The chip is partitioned into physical hierarchies based on connectivity, functionality, timing and placement constraints, power and signal integrity requirements. All concerned functional teams need to sign off on the final floorplan and be actively involved at all intermediate stages. The floorplanning team is also responsible for providing frames (with size and pin descriptions) to the Physical Synthesis team.

1.1.7 Chip Level Power Grid Design

This stage is intricately associated with the floorplanning process, and begins as soon as the initial floorplan is ready. The chip level power grid design needs to be signed off once the floorplan is frozen and considered final.

1.1.8 RTL Verification

The IPs are put together and verified on the chip before each RTL is considered final. The verification team is responsible for running typical functional patterns to check for connectivity mismatches or RTL bugs.

1.1.9 RTL Level Power Estimation

Each RTL is also analyzed for power consumption estimates and power reduction opportunities before they are considered frozen.

1.1.10 Physical Synthesis

The physical synthesis flow reads in IP netlists (RTL/Gates), frame definitions, timing and physical libraries and timing constraints to synthesize and place designs concurrently so as to ensure placement-aware synthesis. Parasitic information for the routes is estimated through global routes and a scan-stitched gate level netlist and placement DEF file is dumped out for clock tree synthesis and routing flows.

1.1.11 Prelayout DFT

The gatelevel netlists dumped out from physical synthesis are used to generate DFT patterns and simulated subsequently to get early estimates of DFT issues.

1.1.12 Prelayout STA

The STA process reads in data output from physical synthesis and parasitic information for the top level nets to obtain early estimates of chip level timing issues.

1.1.13 Clock Tree Synthesis

The clocks in the chip need to be balanced (in terms of delay) in accordance with the limits laid down in the timing constraints file. This is one of the most crucial steps in an IC design flow, since clock tree synthesis is fine art of achieving an optimal balance between clock path delay (insertion delay/latency), skew (uncertainty) and clock path active current drain.

1.1.14 Block Level Routing

Once clock tree synthesis is complete, the database is shipped to the router to route the design. Current generation routers need to consider issues like timing, functional and delay noise and signal net electromigration during the routing process. The routing process delivers gate level postlayout netlists, extracted parasitics (in SPEF format) and LEF/DEF files.

1.1.15 Top Level Routing

A frozen floorplan with the power grid laid out can be routed to complete the connections between the various physical hierarchies in the chip, as also the external pins. It is the physical integration teams responsibility to ensure that top level routing does not interfere with block level routing (or vice versa) in any way whatsoever. Routing Blockages (at the block or macro level) are thus specified to minimize or prevent the above from occurring.

1.1.16 Extraction

A routed database is fed to a 2.5D or 3D extraction engine to create a parasitic netlist consisting of distributed RC pi networks. Extraction is performed independently at the block and top levels. The extracted netlist(s) is (are) then used by the STA and Functional Noise Analysis processes to estimate delay and noise glitches

respectively. An R-only network for the power grid is also extracted for the power grid and ground bounce analysis processes.

1.1.17 Postlayout STA

The postlayout gate level netlists and parasitics are read in by the delay calculator and timing analysis tool to check for timing and delay noise violations in the partitioned physical block as well as the full chip. Violations are fixed by passing appropriate directives to the router or placement engine. The STA process is also required to deliver delay numbers, for DFT and Functional simulations, in the industry standard SDF format.

1.1.18 Functional Noise Analysis

This process analyzes the design (both block as well as top level) for noise glitch violations. If found, directives are passed on to the router or placement engine to fix them.

1.1.19 Power and IR Drop Analysis

The power analysis process calculates active and leakage current drain in designs based on connectivity, activity information and interconnect parasitics. The calculated power numbers are then used to analyze the chip power grid to ensure compliance with the IC specifications pertaining to IR drop and technology bounds for current density (electromigration).

1.1.20 Postlayout DFT

DFT simulations are run on the postlayout netlist in presence of delay numbers (in form of an SDF file) delivered by the STA process. Postlayout simulations check for scan chain discontinuities, races, glitches etc.

1.1.21 Postlayout Functional Verification

Chip level and/or system level patterns are analyzed on the postlayout database consisting of gate level netlists and SDF files delivered from STA. Postlayout functional simulation focusses on detecting races, glitches and timing bugs in the design.

1.1.22 Physical Integration and Verification

Once the physical partitions and the top level are complete in isolation, the integration process begins to put them together on the chip. Physical Verification broadly relates to checking for consistency between the layout (GDS format) and the schematic (spice netlists created out of gate level verilog netlists), called LVS (Layout Versus Schematic), and DRC (Design Rule Checking), which addresses geometrical rules that have to be met for the polygons (metal, poly, wells, substrates etc.) so as to ensure zero defects during manufacture in the fabrication facilities. The physical verification step can begin only once the rest of the design processes have been completed. Physical verification is typically the final process before tapeout.

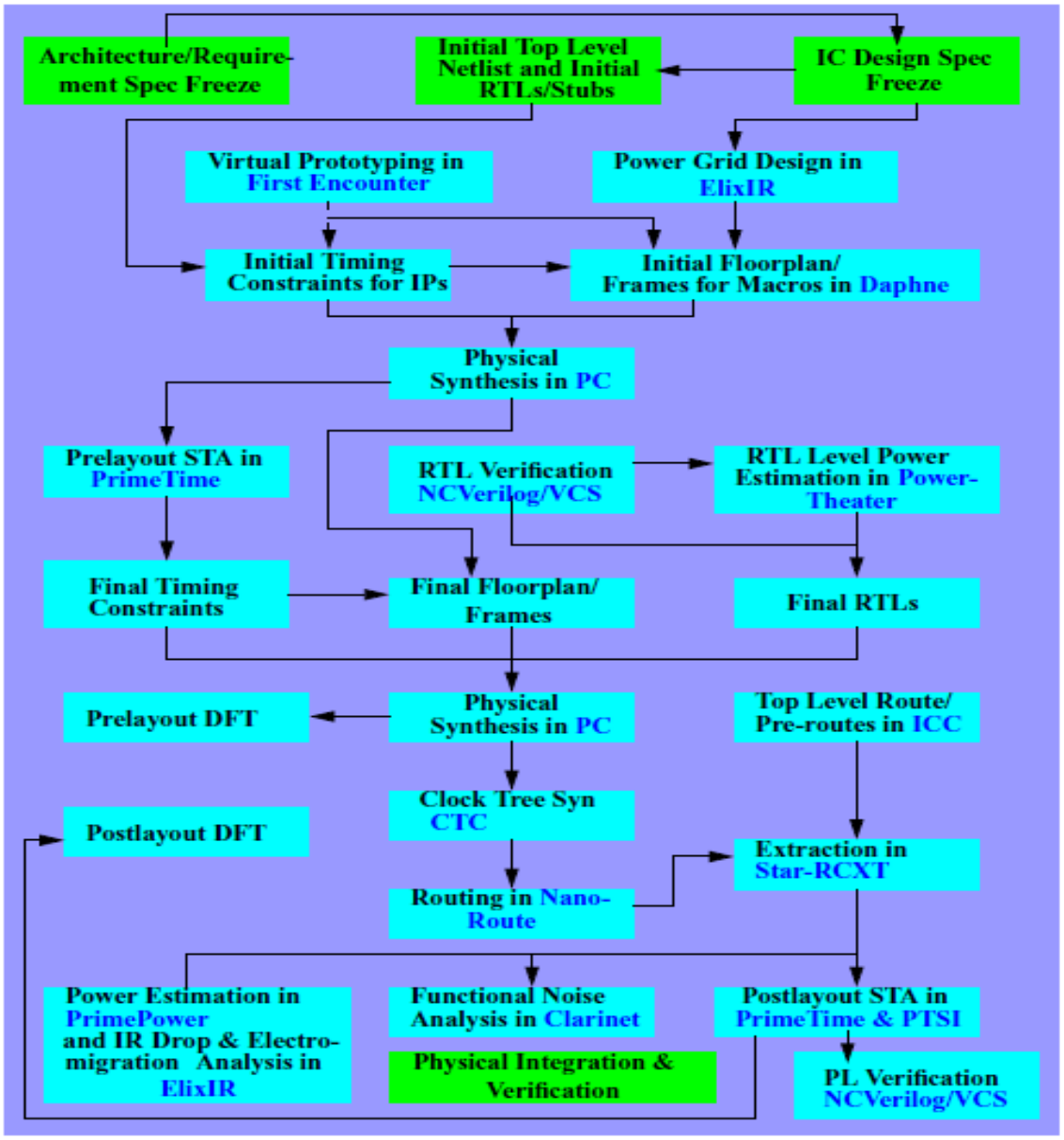


Figure 1.1: SoC Design Flow

Chapter 2

Test Chip Architecture

2.1 Test Chip IP's

This CMOS SoC has integrated multiple instances of Digital Design Kit standard cells, dual port register files, RAMs and ROMs, Standard cells , Ring Oscillators and 2PRFs have been tested in two orientations 0 degree and 90 degree. Following is the list of IP's being integrated in this SOC.

1) 21 instances if 2PRF

2) 12 instances of ROM

3) 6 instances of ST-RAM

4) Standard cells including core, clk, srpg, multibit srpg, sequential cells

5) Ring oscillators

6) Clock and reset control

7) 24 instances of frequency counters to measure frequency of different clocks, tap delay chains and Ring Oscillators.

There are 5 instances of FSL BIST engine implemented for testing these memories. All BIST engines memory testing and access time characterization logic has been encapsulated in MTR The registers used for memory characterization and DMA control are implemented in MTR. The registers for controlling clock, reset

and frequency counters are implemented in core logic.

2.2 Test Chip's Architectural Overview

This cmos testchip architecture is based on the skyblue bus protocol structure, which is the primary functional mechanisms for accessing the devices on the chip. In this architecture following buses are used:

1) Skyblue bus Standard skebpue IPS bus for accessing the SOC devices

2) System Control Bus for reset and system clocking

3) Memory Control Bus for providing the control signals for memories in DMA mode

4) Test bus for checking scan capabilities of individual memories

This chip hosts different two port register files, ROMs, ST provided RAMs and e-fuse devices. In this CMOS SOCs the functional access to all the devices under test is provided through skyblue bus. The bus has 32 bit addresses and different schemes of addressing the RAMs, ROMS and dual port register files have been adopted. There is a separate read and write data bus in the Skyblue protocol. The chip is enabled using a module enable pin and read and write signal is controlled by another pin on the IPS bus. Chip id module is inside MTR and can be referred in MTR section.

This soc also tests standard cells library and Ring Oscillator structures. All the memories are encapsulated in a memory test and repair framework ⟨MTR⟩ which has 5 BIST engines to test different types of memories and a repair processor connected to e-fuse. This architecture is open and flexible because it allows to add or remove any CUTs from the core or MTR as well as to add or remove a whole core without any changes in bus structure, access and addressing methods. The skyblue bus is byte aligned and the chip is aligned on 32-bit word boundary.

The memories are tested using FSL BIST and also are accessible from the direct

memory access enabled through skyblue bus The memories are tested for functional correctness using BIST and characterized through a tap delay chain mechanism. All memory accesses are enabled either through BIST engine or through direct memory access DMA. The DMA mode also provides a provision for a bypass mode of memory access where the pipeline flops are bypassed. The chip can drive the ipt bus from top and debug the memories in scan mode.

Other important blocks to enable the testing of the above are frequency counters, clock and reset generators.

2.3 Standard Cell Core

2.3.1 All Cell Block Description

There will be a total of 8 blocks for all cells namely SEQ, CLK, CORE, MULTI-BIT, MULTIBITSRPG, SRPG, DELAY, METSTABILITY. These blocks are based on library grouping of cells.

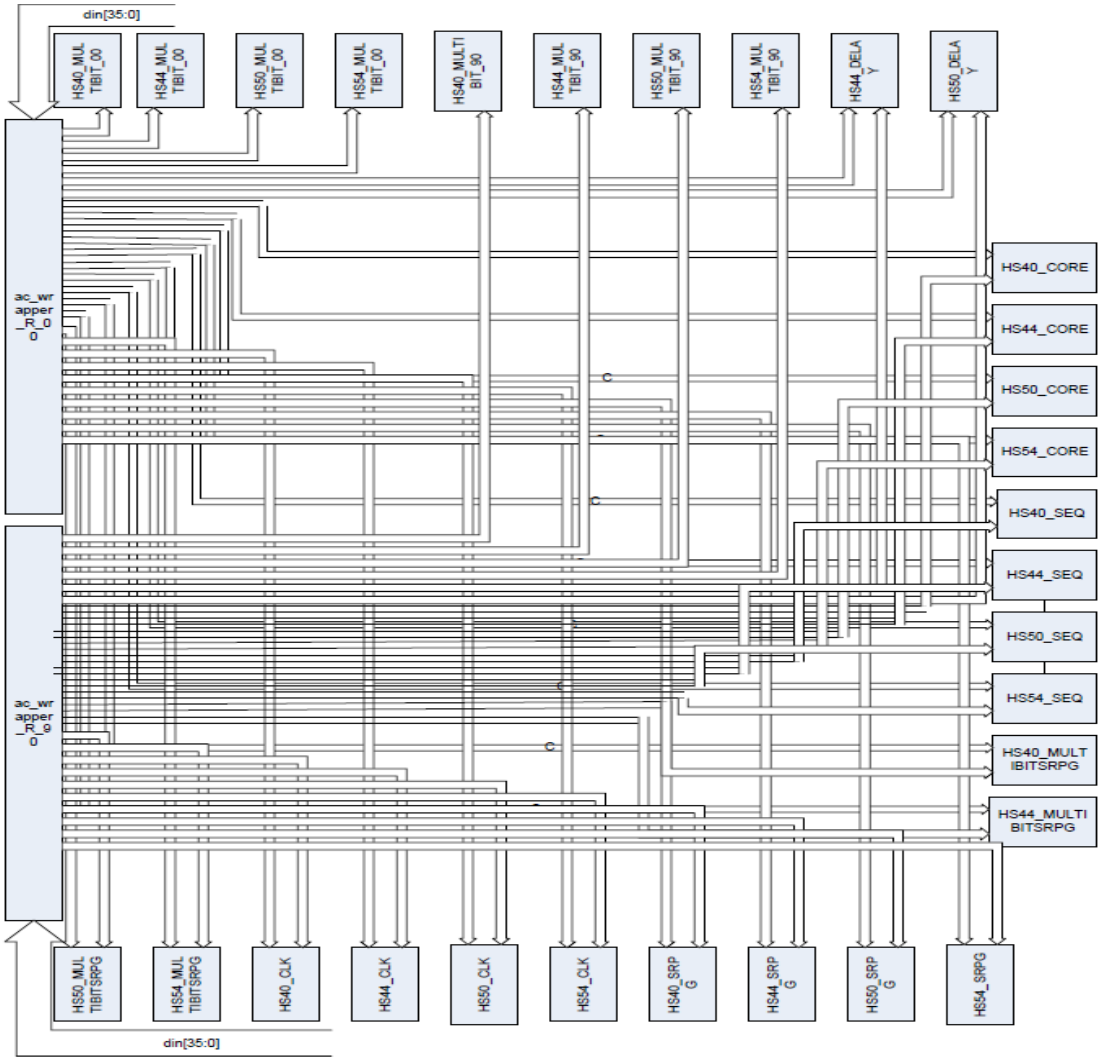


Figure 2.1: All Cell Core

For any given cell, the output of the cell across all pvts, channel lengths , rotations will be available at dout at the same time.

The cut select will act as gating element for the inputs. i.e if the cut is selected the block will get the inputs as it is otherwise it will be tied to zero.

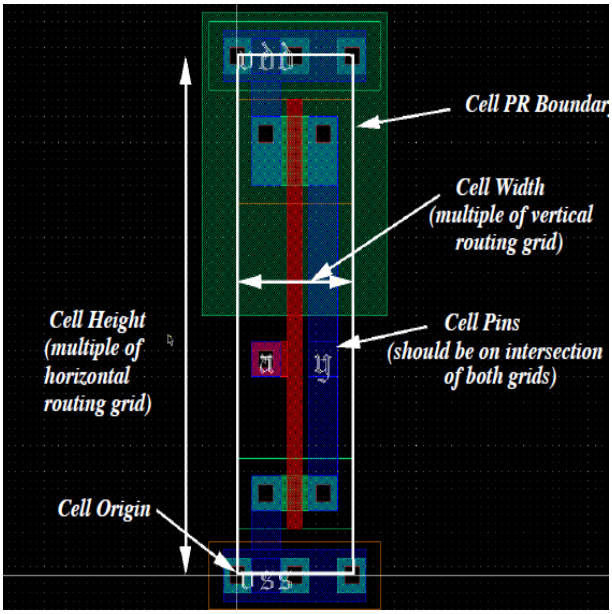


Figure 2.2: Standard cell Layout - Inverter

Cell libraries determine the overall performance of the synthesized logic.
Synthesis engines rely on a number of factors for optimization. The cell library should be designed catered solely towards the synthesis approach. Here are some guidelines:

- A variety of drive strengths for all cells especially for inverters and buffers.
- Cells with balanced rise and fall delays (for clock tree buffers/gated clocks).
- Same logical function and its inversion as separate outputs, within same cell.
- Complex cells (e.g. AOI, OAI)and High fanin cells.
- Using high fan-in reduce the overall cell area, but may cause routing congestion inadvertently causing timing degradation. Therefore they should be used with caution.

- The cell height/width must be a multiple of the horizontal/vertical grid spacing. All cells must have the same height, but some complex cells can be designed with double height.

2.4 Clock Generation and Control Register

The CMOS Architecture requires one external clock that is a clock generated off-chip and used as system clock. This clock is named ipg_clock and is an ungated continuous clock. The external clock frequency has the limitation of pads i.e. approx 25 MHz. All internal clocks are generated and controlled in the module clock_gen. Due to the requirements of running BIST at speed, bist clocks are generated separately and have an internal oscillator muxing option to route ipg_clk or internal oscillator clock to the BIST engines.

2.4.1 Clock Control Register

The memories are not gated in this SOC as the memories stop their clocks if the enable to memories is stopped. The two clock control registers are for controlling bist clock generation and selection.
The clock gate is enabled when 1b0 is programmed in relevant clock control register. Thus at reset all clocks are free and will be gated only by programming.

Address	**Register Name**	**width**	**Type**	**Reset Value**
13'h00	bist_clk_ctrl	[31:0]	R/W	25'h000_0000
13'h04	all_cell_clk_ctrl	[31:0]	R/W	21'h1F_FFFF

Table 2.1: Clock Register

2.5 Reset Generation

A reset generation module generates the asynchronous hard reset with synchronous de-assertion. This block also provides an early reset for REP processor and a seperate reset for different qreset pins of the memories. The reset signal is active low, asserts asynchrounously and deasserts synchronously to system clock ipg_clk. The SOC system reset is generated when ipg_hard_asynch_reset_b signal is asserted and the uppermost address of address map that is 0xFFFF_FFFF is asserted. To generate the qreset signal for each individual memory devices, the memory device needs to be addressed as per the address memory map and the ipg_hard_asynch_reset_b signal needs to be asserted.

The system resets the entire soc i.e. it resets every block of the chip.

To generate system reset follow the steps below:

1. Drive ips_addr 0xFFFF_FFFF on address bus - ips_adddr pins
2. Drive ipg_hard_asynch_reset_b signal to low. For generating a CUT specific

reset_cut	**Module**
0	All
1	Early Reset for REP
2	Frequency Counter block reset
3	MTR Register block reset
7-45	Individual memory device output reset

Table 2.2: Reset Table

reset an address in the devices system address map is driven on address bus i.e. ips_Addr and then asynchronous reset signal is asserted.

It is required to clear frequency counter registers to be cleared with software control. This LMTV would not have separate resets for register blocks. The reset generation is fully controllable by a combination of external asynchronous reset signal ipg_hard_asynch_reset_b and ips_address. The repair processor needs to be driven by early reset as it has to ensure that the power on reset does not write the data

accidentally on the effuse while repair processor is still busy.

2.6 Ring Oscillator

This test-chip is proposing to have 891 ROs in different cells of the DDK. These ROs will have two orientations of 0 and 90 degrees. The OCV ROs are repeated , i.e. have two instances each close to each other. The ROs will be using L,H and S Vt cells. At one point there are maximum 24 ROs that can be active and read in parallel. The expected number of rows is 48 and maximum 24 columns.
Ips_addr[6:2] are used for column addressing and ips_addr[12:7] for address decoding. Ips_addr[15] will select all the columns.
There are two orientations:
1. First block is having 60 unique RO in R90 orientation.Since these are OCV cells hence these ROs are repeated so there are 120 ROs in 90 degree rotation block.
2. Second block is having 773 ROs and in R0 orientation and 120 RO instances of the same cells that are in 90 degree rotation block.
The ro block will have 3 inputs and a 24 bit wide output bus.
The inputs are :
Ips_addr[15:0] To select the ros.
Block_en To enable the ring oscillators block. (gets enabled when ips address is in RO address memory map range.
Osc_En to enable the oscillations in the ro's 1 is written on the RO address. (Connected to ips_wdata[00] pad). [Active high]
To enable parallel testing a maximum of 24 ring oscillators output can be observed parallel based on the addr.
The RO block is selected whenever there is an ips access to this block of RO addresses. To enable the oscillations, select the ring oscillators block by programming the ips address correctly as given in the RO names and address maps. Now pro-

gram the address based on the rings that needs to be tested. The oscillations are enabled whenever the data 0x0000_0001 is written on the selected address. Enable the oscillations by asserting the en pin. The output of the oscillators pass through a 32 bit divider (four stages of dividers) before it reaches the frequency counters i.e. if the period of oscillations is say 20 ns from the rings a pulse of period 320 ns will reach the frequency counter inputs. The period can then be calculated by using the frequency counters.
The addr pins are used as follows:

Address Pin	**Function**
addr[15]=1'b1	Parallel Output
addr[12:7]	Decoder to rows
addr[6:2]	To select individual RO output column

Table 2.3: RO Selection Address Range

Here is an example of testing the RO and reading the frequency from the frequency counters.
To count oscillations on

1. Set calibr_data_sel register for frequency counter input select write 4b1001 in fq_calibr_data input register.
2. Select the RO by addressing the RO from the table.
3. Enable oscillations in the RO block by writing 0x0000_0001.
4. Assert start_counting.
5. Wait for desired time.
6. Deassert start_counting.
7. Read the frequency counter by providing the correct address.

2.7 Frequency Counter

This CMOS test-chip has 24 addressable (for read only) 32 bit counters.

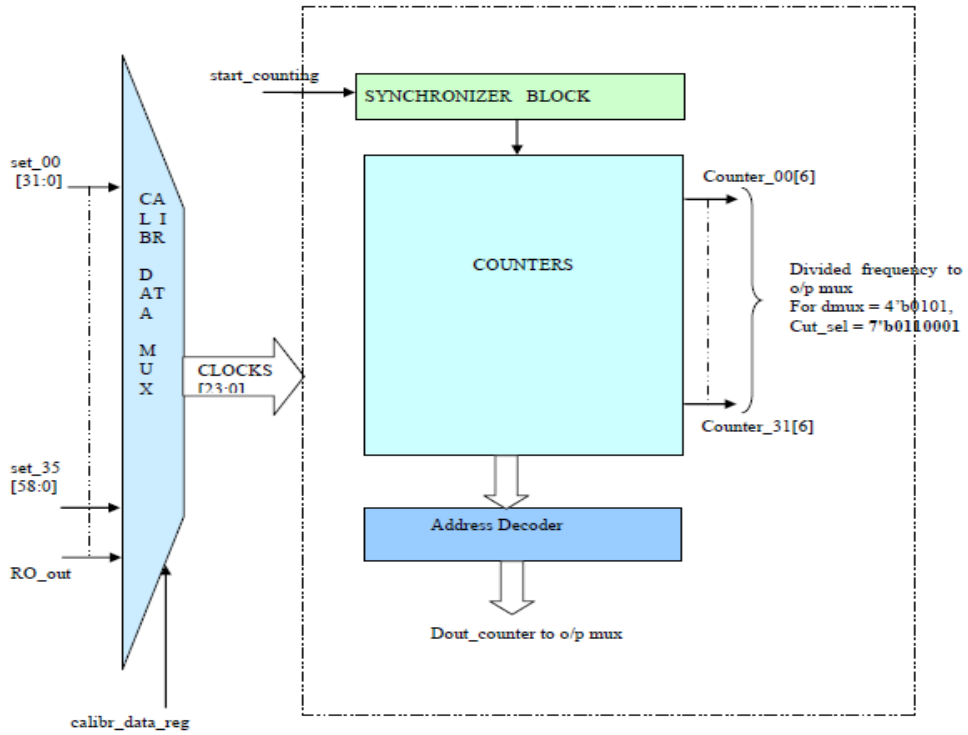


Figure 2.3: Frequency Counter Block Diagram

As shown in the block diagram Oscillating delay chains get multiplexed in calibr_data_mux. These are further supplied as clocks each supplying to one counter. The start_counting input is synchronized to local clock for each counter. The counters can be read after start_counting is deasserted at dout when cut_sel = 7'b0110001. Also the seventh bit of every counter can be read as a frequency divided output when dmux is set to 4b0101 at cut_sel = 7'b0110001 value. Note that dmux value should be other than 4b0101 for normal read operation of o/p multiplexer. The counters can be read as registers provided start_counting is deasserted atleast 2 clock cycles prior. They are mapped from 5h00 to 5h1f on addr bits[4:0]. Delay line n corresponds to counter n.

2.8 Memory IP

Semiconductor Memories are classified according to the type of datastorage and the type of data access mechanism into the following two main groups:

- Non-volatile Memory (NVM) also known as Read-Only Memory(ROM) which retains information when the power supply voltage is off. With respect to the data storage mechanism NVM are divided into the following groups:
  - Mask programmed ROM. The required contents of the memory is programmed during fabrication,
  - Erasable PROM (EPROM). Data is stored as a charge on an isolated gate capacitor (floating gate). Data is removed by exposing the PROM to the ultraviolet light.
  - Electrically Erasable PROM (EEPROM) also known as Flash Memory. It is also base on the concept of the floating gate. The contents can be re-programmed by applying a suitable voltages to the EEPROM pins. The Flash Memories are very important data storage devices for mobile applications.
- Read/Write (R/W) memory, also known as Random Access Memory (RAM). From the point of view of the data storage mechanism RAM are divided into two main groups:
  - SRAM (Static Random Access Memory)
  - DRAM (Dynamic Random Access Memory)

2.9 Static RAM

The memory cell is a 6 transistor circuit which is a flip flop comprising two cross-coupled inverters and two access transistors, the access transistors turn on when the

word line is selected (high) and its voltage rises to Vdd, and they connect the flip flop to the bit lines. Sizing of the transistors in the memory cells is very important especially for speed and chip cost.

The sense amplifier is important in the total performance of the SRAM chip since the sense delay time directly affects the access time. Sense amplifier is used to sense the small changes in voltage that results when a particular cell is switched onto the bit line. One stage differential pair of sense amplifier is utilized here. The sense amplifier circuit is controlled by a clock signal, which is synchronized with the pre-charging and word-line signals.

- TYPE OF SRAM:-
  - Single Port static Random Access Memory (SPRAM)
  - Dual Port static Random Access Memory (DPRAM)

2.9.1 Single Port Random Access Memory (SPRAM)

SPRAM is a single-port synchronous static RAM contains single port to used for read and write operation.
When clock goes to negative to positive edge, CSN and WEN is low, Memory is written whatever data is given to data port at current address given at address bus. When WEN is HIGH, CSN is low and CLK goes to POSEDGE, Memory is read at current address given by address bus and data reflects on the Output Bus.

- Memory has logic low control pin of CSN and WEN.
- WEN - Write Enable logic low Write when Logic low otherwise read.
- CSN - Chip select logic low Chip selected Write when Logic low otherwise disable.

- Memory has SLEEP which is used to switch off peripheral. Memory also has test mode,scan chain pins,special testing pins.

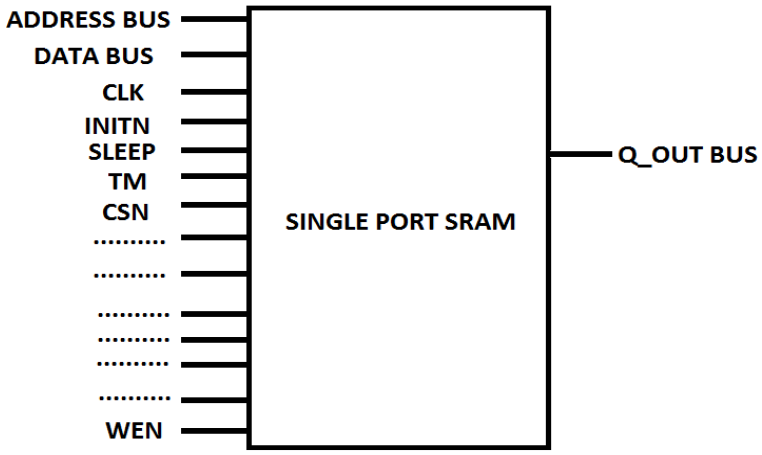


Figure 2.4: SINGLE PORT SRAM

2.9.2 DUAL Port Static Random Access Memory (DPRAM)

DUAL Port SRAM gives advantage of read and write perform simultaneously by adding extra data bus, address bus and extra control signal.In some application like Video RAM , Elevators to Robot Control, Commercial Aircrafts to Unmanned Flight Controls, Surveillance cameras to Night vision systems.Dual Port Memory has increased bandwidth approximately 2x the speed of a similar single port RAM.

2.9.2.1 DUAL Port Static Random Access Memory TYPE-1

This type of dual port memory has only one data port though it contains read and write different address bus as well as different chip selection bit. It has one write enable signal common for both.

When Memory is going to read, Address is captured from read address bus and when Memory is going to write, address is captured from write address bus. Both

read and write also have different chip selection pins.This dual port memory has limitation that it can read or write at a time.

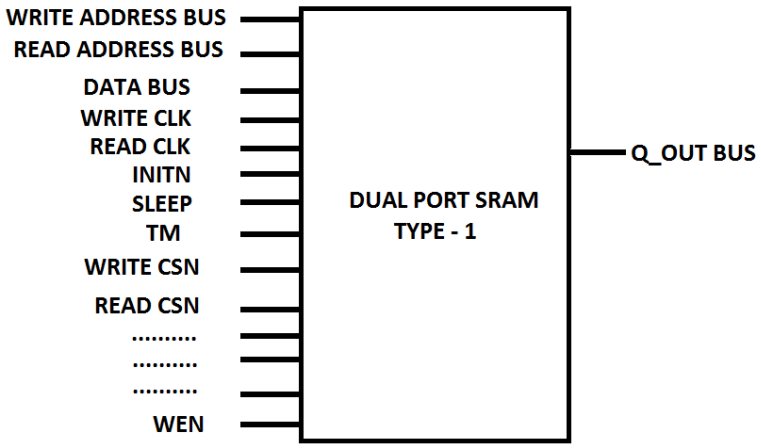


Figure 2.5: DUAL PORT RAM TYPE - 1

2.9.2.2 DUAL Port Static Random Access Memory TYPE-2

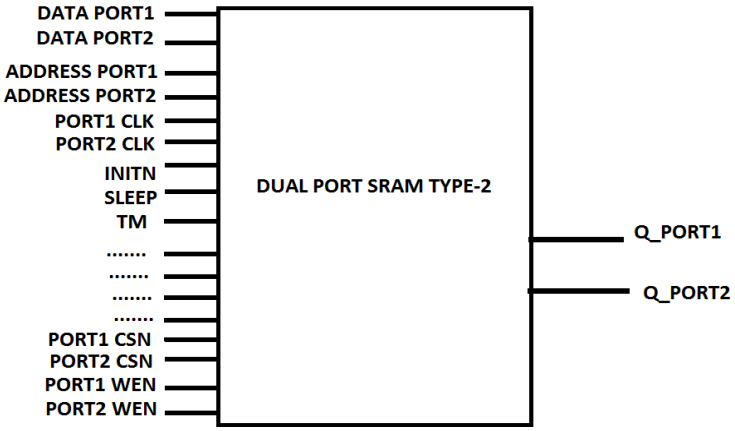


Figure 2.6: DUAL PORT RAM TYPE - 2

This type of dual port memory has two data port though it contains read and write different address bus as well as different chip selection bit. It also contains different write enable signal which gives permission to read and write at a time.

Memory has 2 data port , 2 CSN, 2 WEN , 2 Address Bus as well as 2 Output Data port. It is same like two SPRAM (Single Port SRAM) are going to attached together.

DUAL PORT SRAM port is independent of each other. Both port has their own WEN and CSN signals as well as their operating frequency.

Chapter 3

Front End Integration

3.1 Introduction

Front end integration is the process of integrating the design database, based on the L3 specification of the chip. The process involves generation of top-level chip netlist and providing a testbench in which the module owners can start verification of their modules. Front-end integration is not completed until and unless clearance is given by the verification team on the connectivity of all the modules.

To verify a module, for its functionality, at different stages in the design, a testbench is required. At the stand alone stage, only the module under test (MUT) is instantiated inside the testbench, and requires the Bus Functional Models (BFMs) of the cores.

A full functional testbench consists of the following:

- Micro Operating System (MOS) - Responsible for booting and initialization of the cores.
- Embedded codes - to program the cores in the required state and to check the desired functionality.
- Verilog stimulus - to provide external stimulus (if required)

- Drivers/Monitors/Transactors - required for different transactions e.g driving a pin, monitoring an output or interacting with the external world etc.
- Boot Loaders:- In order to load the code inside the memories, boot loaders/memory loaders are required.

3.2 Deliverables to Front-end Integration

The following bullets are the deliverables to the FE team:-

- Integration guide of the modules.
- Preliminary and final stubs of the modules and platforms.
- Preliminary and final RTLs of the modules.
- All the behavioral models of the analog portions of the design.

3.3 Deliverables from Front-end Integration

The following bullets are the deliverables from the FE team:-

- Verilog compatible and correctly connected Toplevel netlist of the chip.
- A Testbench in which RTL and post-layout verification of the modules can be done.

3.4 Flow

The top-level netlist generation involves the following stages:

3.4.1 Phase 0

Stage1:-

The module stub file is delivered by the module design owners.The initial platform netlist is generated using the module stub files based upon L3 specifications of the chip.

Stage2:-

The final stubs and preliminary RTLs are delivered by the module designers. Using the platform level netlists and the module level stubs of the hard blocks, an initial toplevel chip netlist is generated. Hard blocks are the modules whose GDSII is delivered by the designers and they are not included in the PC and APR flow of the platforms.

The netlists are usually integrated either manually or by using tools like Flexior and Rabbit.

3.4.2 Phase 1

Stage1:-

Final module RTLs are delivered by the module design owners.The module RTL port list is compared with the module stub portlist. Final stubs should match the port-list in the RTLs. If there is no issue in the port-lists, the module RTL is used for generating the platform netlist.

Stage2:-

The platform netlists, generated by the platform owner, is used for integrating the top-level netlist. A final toplevel netlist is generated by integrating all the

platform netlists and the hard macros which are outside the platforms.

Stage3:-

Platform owner delivers the postlayout platform netlist. Hard module design owners for the modules which are outside any platform, deliver the blocks postlayout netlist.Postlayout platform netlist and the postlayout block netlist is used for integrating the top-level netlist.

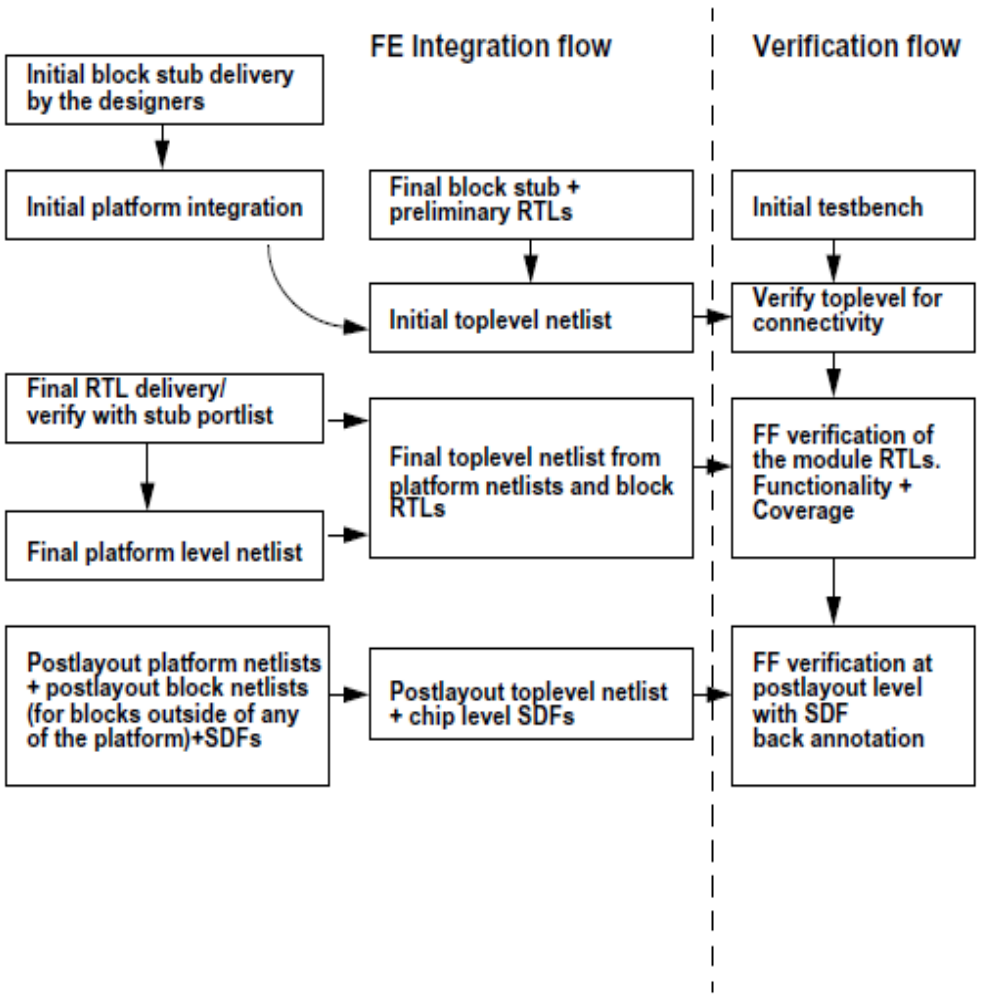


Figure 3.1: Front End integration/Verification Flow

3.5 Functional Verification

3.5.1 Introduction

Verification is a process of verifying the design for any incompatibility with the desired specifications. The process includes, checking for the basic functionality, checking for the inter-module interaction, checking for the interaction with the cores and the memories, checking for the interaction between the module and the external chip-sets etc. Verification of a chip requires, the design database (RTLs, platforms, top-level netlist, SDFs etc.), stimulus files (written in verilog, C, C++, Vera, SystemC etc.) and a verification environment.

3.5.1.1 Deliverables to Verification Team

The following bullets are the deliverables to the verification team:-

- A toplevel netlist and testbench from the front end integration team.
- Chip verification methodology.
- SDFs and post-layout netlists from STA and APR teams.

3.5.1.2 Deliverables from Verification Team

The following bullets are the deliverables from the verification team:-

- Sign off on the RTLs and post-layout netlists of the modules.
- Vectors of the modules for regressions.

3.5.2 The Flow

Verification process basically involves following 3 stages:

3.5.2.1 Phase 0:

Stand-alone verification of the modules- Stand-alone verification is done for checking the basic functionality of the module (e.g register rd/wr tests). Almost all the modules, interact with one or more cores. The Bus Functional Models (BFM) are used for the testcases involving interaction with the core. If the module interacts with the memories, the behavioral models of the memories are used.

During standalone verification, there are several corner cases which are not thought of by the designer.In order to attain maximum functional coverage, random verification is used.Vera is the tool of choice for doing random verification.Functional coverage is the term which is used to determine if all the states in the design are exercised in proper sequence.

3.5.2.2 Phase 1:

Full Functional verification of the modules - Almost all the functionality of the modules are verified in the full functional verification. Full functional verification requires a Micro Operating System (MOS), which is capable of feeding inputs (stimulus) to the design and reading the outputs and verifying it against the specifications at RTL level and later on at post-layout level both for the timing and functionality.

3.5.2.3 Phase 2:

System level verification - The functionality which is not covered in the FF verification is covered in the system level verification. This includes, intermodule interaction, data path checks, external module interaction, pad connectivity checks etc.

Verification process is not complete until the functional coverage has been checked.

Chapter 4

Synthesis

4.1 Introduction

Logic Synthesis is the process by which an abstract form of desired circuit behaviour, typically Register Transfer Level(RTL), is turned into design implementation in terms of logic gates.

4.2 RC Synthesis Flow

4.2.1 Prerequisites for Synthesis with RC

(i) Liberty files(.lib)

(ii) Library Exchange Format(.lef)

(iii) Verilog and/or VHDL and/or System Verilog File(s)

RC accepts verilog/vhdl and even mixed style of RTL. The RTL files can contain structural code for combining lower level modules, behavioral design specifications, or RTL implementations.

Synopsys Design Constraints (.sdc) File(s)

RC supports reading SDC format for constraints. Make sure that all clock definitions, proper i/o delays ,correct uncertainty and derates included in the sdc

file.

(v) Dont Use Cell List.

(vi) Dont touch cells/instances/modules: There might be certain cells/instances/modules in the design which have to be kept as dont touch during synthesis. So, the dont touch list is needed.

In order to preserve these cells/instances, use the following command in RC:

set_attr preserve true [find / -instance ⟨ instance name ⟩]

If it is desired to preserve a particular cell with upsizing and downsizing, use:

set_attr preserve size_ok [find / -instance ⟨ instance name ⟩]

To allow the tool to resize or delete a mapped subdesign or child instance during optimization, but not rename or remap it, use :

set_attr preserve size_delete_ok [find / -instance ⟨ instance name ⟩]

(vii) Optional but Recommended Files:

- Capacitance Table File (.captbl)
- Design Exchange Format (.def) Floorplan File
- Switching Activity File(s): .saif, .tcf, .vcd
- Common Power Format (.cpf) File

4.2.2 What Tool does during Synthesis

RC Synthesis involves the following stages:

(i) Elaboration

(ii) Generic Synthesis

(iii) Global Mapping

(iv) Incremental Optimization

(i) Elaboration:

Before Elaboration the design should be read in along with other inputs. Make sure that general compiler settings are done.

Elaboration is required on the top level design which automatically elaborates all its references. The *elaborate* command does following things:

- Builds Data structures and infers registers in the design.
- Performs High level HDL optimization, such as dead code removal.
- Identifies Clock Gating and operand isolation candidates.
- During elaboration, the tool reports the following :
- Unresolved references, that is the instances found with no corresponding module or library cell.
- Semantic problems, including unused ports, inconsistent resets.etc
- *check_design* command can be used to report all these things in detail.

(ii) Generic Synthesis:

- A technology independent synthesis is done during this stage. Following are some of the optimizations done in this stage:
- Carry save arithmetic optimization
- Logic Pruning
- Resource Sharing
- Implementation Selection
- Redundancy Removal
- Mux Optimizations

- Common Sub-expression sharing

(iii) Global Map:

- The global mapping process in RC involves:
  - Structuring
  - Mapping
  - Target setting
  - Restructuring/Mapping as per target settings

(iv) Incremental Optiization :

- Incremental Mapping Performs:
  - Critical Range Re-synthesis
  - DRC fixing
  - Buffering
  - Area Recovery

4.2.3 Area-based Optimization

(i) Analyze the timing reports prior to synthesis, i.e., at pre-synthesis diagnosis stage. If the I2C, I2O paths are already meeting timing, then area can be recovered from these paths. For this, under constrain these path groups before the global mapping stage:
Ex: The following command will relax the timing on I2O paths by 200 ps. So, now the effective clock-period for these paths are [Clock_period + 200].
path_adjust name UNDER_CON delay 200 from [all::all_inps] to [all::all_outs]
After global mapping phase, remove the path-adjust, so that timing reports would be accurate and with respect to the actual clock period.
(ii) Synthesize -to_generic effort medium

As per LEC requirement,it is preferred to use medium effort for synthesize to_generic. In cases where timing is critical ,the effort level can be set to high if and only if LEC is not an issue.Here ,RC uses more aggressive csa optimization algorithms for better timing. This might result in larger area as well.

(iii) Set the attribute drc_first to false, in case if you have set it to true (default is false). This has to be set to false, so that RC does not fix DRCs aggressively & bump up the area.

(iv) Review and remove any preserve attributes (dont_touch) that are not needed. This will help RC to optimize the results better.

(v) Check if there are any datapath elements in the critical paths with preserve attributes.

(vi) Set the attribute dp_postmap_downsize to true on data path elements present in the design before incremental synthesis. This will perform architecture downsizing after mapping. This attribute is effective only in incremental optimization.

(vii) If timing is not critical in the design, you do not need to turn on the attribute tns_opto.

(viii) Area multiplier: The area of a cell can be modified by using the attribute area_multiplier. The default value for this attribute is 1.0. When set to a value less than default value can favor a cell to be picked up by the tool. For example, when applied on complex cells, it can bias the tool for mapping them on non-critical paths. To do so, one could set the area_multiplier value to less than one 1.0 for these kind of cells. However, for accurate area reporting, you should change the multiplier to 1.0 before issue any cell or area reporting command.

(ix) Check if proper complex libcells are selected when needed. Otherwise one can bias the tool to pick them up using area multiplier trick discussed above.

(x) If you have a max_delay constraint set on your design, RTL Compiler interprets the constraint more restrictively and may produce larger area and instance count. To improve area, check for these and convert them in to set_input_delay set_output_delay constraints that refererence to the appropriate clock.

4.2.4 Timing-based Optimization

Following are the important steps in addition to those mentioned in the flow, which will improve the performance:

(i) Do a detail analysis of timing reports in RC just prior to synthesis, i.e., just after loading the RTL into RC (Pre-synthesis Diagnosis). If there is some MACRO, which is in the critical path of reg-to-reg path groups and which has huge delay, then make a separate path group for all the paths passing through that MACRO. This will greatly improve the optimizations of all the other reg-to-reg paths not passing through that MACRO.This MACRO path can also be optimized in a better way by over constraining it.

(ii) At the beginning of global mapping, based on the libraries, logic structure and constraints RC will estimate a target slack for all cost groups. RC tries to meet these numbers. In the logfile look for the word target slack. If there is a cost group with large negative slack normally is a problem area. If the constraints are clean, one could set the initial target to 0 or some positive value using the initial_target attribute. This forces the mapper to do aggressive structuring and optimization on them to meet the target set.

set_attribute initial_target 0 [find / -cost_group name_cg]

Note : This may increase area.

(iii) Over constrain the paths, which are not meeting timing.

Ex: The following command will over constrain the reg-to-reg paths by 200 ps. So, the effective clock-period for these paths is now [Clock_period 200 ps]. Hence, the tool will try to meet its target in this effective clock-period.

path_adjust name OVER_CON delay -200 from [all::all_seqs] to [all::all_seqs]

This command is to be used before global mapping stage. After global mapping, remove the over constraint and do report timing, so that the timing reports are with

respect to the actual clock-period.

(iv) If timing degradation is due to high fanout nets in the design, idealizing those nets improves timing at the synthesis stage. In the later stages a place and route tool can build a better buffer tree for these nets. The script mentioned in the Tcl procedures section can be used to find out the high fanout nets and idealizing them.

(v) Usage of Low Vth cells significantly improves timing. But usage of these cells significantly increases the power numbers as these have high leakage power.

(vi) Set the attribute tns_opto to true. When set, it forces the tool to consider all the endpoints for the optimization.

Note: This may increase the area.

(vii) Make sure that the attribute drc_first is not set to true. By default it is set to false. If set to true the tool will give higher priority to design rule constraints than the timing constraints.

(viii) Set the attribute iopt_ultra_optimization to true .When set, it enables ultra optimization in incremental optimization to achieve best QOR with higher runtime.

4.2.5 Special Care abouts and Recommendations

1. Boundary Optimization

By default, RTL Compiler performs boundary optimization during synthesis for all subdesigns in the design. It controls boundary optimization on the subdesign and hierarchical pin inversion. To preserve the input and output pins of a subdesign, you can turn off the boundary optimization. In this case, no hierarchical pin inversion will be done either for this subdesign.

Note: To exclude individual pins from boundary optimization, use the preserve attribute

The boundary optimization can be turned off as below :

set_attr boundary_opto false [find / -subdesign ⟨ subdesign_name ⟩]

2. Retiming

Improves the performance of the design by either optimizing the area or the clock period (timing) of the design. Retiming moves the registers across the combinational logic to improve the performance without changing the input/output behavior of the circuit .Generally it is not recommended retiming on the whole design due to multiple reasons :

- You will lose traceability of the flops since all the retimed flops will become retime*reg. Verification could be a major issue for both formal and simulation. The retiming space explodes to the complete design and flops can get spread out e from one module to one or more others.
- There will be runtime increase since the Retiming flow would require an initial mapping during prepare phase, retiming step and a final mapping on the whole design.
- It is recommended to use Retiming only on selected subdesigns. The ideal subdesign would those which have been designed for pipelining or if not choose that would benefit from register spreading based on unbalanced paths paths which have positive slack on one side and negative slack on the other side. So moving the registers in the direction of positive slack would reduce the negative slack.
- On a Post-mapped design, you could try incremental retiming which makes use of the unbalanced paths. But on typical designs, if the negative slack cannot be improved by incremental retiming, then success would be limited.

3. Ungrouping

RC by default ungroups the user created hierarchies during synthesis to improve area and timing during synthesis. This can be controlled by setting the attribute

auto_ungroup to both/none and must be specified before synthesis.
none Ungrouping will not be performed.
Both Ungrouping will be performed with an emphasis on both optimizing timing and area.
You can also set the ungroup_ok attribute to false to control any subdesign/hierarchy not to ungrouped. This way, while achieving better QoR, you can also keep the hierarchy of interest intact.
Note: Ungrouping of user hierarchies happens during

- RTL optimization synthesize -to_gen -effort high
- Technology mapping synthesize -to_map -effort high

4. Information to be looked in log files

(i) For unresolved instances during elaboration. The command for checking unresolved instances is check_design unresolved
(ii) check if any sdc errors exist and verify the SDC summary report in the log file.
(iii) Warnings pertaining to sequential logic deletion.
(iv) Clock gating fanout statistics.
(v) At the beginning of the global mapping step, RC will estimate a target slack for each cost group. This estimated target is based on the libraries, the logic structure, and the constraints. RC will work toward this target number during the optimization process. In the logfile, search for the keyword target slack, as it will be printed before and after the global mapping step for each cost group in the design. A cost group with large negative target slack would normally indicate a problem area. Also check whether these targets are met after mapping or not.
(vi) On some designs RC might spend time in incremental optimization. To debug such cases one should have set the debug variable iopt_stats to 1 and the attribute information_level to 9.
(vii) Before attempting to run synthesis, the user should check the input data, pay

attention to the warning messages and correct any obvious issues.

Chapter 5

Configuring SDC

5.1 Specifying Clocks

To define a clock, we need to provide the following information:

i) Clock source: it can be a port of the design, or be a pin of a cell inside the design (typically that is part of a clock generation logic).

ii) Period: the time period of the clock.

iii) Duty cycle: the high duration (positive phase) and the low duration(negative phase).

iv) Edge times: the times for the rising edge and the falling edge.

By defining the clocks, all the internal timing paths (all flip-flop to flip-flop paths) are constrained; this implies that all internal paths can be analyzed with just the clock specifications. The clock specification specifies that a flip-flop to flip-flop path must take one cycle.

Here is a basic clock specification1.

create_clock -name SYSCLK -period 20 -waveform 0 5 [get_ports2 SCLK]

The name of the clock is SYSCLK and is defined at the port SCLK. The period of SYSCLK is specified as 20 units - the default time unit is nanoseconds if none has been specified. (In general, the time unit is specified as part of the technology

library.) The first argument in the waveform specifies the time at which rising edge occurs and the second argument specifies the time at which the falling edge occurs.

There can be any number of edges specified in a waveform option. However all the edges must be within one period. The edge times alternate starting from the first rising edge after time zero, then a falling edge, then a rising edge, and so on. This implies that all time values in the edge list must be monotonically increasing.

-waveform time_rise time_fall time_rise time_fall ...

In addition, there must be an even number of edges specified. The waveform option specifies the waveform within one clock period, which then repeats itself.

5.2 Clock Uncertainty

The timing uncertainty of a clock period can be specified using the set_clock_uncertainty specification. The uncertainty can be used to model various factors that can reduce the effective clock period. These factors can be the clock jitter and any other pessimism that one may want to include for timing analysis.

set_clock_uncertainty -setup 0.2 [get_clocks CLK_CONFIG]

set_clock_uncertainty -hold 0.05 [get_clocks CLK_CONFIG]

Note that the clock uncertainty for setup effectively reduces the available clock period by the specified amount.

5.3 Clock Latency

Latency of a clock can be specified using the set_clock_latency command.

There are two types of clock latencies: network latency and source latency. Network latency is the delay from the clock definition point (create_clock) to the clock pin of a flip-flop. Source latency, also called insertion delay, is the delay from the clock source to the clock definition point. Source latency could represent either

on-chip or off-chip latency. Figure shows both the scenarios. The total clock latency at the clock pin of a flip-flop is the sum of the source and network latencies.

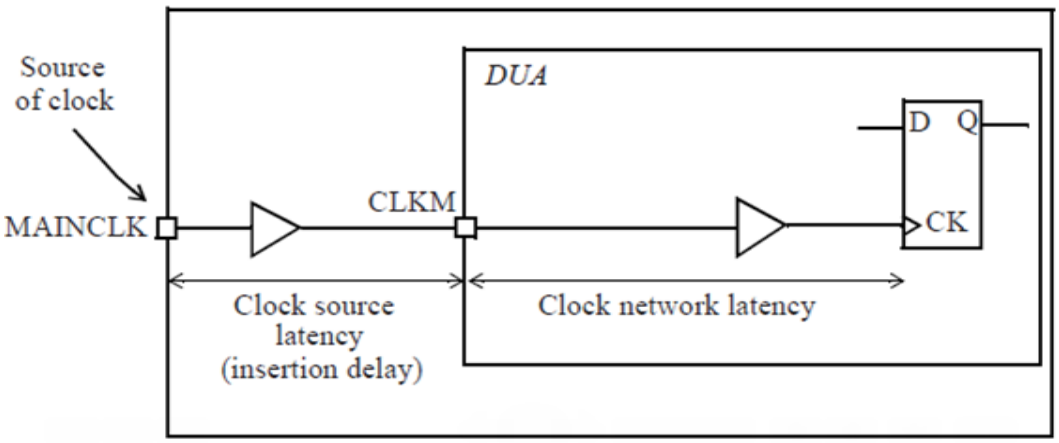


Figure 5.1: Clock Latency

One important distinction to observe between source and network latency is that once a clock tree is built for a design, the network latency can be ignored (assuming set_propagated_clock command is specified). However, the source latency remains even after the clock tree is built. The network latency is an estimate of the delay of the clock tree prior to clock tree synthesis. After clock tree synthesis, the total clock latency from clock source to a clock pin of a flip-flop is the source latency plus the actual delay of the clock tree from the clock definition point to the flip-flop.

5.4 Generated Clocks

A generated clock is a clock derived from a master clock. A master clock is a clock defined using the create_clock specification.

When a new clock is generated in a design that is based on a master clock, the new clock can be defined as a generated clock. For example, if there is a divide-by-3 circuitry for a clock, one would define a generated clock definition at the output of this circuitry. This definition is needed as STA does not know that the clock period

has changed at the output of the divide-by logic, and more importantly what the new clock period is.

create_clock -name CLKP 10 [get_pins UPLL0/CLKOUT]
Create a master clock with name CLKP of period 10ns with 50% duty cycle at the CLKOUT pin of the PLL.

create_generated_clock -name CLKPDIV2 -source UPLL0/CLKOUT -divide_by 2 [get_pins UFF0/Q]
Creates a generated clock with name CLKPDIV2 at the Q pin of flip-flop UFF0. The master clock is at the CLKOUT pin of PLL. And the period of the generated clock is double that of the clock CLKP, that is, 20ns.

Defining a master clock instead of a generated clock creates a new clock domain. This is not a problem in general except that there are more clock domains to deal within setting up the constraints for STA. Defining the new clock as a generated clock does not create a new clock domain, and the generated clock is considered to be in phase with its master clock. The generated clock does not require additional constraints to be developed. Thus, one must attempt to define a new internally generated clock as a generated clock instead of deciding to declare it as another master clock.

Another important difference between a master clock and a generated clock is the notion of clock origin. In a master clock, the origin of the clock is at the point of definition of the master clock. In a generated clock, the clock origin is that of the master clock and not that of the generated clock. This implies that in a clock path report, the start point of a clock path is always the master clock definition point. This is a big advantage of a generated clock over defining a new master clock as the source latency is not automatically included for the case of a new master clock.

5.5 Typical Clock Generation Scenario

Figure below shows a scenario of how a clock distribution may appear in a typical ASIC. The oscillator is external to the chip and produces a low frequency (10-50 MHz typical) clock which is used as a reference clock by the on-chip PLL to generate a high-frequency low-jitter clock (200-800 MHz typical). This PLL clock is then fed to a clock divider logic that generates the required clocks for the ASIC.

On some of the branches of the clock distribution, there may be clock gates that are used to turn off the clock to an inactive portion of the design to save power when necessary. The PLL can also have a multiplexer at its output so that the PLL can be bypassed if necessary.

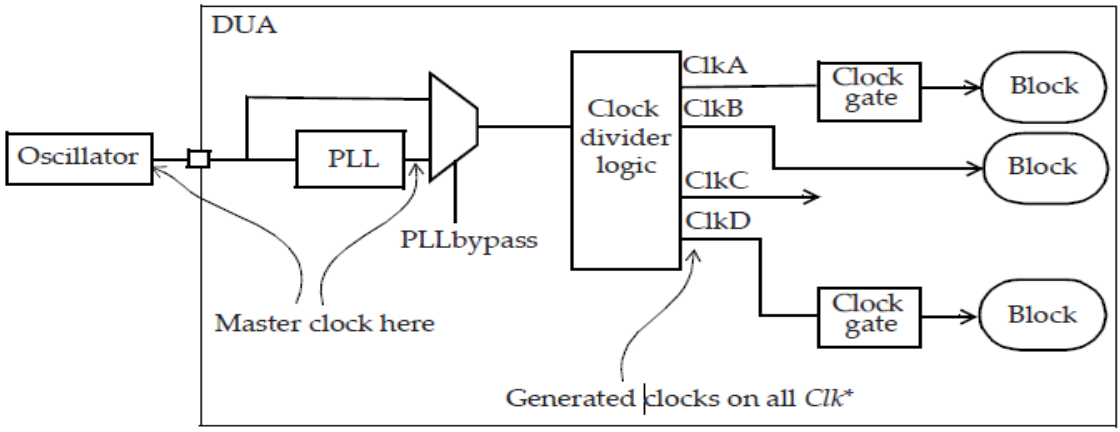


Figure 5.2: Clock distribution in a typical ASIC

A master clock is defined for the reference clock at the input pin of the chip where it enters the design, and a second master clock is defined at the output of the PLL. The PLL output clock has no phase relationship with the reference clock. Therefore, the output clock should not be a generated clock of the reference clock. Most likely, all clocks generated by the clock divider logic are specified as generated clocks of the master clock at the PLL output.

5.6 Timing Path Groups

Timing paths in a design can be considered as a collection of paths. Each path has a startpoint and an endpoint. In STA, the paths are timed based upon valid startpoints and valid endpoints. Valid startpoints are: input ports and clock pins of synchronous devices, such as flip-flops and memories. Valid endpoints are output ports and data input pins of synchronous devices. Thus, a valid timing path can be:

i. from an input port to an output port

ii. from an input port to an input of a flip-flop or a memory

iii. from the clock pin of a flip-flop or a memory to an input of flipflop

iv. from the clock pin of a flip-flop to an output port

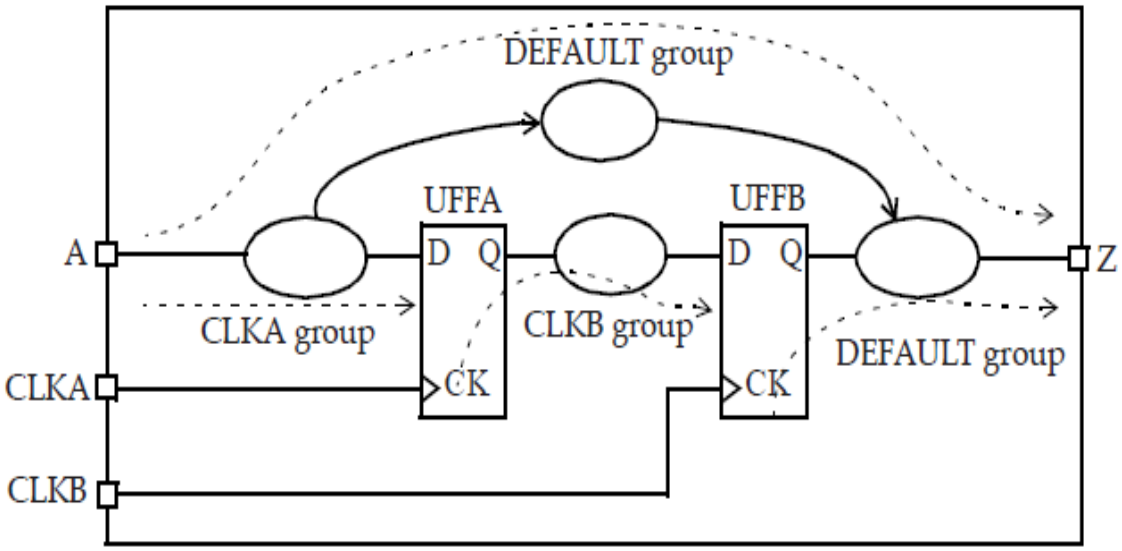


Figure 5.3: Path Groups

5.7 Virtual Clocks

A virtual clock is a clock that exists but is not associated with any pin or port of the design. It is used as a reference in STA analysis to specify input and output

delays relative to a clock. An example where virtual clock is applicable is shown in figure below. The design under analysis gets its clock from CLK_CORE, but the clock driving input port ROW_IN is CLK_SAD.

To specify the IO constraint on input port ROW_IN or STATE_O, a virtual clock can be defined with no specification of the source port or pin. In the example of figure below, the virtual clock is defined for CLK_SAD and CLK_CFG.

create_clock -name VIRTUAL_CLK_SAD -period 10 -waveform 2 8

create_clock -name VIRTUAL_CLK_CFG -period 8 -waveform 0 4

create_clock -period 10 [get_ports CLK_CORE]

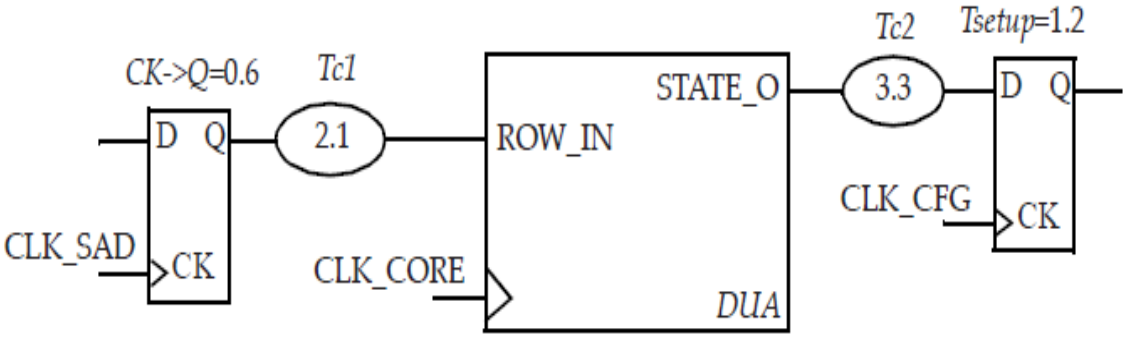


Figure 5.4: Virtual Clock Example

5.8 Refining the Timing Analysis

Four common commands that are used to constrain the analysis space are:

i. set_case_analysis: Specifies constant value on a pin of a cell, or on an input port.

ii. set_disable_timing: Breaks a timing arc of a cell.

iii. set_false_path: Specifies paths that are not real which implies that these paths are not checked in STA.

iv. set_multicycle_path: Specifies paths that can take longer than one clock cycle.

5.8.1 Specifying Inactive Signals

In a design, certain signals have a constant value in a specific mode of the chip. For example, if a chip has DFT logic in it, then the TEST pin of the chip should be at 0 in normal functional mode. It is often useful to specify such constant values to STA. This helps in reducing the analysis space in addition to not reporting any paths that are irrelevant. For example, if the TEST pin is not set as a constant, some odd long paths may exist that would never be true in functional mode. Such constant signals are specified by using the set_case_analysis specification.

set_case_analysis 0 TEST

set_case_analysis 0 [get_ports testmode[3]]

set_case_analysis 0 [get_ports testmode[2]]

set_case_analysis 0 [get_ports testmode[1]]

set_case_analysis 0 [get_ports testmode[0]]

If a design has many functional modes and only one functional mode is being analyzed, case analysis can be used to specify the actual mode to be analyzed.

set_case_analysis 1 func_mode[0]

set_case_analysis 0 func_mode[1]

set_case_analysis 1 func_mode[2]

Note that the case analysis can be specified on any pin in the design. Another common application of case analysis is when the design can run on multiple clocks, and the selection of the appropriate clock is controlled by multiplexers. To make STA analysis easier and reduce CPU run time, it is beneficial to do STA for each clock selection separately. Figure below shows an example of the multiplexers selecting different clocks with different settings. set_case_analysis 1 UCORE/UMUX0/CLK_SEL[0]

set_case_analysis 1 UCORE/UMUX1/CLK_SEL[1]

set_case_analysis 0 UCORE/UMUX2/CLK_SEL[2]

The first set_case_analysis causes PLLdiv16 to be selected for MIICLK. The clock

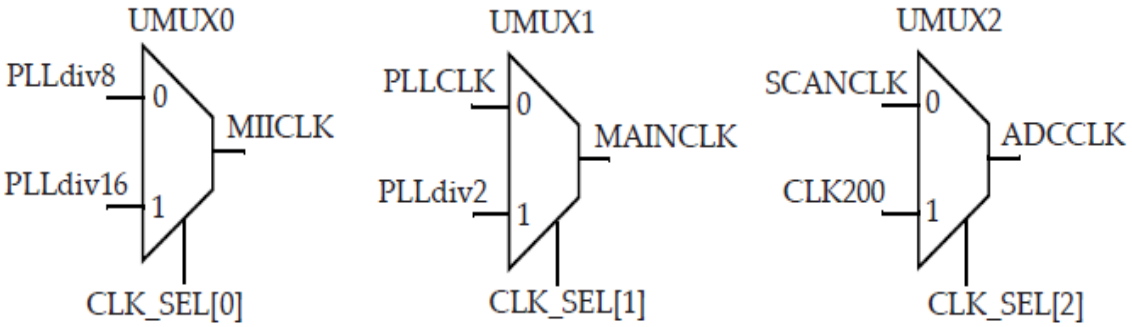


Figure 5.5: Selecting Clock mode for timing analysis

path for PLLdiv8 is blocked and does not propagate through the multiplexer. Thus, no timing paths are analyzed using clock PLLdiv8 (assuming that the clock does not go to any flip-flip prior to the multiplexer). Similarly, the last set_case_analysis causes SCANCLK to be selected for ADCCLK and the clock path for CLK200 is blocked.

5.8.2 Breaking Timing Arcs in Cells

Every cell has timing arcs from its inputs to outputs, and a timing path may go through one of these cell arcs. In some situations, it is possible that a certain path through a cell cannot occur. For example, consider the scenario where a clock is connected to the select line of a multiplexer and the output of the multiplexer is part of a data path. In such a case, it may be useful to break the timing arc between the select pin and the output pin of the multiplexer. An example is shown in figure below. The path through the select line of multiplexer is not a valid data path. Such a timing arc can be broken by using the set_disable_timing SDC command.

set_disable_timing -from S -to Z [get_cells UMUX0]

Since the arc no longer exists, there are consequently fewer timing paths to analyze. Another example of a similar usage is to disable the minimum clock pulse width check of a flip-flop.

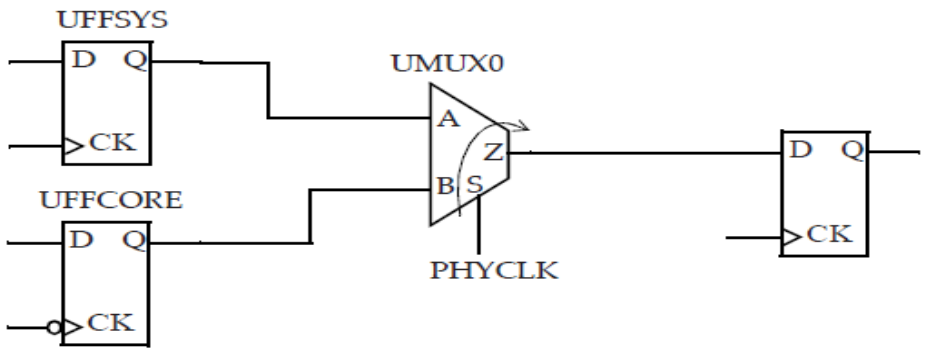


Figure 5.6: disable_timing example

One should use caution when using the set_disable_timing command as it removes all timing paths through the specified pins. Where possible, it is preferable to use the set_false_path and the set_case_analysis commands.

5.8.3 False Paths

It is possible that certain timing paths are not real (or not possible) in the actual functional operation of the design. Such paths can be turned off during STA by setting these as false paths. A false path is ignored by the STA for analysis. Examples of false paths could be from one clock domain to another clock domain, from a clock pin of a flip-flop to the input of another flip-flop, through a pin of a cell, through pins of multiple cells, or a combination of these. When a false path is specified through a pin of a cell, all paths that go through that pin are ignored for timing analysis. The advantage of identifying the false paths is that the analysis space is reduced, thereby allowing the analysis to focus only on the real paths. This helps cut down the analysis time as well. However, too many false paths which are wildcarded using the through specification can slow down the analysis. A false path is set using the set_false_path specification. Here are some examples.

set_false_path -from [get_clocks SCAN_CLK] -to [get_clocks CORE_CLK]

Any path starting from the SCAN_CLK domain to the CORE_CLK domain

is a false path.

set_false_path -through [get_pins UMUX0/S]

Any path going through this pin is false.

5.8.4 Multicycle Paths

In some cases, the combinational data path between two flip-flops can take more than one clock cycle to propagate through the logic. In such cases, the combinational path is declared as a multicycle path. Even though the data is being captured by the capture flip-flop on every clock edge, we direct STA that the relevant capture edge occurs after the specified number of clock cycles. Figure below shows an example. Since the data path can take up to three clock cycles, a setup multicycle check of three cycles should be specified.

The multicycle setup constraints specified to achieve this are given below.

create_clock -name CLKM -period 10 [get_ports CLKM]

set_multicycle_path 3 -setup -from [get_pins UFF0/Q] -to [get_pins UFF1/D]

The setup multicycle constraint specifies that the path from UFF0/CK to UFF1/D can take up to three clock cycles to complete for a setup check. This implies that the design utilizes the required data from UFF1/Q only every third cycle instead of every cycle.

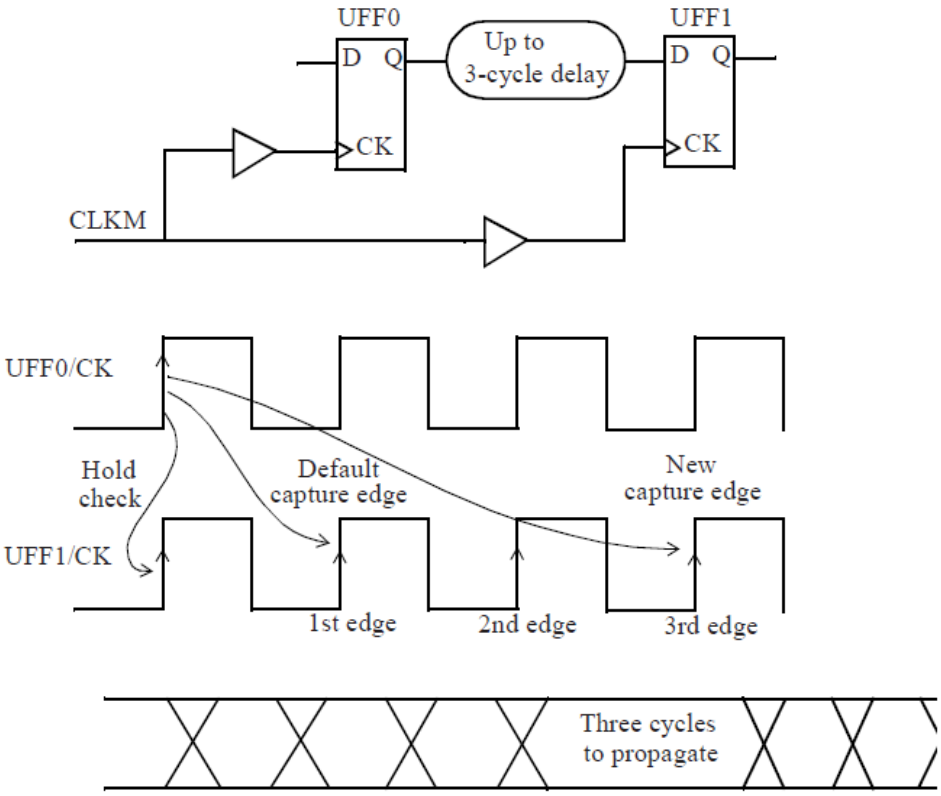


Figure 5.7: Multicycle path example

Chapter 6

Results

6.1 Results

Figure 6.1: Top Level Instantiation

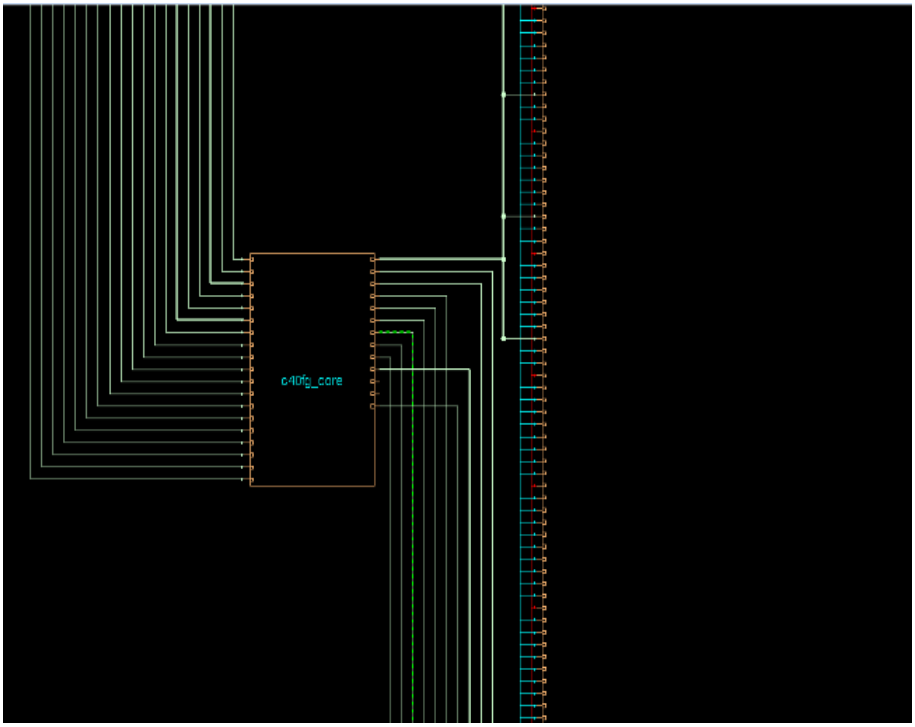


Figure 6.2: Core Module

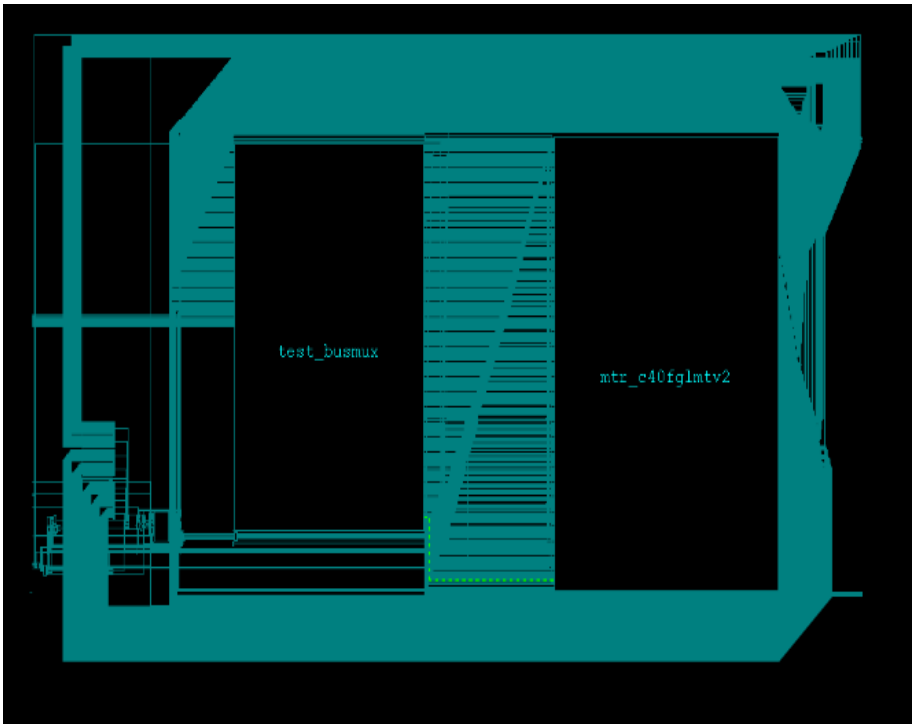


Figure 6.3: Core Submodules

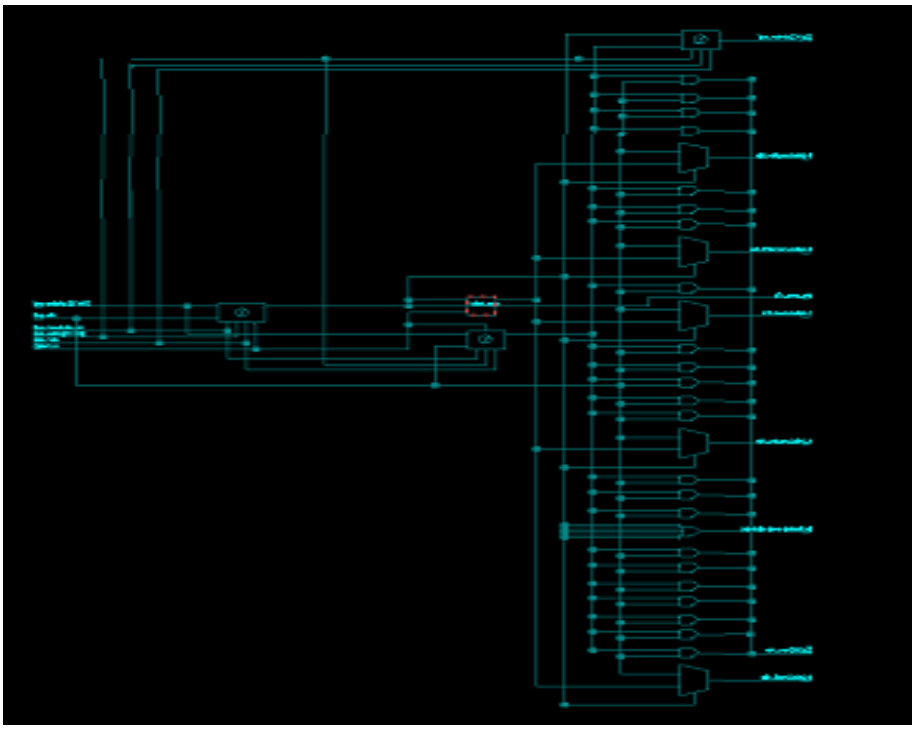


Figure 6.4: Clock Generator Block

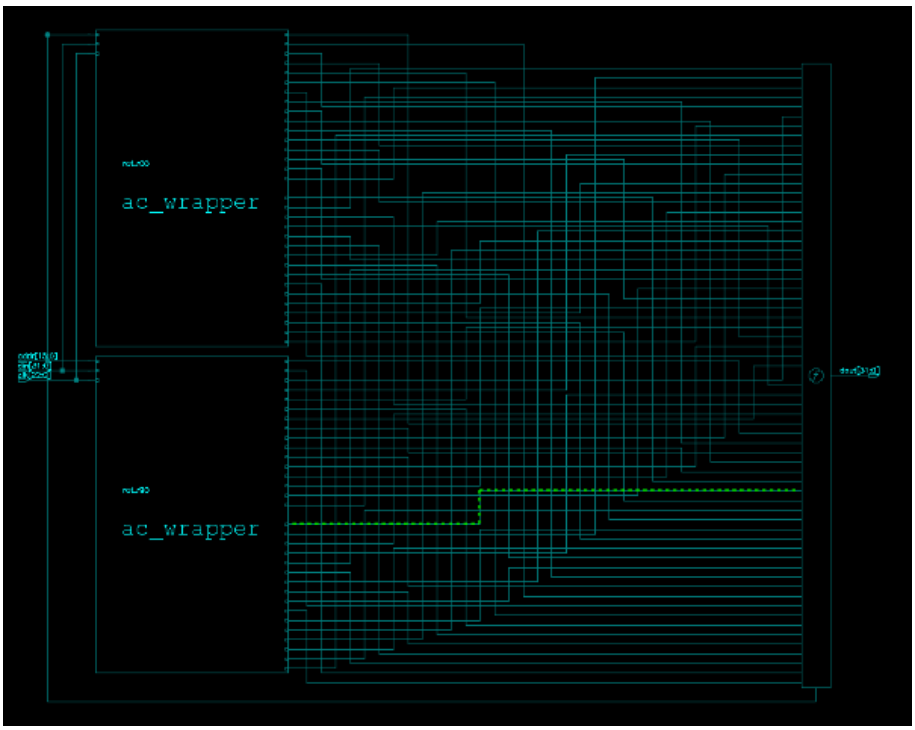


Figure 6.5: All Cell Block

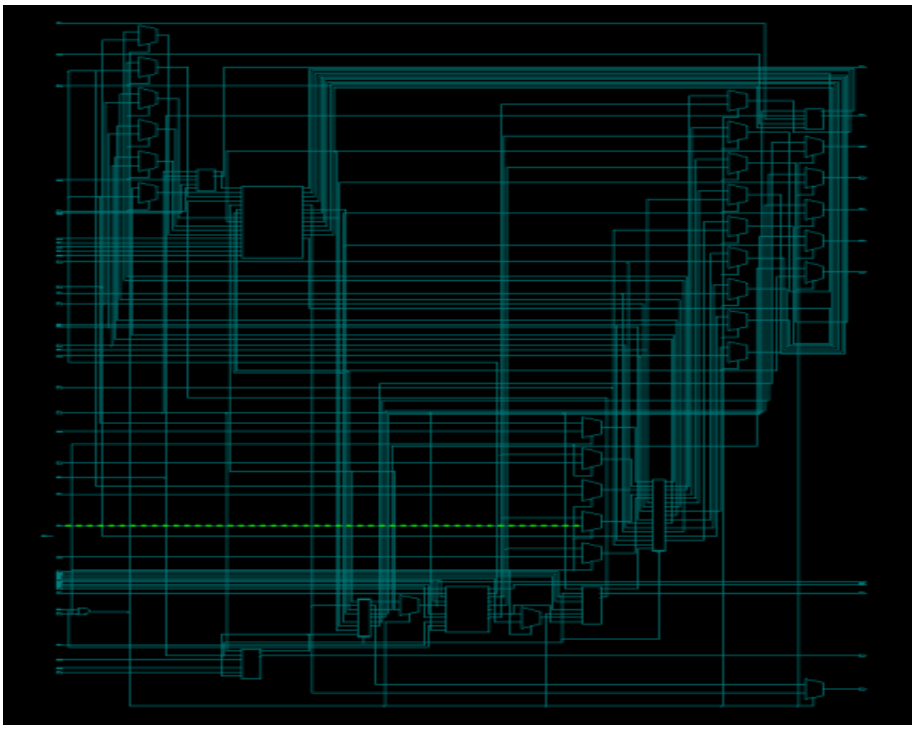


Figure 6.6: Memory Test and Repair Block

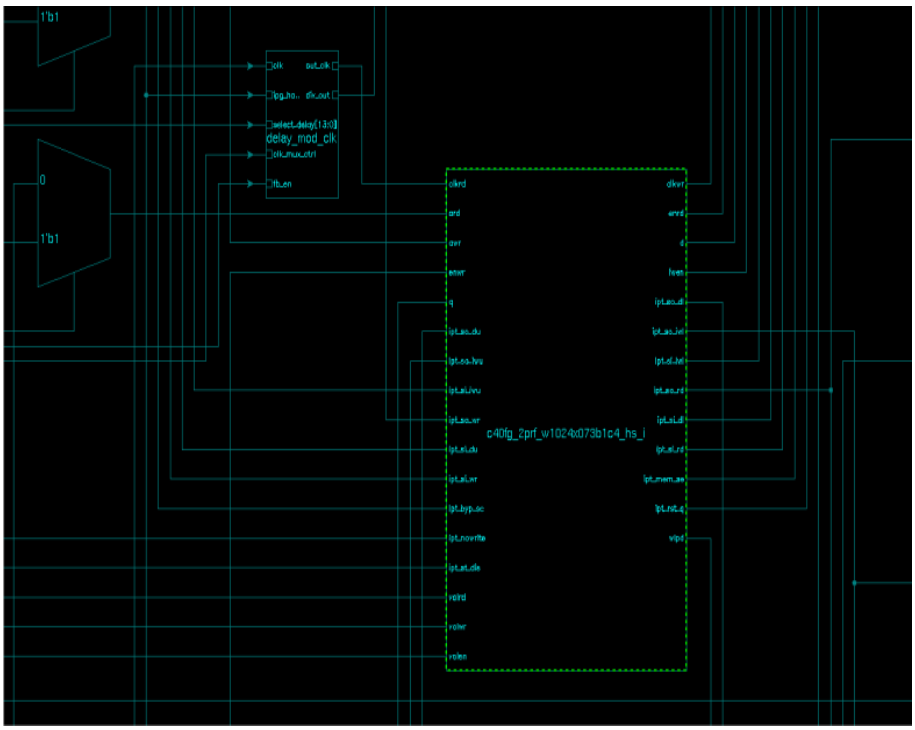
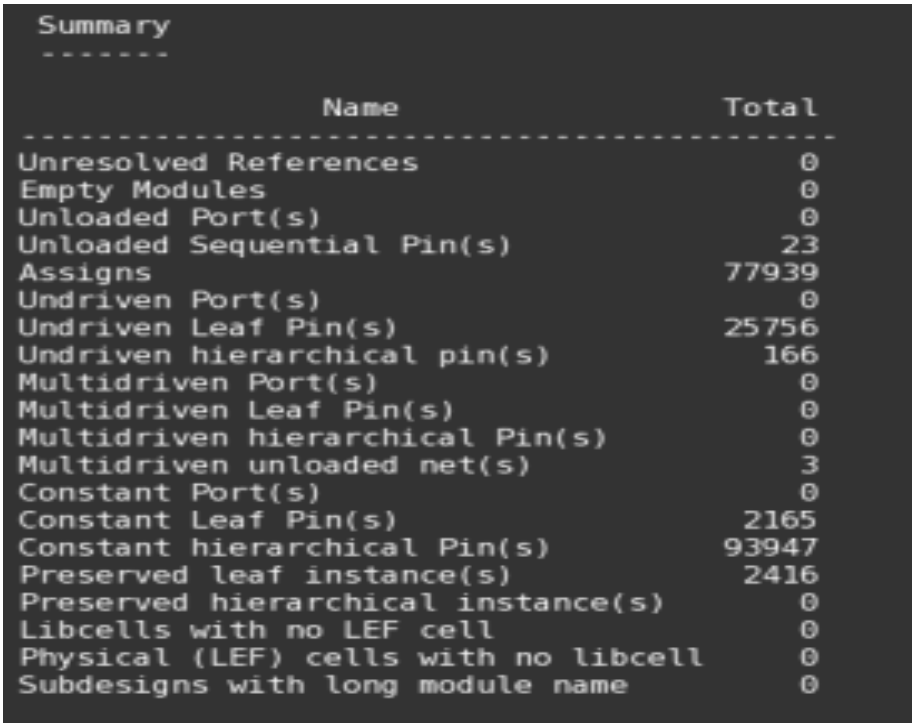


Figure 6.7: Memory Instance



Summary

Name	Total
Unresolved References	0
Empty Modules	0
Unloaded Port(s)	0
Unloaded Sequential Pin(s)	23
Assigns	77939
Undriven Port(s)	0
Undriven Leaf Pin(s)	25756
Undriven hierarchical pin(s)	166
Multidriven Port(s)	0
Multidriven Leaf Pin(s)	0
Multidriven hierarchical Pin(s)	0
Multidriven unloaded net(s)	3
Constant Port(s)	0
Constant Leaf Pin(s)	2165
Constant hierarchical Pin(s)	93947
Preserved leaf instance(s)	2416
Preserved hierarchical instance(s)	0
Libcells with no LEF cell	0
Physical (LEF) cells with no libcell	0
Subdesigns with long module name	0

Figure 6.8: Synthesis Summary Report

Chapter 7

Conclusion

7.1 Tasks handled

The following is a short description of tasks performed during internship phase:

- Performed Verification of LMTV Test-Chip.
  - Several TestCases were written for verification of standard cells(All_Cell block).
  - Testcases for verification of Memory Structures.
- Performed Synthesis at SoC level.
- Developed SDC file for Synthesis and Static Timing Analysis(STA).
- Executed RTL to GDS flow at Module level for All_Cell block.
- Performed simulation for analog block i.e MICA simulation.
- Learned Perl, shell and Tcl scripting languages.
- Understood the concepts of STA.

7.2 Conclusion

The test-structures that are implemented on silicon are used for characterization, which provides necessary data for analysis. The data available for analysis provides necessary information as to whether the structures implemented are suitable for realization in a particular technology node. Thus test chip SoC helps in advancement of technology to even lower technology nodes.

Bibliography

[1] Sung-Mo Kang & Yusuf Leblebici "Cmos Digital Integrated Circuits Analysis and Design"

[2] Jayaram Bhasker & Rakesh Chadha Static Timing Analysis for Nanometer Designs A Practical Approach

[3] IEEE Standard for Verilog Hardware Description Language by samir palnitkar

[4] FSL Internal Documents

[5] User Guides of Cadence RTL Compiler, Cadence Encounter Timing System, Cadence Encounter Digital Implementation

[6] IeeeXplore

[7] www.asic-world.com

[8] Technical papers on Multi-Project SoC