Design and Fabrication of an AMBA (ARM AHB-Lite Slave) Circuit Using CAD Tools at 350nm Technology

A thesis presented for the Faculty of the Department of Electrical and Computer Engineering in Partial Fulfillment of the Requirements for the Diploma of Science

Author:
Konstadinos Mourtziapis

Supervisors:
Dr. John Moondanos
Dr. Christos Sotiriou

Department of Electrical and Computer Engineering
UNIVERSITY OF THESSALY
Volos, Greece
July, 2015

UNIVERSITY OF THESSALY

Department of Electrical and Computer Engineering

Design and Fabrication of AMBA (ARM AHB-Lite Slave) Circuit Using CAD Tools at 350nm Technology

by
Mourtziapis Konstadinos

Graduate Thesis
for
the degree of
Diploma of Science in Computer and Communication Engineering

Approved by the two member inquiry committee at 10th of July 2015

....	....
Dr. John Moondanos	Dr. Christos Sotiriou

Declaration of Authorship

I, Mourtziapis Konstadinos, declare that this thesis titled, 'Design and fabrication of AMBA (ARM AHB-lite slave) circuit using CAD tools at 350nm technology' and the work presented in it are my own. The research was carried out wholly or mainly while in candidature for the graduate degree of Diploma of Science in Computer and Communication Engineering, at the University of Thessaly, Department of Electrical and Computer Engineering, Volos, Greece. No part of this thesis has been previously submitted for a degree or any other qualification at this University or any other institution. Wherever I have consulted or quoted from the work of others, it is always attributed and the source is given. The main sources of help are referenced in the Bibliography section of this thesis.

....

Mourtziapis Konstadinos

To Ioanna and my family

Acknowledgements

I would like to express my wholehearted gratitude to my supervisor, Dr.John Moondanos, who trusted me in dealing with this thesis. He did not lose hope in me even in the midst of many hardships. His patience, expertise and advising, contributed to my graduate experience and finally this thesis completion. I strongly appreciate his in depth knowledge and skills. The development of this project would not have been possible without his assistance.

Furthermore, I would like to thank my co-advisor Dr. Christos Sotiriou for his assistance and support he provided at all levels of this thesis.

Moreover, i would like to thank Ioanna for her moral and technical support that she gave me in order to complete my thesis.

Finally, I have to thank my family for their endless and invaluable moral support that offered me all those academic years.

Mourtziapis Konstadinos
Volos, 2015

Contents

List of Tables

List of Figures

Listings

Abbreviations

ACE AXI Coherency Extensions. 15

AHB Advanced High-performance Bus. xii, 3, 15–22, 24, 25, 32, 33, 35, 36, 60, 89

AMBA Advanced Microcontroller Bus Architecture. xii, 3, 15, 19, 36, 89, 130

AMS Austria Microsystems. xii, 11, 89

APB Advanced Peripheral Bus. 15, 19

ARM Advanced RISC Machines. xii, 3, 6, 15, 35, 36, 89

ASB Advanced System Bus. 15

ASCII American Standard Code for Information Interchange. 83

ASIC Application Specific Integrated Circuit. 11, 15

ATB Advanced Trace Bus. 15

ATPG Automatic Test Pattern Generation. 12, 63, 65, 89

AXI Advanced Extensible Interface. 15

BDD Binary Decision Diagram. 66

CHI Coherent Hub Interface. 15

CPU Central Processing Unit. 15

CTS Clock Tree Synthesis. xii, 2, 8, 77

CUT Circuit Under Test. 9, 10

DFT Design For Testability. xii, 2, 9, 60, 63, 130

DRC Design Rule Check. 9, 12, 55, 59, 62, 64, 77, 79–81, 90

DUT Device Under Test. 9, 37

EDA Electronic Design Automation. xii, 2, 3, 5, 7, 8, 12, 13, 36, 37, 83, 85, 87, 89

ERC Electrical Rule Checking. 9

FSM Finite State Machine. 66

GDS	Graphic Data System. 3, 8, 12, 80, 81, 83, 87, 89, 90
GPU	Graphics Processing Unit. 15
HDL	Hardware Description Language. 7, 37, 51, 59
IC	Integrated Circuit. xii, 2, 5, 6, 11, 15, 130
iLs	Internet Learning Series. 13
IP	Intellectual Property. 3, 15
LEC	Logic Equivalence Checking. 66
LEF	Library Exchange Format. 69
LSI	Large Scale Integration. 5
LVS	Layout vs. Schematic. 9
MMMC	Multi Mode Multi Corner. 69
MSI	Medium Scale Integration. 5
RTL	Register Transfer Level. xii, 3, 7, 12, 35–37, 49, 50, 60, 66, 67, 69, 71, 89, 130
SDC	Synopsys Design Constraints. 69
SDF	Standard Delay Format. 12
SoC	System on Chip. xii, 2, 3, 5, 8, 12, 15, 35, 71, 72, 77, 78, 81, 83–85, 89, 90, 130
SPEF	Standard Parasitic Exchange Format. 83
SSI	Small Scale Integration. 5
STA	Static Timing Analysis. 8, 83, 90
SVF	Serial Vector Format. 11
TCL	Tool Command Language. 12, 69
TSMC	Taiwan Semiconductor Manufacturing Company. 11
UUT	Unit Under Test. 7
VLSI	Very Large Scale Integration. xii, 3, 5, 89

Περίληψη

Η σχεδιαστική πολυπλοκότητα των Integrated Circuit (IC) (ολοκληρωμένων κυκλωμάτων) έχει αυξηθεί ραγδαία σε σχέση με τα πρώτα κυκλώματα στα τέλη της δεκαετίας του 50, όπου ο αριθμός των τρανζίστορ ήταν πού μικρότερος. Στις μέρες μας, τα System on Chip (SoC) κυκλώματα περιέχουν εκατομμύρια ή δισεκατομμύρια τρανζίστορ και όχι μόνο η εμπειρία του σχεδιαστή αλλά και η μεθοδολογία σχεδίασης με εργαλέια Electronic Design Automation (EDA) χρειάζεται για να καταλήξουμε στην κατασκευή ενός ολοκληρωμένου κυκλώματος.

Ο στόχος αυτής της διπλωματικής είναι η σχεδίαση και υλοποίηση σε τεχνολογία 350nm Austria Microsystems (AMS) ενός Advanced RISC Machines (ARM) Advanced Microcontroller Bus Architecture (AMBA) Advanced High-performance Bus (AHB)-lite slave με την χρήση εργαλείων CAD των τριών μεγαλύτερων κατασκευαστών (Cadence, Synopsys, Mentor Graphics) ακολουθώντας την τυπική βιομηχανική ροή σχεδίασης. Κατά τα διάφορα βήματα της σχεδιαστικής ροής Very Large Scale Integration (VLSI) (συμπεριφορά και επαλήθευση των προδιαγραφών του κυκλώματος, σύνθεση και εξαγωγή του τελικού layout), θα δείξουμε πως αντιμετωπίσαμε τα προβλήματα που προέκυψαν (Εισαγωγή Design For Testability (DFT), Κάλυψη λαθών, σύνθεση δέντρου ρολογιού, κτλ.)

Ξεκινώντας απο τον κώδικα Register Transfer Level (RTL) τον οποίο βρήκαμε στο website opencores, το τελικό αποτέλεσμα θα είναι ένα πλήρως λειτουργικό κύκλωμα Advanced RISC Machines (ARM) Advanced Microcontroller Bus Architecture (AMBA) Advanced High-performance Bus (AHB)-lite slave που θα σταλεί για κατασκευή στις 27th Ιουλίου σύμφωνα με Europractice 2015 MPW run schedule. Τέλος, η περίοδος του ρολογιού θα είναι στα 80,64 MHz (12.4ns), το τελικό μέγεθος $6,76mm^2$ και η τάση που θα λειτουργεί το κύκλωμα θα είναι 3,3 Volts.

Abstract

Integrated Circuit (IC) design complexity has increased radically since the first designs in the late 50s, with a few transistors. Nowadays, System on Chip (SoC) designs contain million or even billion transistors and not only the experience of the designer, but also Electronic Design Automation (EDA) tools and a design methodology is needed in order to reach at the fabrication of a chip.

The goal of this thesis is the design and implementation in 350nm Austria Microsystems (AMS) technology of an Advanced RISC Machines (ARM) Advanced Microcontroller Bus Architecture (AMBA) Advanced High-performance Bus (AHB)-lite slave using Electronic Design Automation (EDA) tools from all three big vendors(Cadence, Synopsys, Mentor Graphics)following a typical industrial tool flow. During the different steps of the Very Large Scale Integration (VLSI) design flow (behavioral specification and verification, synthesis and layout generation), it will be shown how to deal with the design issues that arise: (Design For Testability (DFT) insertion, test coverage, Clock Tree Synthesis (CTS), etc.)

Starting from the Register Transfer Level (RTL) code originating from opencores website, the final result is a fully functional and tested Advanced RISC Machines (ARM) Advanced Microcontroller Bus Architecture (AMBA) Advanced High-performance Bus (AHB)-lite slave circuit with no violations of any kind that will be sent for fabrication at 27th of July according to Europractice 2015 MPW run schedule. Finally, the clock frequency of the circuit will be at 83.3 MHz, the final area will be $7mm^2$ and the voltage that will operate will be at 5 Volts.

Chapter 1

Introduction

In the past people used technology as a tool for making survival an easier endeavor. Nowadays we are witnessing a shift in the role technology is playing in our society and may continue to play for generations to come. Every human out there in this age simply cannot imagine his life without technology and an internet connection. Some applications of technology evolution are helpful for humanity such as bio medical implants, others main concern is the improvement of living such as home applications while others are purely for entertainment such as smart phones, personal computers, tablets, smart TVs. As mentioned before there are many different electronic systems out there (Figure 1.1). In the scheme of continuous development, electronic devices becoming more and more complex (which sometimes means difficult to use), rather than the first Integrated Circuit (IC) developed in the late 50s with a few transistors design all by hand to current Integrated Circuit (IC) [4] and System on Chip (SoC) [6] with billions of transistors. The key point to that evolution was the development of Electronic Design Automation (EDA) tools. Another challenge that we have to face nowadays is the "shrinking" of the transistors. This "shrinking" is necessary because we want to place billions of transistors to a small area at the size of the head of a nail. The field of this thesis is to examine the parts(synthesis, Clock Tree Synthesis (CTS), Design For Testability (DFT), power distribution,...) of a puzzle that we have to solve in order to fabricate a chip.

(a) Smartphones

(b) Home automation

(c) Motor sports application

(d) Bio medical application

Figure 1.1: Electronic systems example

1.1 Motivation

Nowadays, many new companies started to develop their own Intellectual Property (IP) such as ARM [7] and provide them to other companies that needed to integrate them to their electronic devices. These IPs are developed with the help of Electronic Design Automation (EDA) tools in order to achieve the desired performance, power and area requirements. The EDA tools run algorithms developed from the experience of designers in the field of VLSI and they automate many parts of the design process that could take years if SoCs designed by hand. So, the need for people who know to handle these tools as well as having the relevant VLSI background is necessary to help the technology and our living become better.

1.2 Thesis Goals

The main goal of this thesis is to complete a VLSI design flow at the 350nm technology node with the latest versions of EDA tools which legally provided for academic use from Europractice agreement my department has, in order to set up a design environment for more complex designs. My secondary goals are :

- Ensure that after the modifications done, the circuit works perfectly and it has the same functionality as the one that is integrated in ARM [7] SoCs.
- Send the circuit after the design flow for fabrication to a foundry that cooperate with Europractice.

1.3 Thesis Sructure

This thesis is divided in six main Chapters which include:

- Chapter 2 provides the necessary background information in order the reader fully understands the development and the final results of this project. First of all, it gives the basic knowledge about methodology and tools used in the design, as well a sort description of the design library that is used.
- Chapter 3 presents the ARM [7] AMBA AHB protocol, explaining the functionality of the signals that compose the specific circuit this thesis is about.
- Chapter 4 is about the front end flow of the design. This includes at the beginning RTL coding and modifications needed to be done in order the circuit will be functional during simulations. Afterwards, is described the logic synthesis part of the flow and the constraints applied to the circuit. Finally, is described the scan chain insertion and test vectors extraction and the final tape-out checks.
- Chapter 5 presents the back end flow of the design starting from importing the design to the back end tool and finishing with the GDSII extraction which was sent for fabrication to the foundry.
- Chapter 6 and 7, summarizes the work done, the problems faced and the results generated and points to possible future work.

Chapter 2

Literature Review

2.1 Brief ICs History

Opinions diverge on who inverted the first IC: Jack Kilby of Texas Instruments or Robert Noyce of Fairchild Semiconductor. The matter is that both of them contributed to achieve the birth of IC in 1958 and initiate a revolution in the circuit design field. Starting with Small Scale Integration (SSI) devices, with tens of transistors in the early 1960s, and Medium Scale Integration (MSI) devices, with hundreds of transistors in the late 1960s, were relatively simple to test. However, in the 1970s, Large Scale Integration (LSI) devices, with thousands and tens of thousands of transistors, created a number of challenges when testing these devices. In the early 1980s, Very Large Scale Integration (VLSI) devices with hundreds of thousands of transistors were introduced. Steady advances in VLSI technology have resulted in devices with hundreds of millions of transistors and many new challenges for developers and designers.

Following the so-called Moore's law [Moore 1965] (Figure 2.1), the scale of ICs has doubled every 18 months. A simple example of this trend is the progression from SSI to VLSI devices. In the 1980s, the term "VLSI" was used for chips having more than 100,000 transistors and has continued to be used over time to refer to chips with millions and now hundreds of millions of transistors. The first designs were designed by hand with the aid of microscopes and based only on the experience of the designer, but the more the transistors inside the circuits increase the more impracticable was the design without any help. This help is known as Electronic Design Automation (EDA) tools and was an essential part of the next evolution step in IC, the Very Large Scale Integration (VLSI) age, which was initiated at the early 80s with circuits including hundreds of thousands of transistors.

Nowadays, this number increased beyond several billion transistors and we reached to System on Chip (SoC) age, what makes clear the need of EDA tools and the methodology or design flow that allows managing huge designs like that. This is a direct result of the steadily decreasing dimensions, referred to as feature size, of the transistors and interconnecting wires from tens of microns to tens of nanometers, with current submicron technologies based on a feature size of less than 100 nanometers (14 nm). The reduction in feature size has also resulted in increased operating frequencies and clock speeds; for example, in 1971, the first microprocessor ran at a clock frequency of 108 KHz, while current commercially available microprocessors commonly run at several gigahertz. [19]

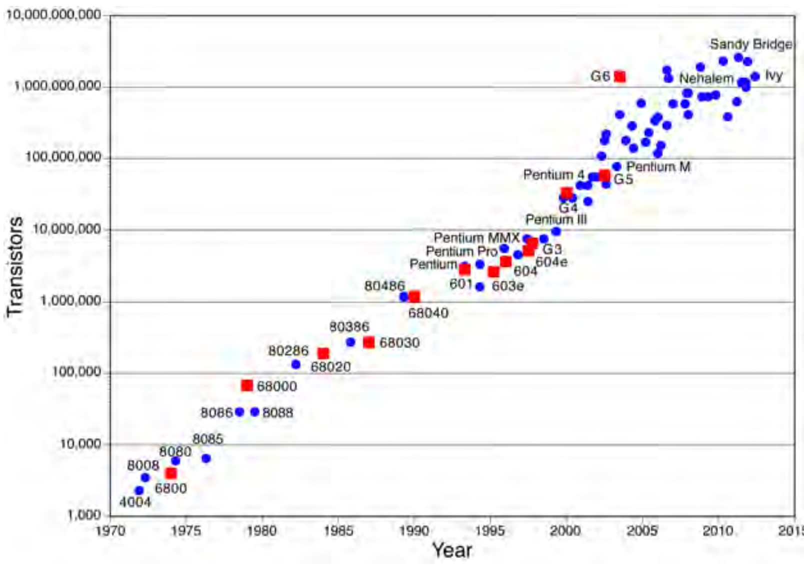


Figure 2.1: Moore's Law

2.2 VLSI Design Flow

As described in the previous section, the need for a methodology was needed in order to create a full design flow that would allow us to handle big designs. In this section, we will present this methodology as was developed through the years from many industries to get a final error-free chip. The diverse steps of the full design flow are shown in Figure 2.2

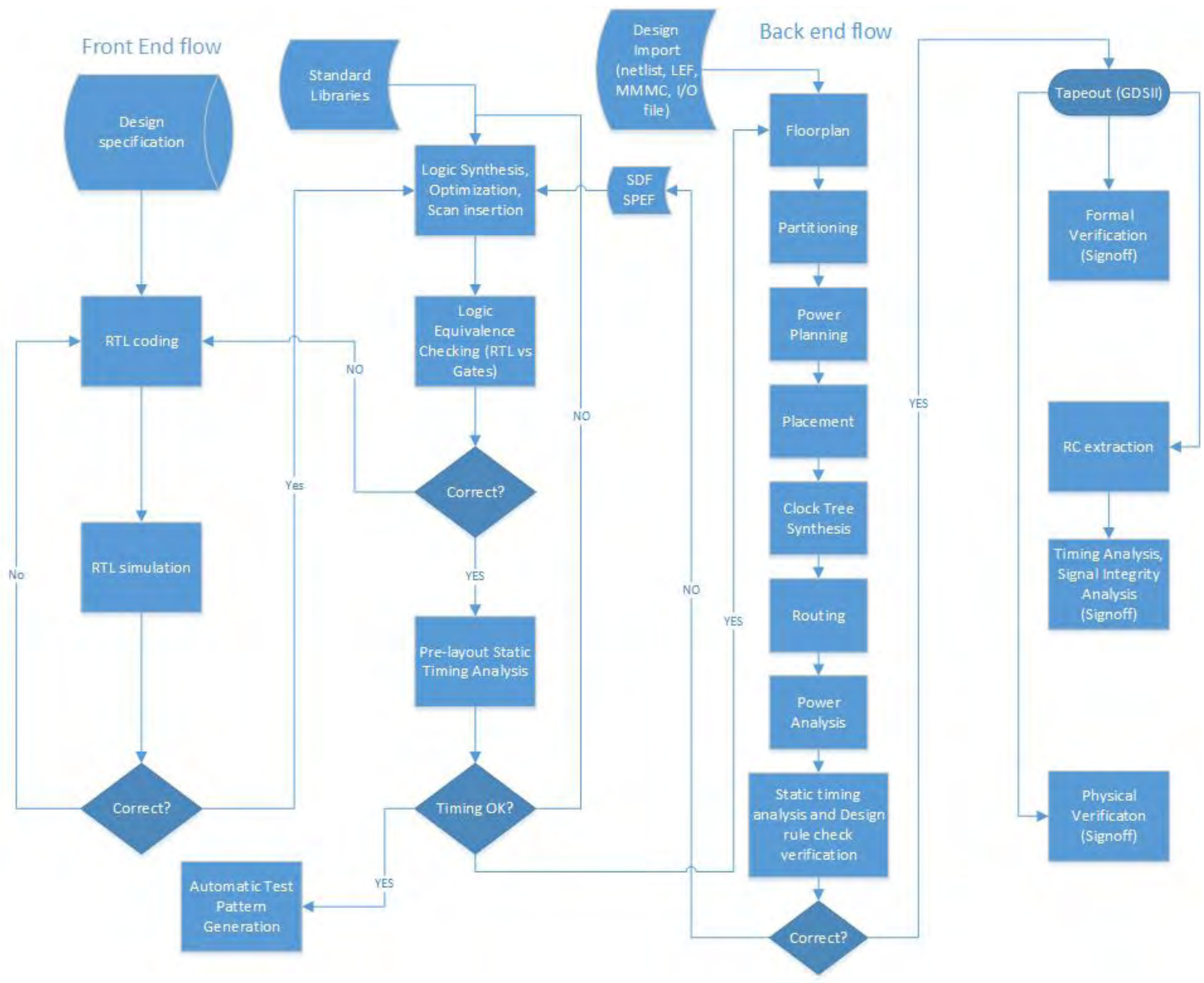

Front End flow
Design specification
Standard Libraries
Design Import (netlist, LEF, MMMC, I/O file)
Back end flow
Logic Synthesis, Optimization, Scan insertion
SDF SPEF
RTL coding
Logic Equivalence Checking (RTL vs Gates)
NO
NO
Correct?
Yes
YES
No
RTL simulation
Pre-layout Static Timing Analysis
Correct?
Timing OK?
YES
Automatic Test Pattern Generation
Floorplan
Partitioning
Power Planning
Placement
Clock Tree Synthesis
Routing
Power Analysis
Static timing analysis and Design rule check verification
Correct?
NO
YES
YES
Tapeout (GDSII)
Formal Verification (Signoff)
RC extraction
Timing Analysis, Signal Integrity Analysis (Signoff)
Physical Verificaton (Signoff)

Figure 2.2: Design Flow

2.2.1 Design Specification

Design Specifications are the first step of the Design flow. They describe the functionality, interface and the architecture of the digital IC circuit to be designed. They also contain block diagrams of the circuit, signal description and some examples of waveforms in order to understand the functionality of the circuit. Finally in some design specification we may find a pseudo-code description of some functionality of the circuit. In this project the design specifications taken from ARM [7] website.

2.2.2 RTL Description

Register Transfer Level (RTL) of a models the design in terms of the flow of digital signals between registers and logical operations performed on those signals. Register Transfer Level (RTL) abstraction is used in Hardware Description Language (HDL) like verilog to create a high-level representation of the circuit, from which lower-level representations are ultimately actual wiring can be derived. A synchronous circuit consists of two kinds of elements: registers and combinational logic. Registers (usually implemented as D flip-flops) synchronize the circuit's operation to the edges of the clock signal, and are the only elements in the circuit that have memory properties. Combinational logic performs all the logical functions in the circuit and it typically consists of logic gates. [5]

2.2.3 Front End Flow

Front End Flow of the design consists of RTL simulation and Logic synthesis of the circuit.

2.2.3.1 RTL simulation

After the RTL description of the design we have to check its functionality and if the code we composed works fine and gives us the behavior of the circuit according to the specifications. To achieve that we create test benches. Test bench is a specification written in verilog that plays the role of a complete simulation environment for the analyzed system (Unit Under Test (UUT)). A test bench contains both the UUT as well as stimuli for the simulation. The UUT is instantiated as a component of the test bench and the architecture of the test bench specifies stimuli for the UUT's ports, usually as waveform assigned to all output and bidirectional ports of the UUT. The entity of a test bench does not have any ports as this serves as an environment for the UUT. All the simulation results are reported using the assert and report statements. The correctness of the behavior is checked by a simulation program (Modelsim from Mentor Graphics, Simvision from Cadence or VCS from Synopsys) using test benches.

2.2.3.2 Synthesis

After checking the correctness of the RTL, the next step is synthesis. Synthesis is the process in which the circuit is converted from a high-level description of design into an optimized gate-level representation. Synthesis includes the following parts:

- **RTL Synthesis, Library Mapping:** The code written in verilog HDL format is translated to a netlist representation of interconnected gates such as AND, OR, NOT, macro cells such as MUXes, adders and registers which mapped to the gates defined in the target library.
- **Design Constraints:** The designer after importing the design in the EDA synthesis tool has to set constraints such as timing, area and power that the design should meet in order to agree with the specifications given at the beginning.

2.2.3.3 Post-synthesis verification

This step has to do with the post synthesis verification of the design. After getting the verilog netlist exported from EDA synthesis tool, we have to check the correctness of the circuit and if the synthesized netlist is the same in terms of logic with the initial RTL code we wrote. This can be done with running again testbenches that have been used before and checking the design with logic equivalence checking EDA tools.

2.2.3.4 Static Timing Analysis

After we check the design for its behavioral and structural correctness, we have to do some final timing tests. A Static Timing Analysis (STA) can be run to check if the circuit meets the constraints we set in the section 2.2.3.2. If not, we should return to synthesis part of the flow to set new constraints that can reach the design requirements.

2.2.4 Back End Flow

Back end flow is the flow that produces the final files which will sent to the foundry in order to fabricate a System on Chip (SoC). We take the netlist exported from the front end flow and we generate the final Graphic Data System (GDS)II file.

2.2.4.1 Layout generation

In the next steps is briefly described the main back end flow:

- **Floorplaning:** The first thing to do is to import the design and configure the floorplan of the design. The design is imported in hierarchical modules and we have to customize the area that is given from the EDA tool to perform the placement of these modules in this area.
- **Placement:** At this stage, modules are decomposed into standard cells and they placed inside the area we defined before. The EDA tool performs iterations with the aid of integrated algorithms which places the standard cells in a way to minimize the wire length of the design, the area the design should take and to decrease as much as possible the delay on the critical paths and achieve the best possible results in timing.
- **Routing:** Moving on, the placed cells need to be interconnected. There are two types of routing, global routing and detailed routing. Global routing allocates routing resources that are used for connections. Detailed routing assigns routes to specific metal layers and routing tracks within the global routing resources.
- **Clock Tree Synthesis and Timing Analysis:** After place and routing the design the static timing analysis reports will show us if the timing requirements of our design is accomplished or not. If not, Clock Tree Synthesis (CTS) may save the day. The goal of Clock Tree Synthesis (CTS) is to minimize skew and insertion delay. Clock is not propagated before CTS. After CTS hold slack should improve. The clock signal has to reach to all sequential elements of the circuit at the same time in order to achieve a correct functionality and avoid clock inaccuracy problems: clock skew and clock jitter.
- **Power Analysis and finishing:**During this step we check our final design in terms of power distribution before we proceed to tape-out.

2.2.4.2 Physical Verification

Physical verification checks the correctness of the generated layout design. This includes verifying that the layout:

- Complies with all technology requirements – Design Rule Check (DRC)
- Is consistent with the original netlist – Layout vs. Schematic (LVS)
- Has no antenna effects – Antenna Rule Checking
- This also includes density verification at the full chip level.Cleaning density is a very critical step in the lower technology nodes
- Complies with all electrical requirements – Electrical Rule Checking (ERC)

Some of these physical verification steps may offered by the foundry while others needed to be done from the designer.

2.3 Design for Test

Design For Testability (DFT) stands for IC design techniques that add certain testability features to a hardware product design. The premise of the added features is that they make it easier to develop and apply manufacturing tests for the designed hardware. The purpose of manufacturing tests is to validate that the product hardware contains no manufacturing defects that could, otherwise, adversely affect the product correct functioning. The tests generally are driven by test programs that execute in Automatic Test Equipment (ATE). In addition to finding and indicating the presence of defects (i.e., the test fails), tests may be able to log diagnostic information about the nature of the encountered test fails. The diagnostic information can be used to locate the source of the failure. In other words, the response of vectors(patterns) from a good circuit is compared with the response of vectors(using same patterns) from a Device Under Test (DUT). If the response is the same or matches, the circuit is good. Otherwise, the circuit is not manufactured as it is intended to do so.

DFT has two main parts:

- **Controllability:** It is the ability to control the nodes in a circuit by a set of inputs. For a given set of input and output pins, when we give the system a set of input test vectors, we should be able to control each node we want to test. The higher the degree of controllability, the better.
- **Observability:** It is the ease with which we can observe the changes in the nodes (gates). Like in the previous case the higher the observability, the better. What I mean by saying higher is that, we can see the desired state of the gates at the output in lesser number of cycles.

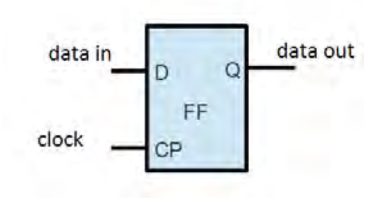
The most common method for delivering test data from chip inputs to internal Circuit Under Test (CUT)s, and observing their outputs, is called scan-design. In scan-design, registers (flip-flops or latches) in the design are connected in one or more scan chains, which are used to gain access to internal nodes of the chip. Test patterns are shifted in via the scan chain(s), functional clock signals are pulsed to test the circuit during the "capture cycle(s)", and the results are then shifted out to chip output pins and compared against the expected "good machine" results. [2]

2.3.1 Scan Chain insertion - Multiplexed Flip-Flop Style

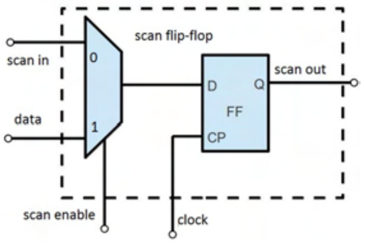
The most common method for delivering test data from chip inputs to internal Circuit Under Test (CUT)s, and observing their outputs, is called scan-design. There are two methodologies:

- **Partial Scan:** Only some Flip-Flops are changed to Scan-FF.
- **Full Scan:** Entire Flip Flops in the circuit are changed to these special Scan FF. This does mean that we can test the circuit 100%.

If a design contains sequential components (like flip/flops), the design compiler can be used to replace these components with what is known as ‘scan-cells.’ A common example is replacing a d-flop-flop with a multiplexed flip-flop (a flip flop with a MUX in front of it). A multiplexed input allows us to load test data into a flip/flop as opposed the regular data, simply by using the ‘select line’ of the mux. When the select is low, the normal data that was meant to go into the flip-flop passes through to Q, but when the select is high, test data propagates through the flip-flop to the Q output(Figure 2.3).



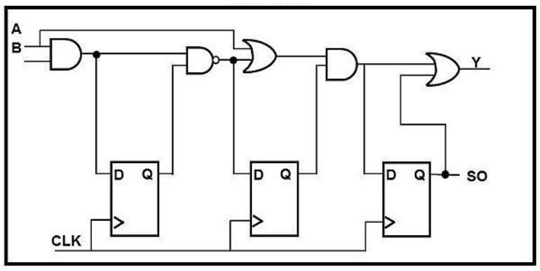
(a) Normal Flip-Flop



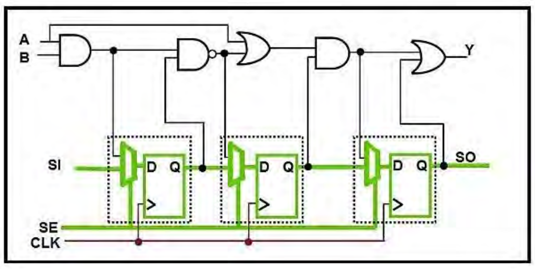
(b) Multiplexed Flip-Flop

Figure 2.3: Non scan vs. Multiplexed Flip-Flop Style

By replacing all of the flip-flops in a large design (assuming there are flip-flops throughout the design) with multiplexed flip-flops (scan-cells), we can increase the observability/controllability of all the non-sequential logic throughout the chip. The ‘test data-input’ and the ‘select’ lines of the scan-cells are wired to input pins; this creates what is known as a scan-chain. Now the user can put the chip into ‘scan-mode’ and load data into all the scan-cells. One can think of all the flip-flops in the design as having been stitched together into a giant ‘shiftregister’ allowing data to be inputted in at 1 pin (serially) and being able to move test data all over the chip, making the non-sequential portions of the design more observable/controllable then they were before (Figure 2.4).



(a) Non-scan



(b) Scan

Figure 2.4: Non-scan vs. Scan mode

Straightforward application of scan techniques can result in large vector sets with corresponding long tester time and memory requirements. Test compression techniques address this problem, by decompressing the scan input on chip and compressing the test output. The output of a scan design may be provided in forms such as Serial Vector Format (SVF), to be executed by test equipment.

2.4 Standard Cell Design

As mentioned in the section 2.2 the goal of fabricating a chip is to produce the final layout of the design. There are two ways to do that, so IC physical design is categorized into:

- **Full Custom:** Designer has full flexibility on the layout design, no predefined cells are used. That means that the designer has to generate the layout of each standard cell specifications he will afterwards use at his design. This approach allows maximizing the performance of the circuit, but it is a time-consuming process if we want to apply it to big designs.
- **Semi Custom:** Pre-designed library cells are used. These pre-designed library cells or standard cells, created by using a full custom technique from foundries and developers, can be seen by the designer in abstract logic representation (logic gates, buffers, flip-flops, etc.) In this case the designer has flexibility in placement of the cells and routing focusing on the high-level aspect of the design.

In this thesis, the second way is used, so a library containing the standard cells should be provided. The library that has been used in this design is explained at the next section.

2.4.1 AMS 350nm Standard Cell Library

The AMS C35B4C3 [9] (**Key specification:** C35B4C3 2P/4M 3.3V CMOS 4 Metal, Mixed Signal, PIP, high-res poly, 5V periphery) library from Austria micro systems used in this thesis is a 350nm standard cell library. AMS 350nm CMOS process family is fully compatible to the 350nm mixed signal base process licensed from Taiwan Semiconductor Manufacturing Company (TSMC). The high density CMOS standard cell library optimized for synthesis and 3 and 4 layer routing guarantees highest gate densities. Peripheral cell libraries are available for 3.3 V and 5 V with high driving capabilities and excellent ESD performance. Qualified digital macro blocks (RAM, diffusion programmable ROM and DPRAM) are available on request. A variety of high performance analog-to-digital and digital-to-analog converters can be provided for integration on the same Application Specific Integrated Circuit (ASIC). Digital, Analog and Mixed Signal Systems. The features of the library are:

- 350nm CMOS polycide-gate process
- Four unrestricted layers of metal
- Second layer of poly for linear capacitors and linear resistors
- Peripheral cells with high driving capability
- High performance digital and mixed signal capabilities

and some general characteristics are:

- p substrate
- Pseudo twin-well CMOS
- stacked contact,via, via2, via3
- Minimum Feature Size: 350nm gates
- Supply Voltage: CMOS 3.3 V; periphery up to 5.5 V
- Gate Delay: 0.10 ns (NAND2 typical)

The library contained all the necessary files for all the tools described in the section below.

2.5 EDA Tools

In this section we give a brief presentation of EDA tools that have been used during the VLSI design flow:

- **Simulation:** Questasim(version 10.4) from Mentor Graphics, VCSMX (version 2014.12) from Synopsys and Simvision (Incisive suite version 14.10) from Cadence. These three tools provides the designer the ability to verify the functionality of the circuit at logic level. They also include timing information through Standard Delay Format (SDF) file, they support multi-language simulation (VHDL, Verilog, System Verilog) as well as Tool Command Language (TCL) scripting, among other features.

- **Synthesis:** Design Compiler (version 2014.09-SP2) from Synopsys. It is the most commonly used EDA tool for that purpose. Design Compiler transform the RTL code given from an abstract description to logic gate circuit mapped with the standard cell library that the designer decides to provide. It offers a robust environment that faces successfully design challenges such as: timing, area, power, scan chain insertion.

- **Automatic Test Pattern Generation (ATPG):** TetraMax (version 2014.09-SP2) from Synopsys. The tool provides accurate measurements for test coverage of the design and it exports the test vectors that will be used to test the design after the fabrication process.

- **Logic Equivalence Checking:** Conformal (version 14.10) from Cadence. After synthesis and after layout generation, the netlists exported needed to be tested for their logic equivalence with the initial RTL code. An error in this stage means that the final netlist has not the same functionality as the initial RTL code, so we need to fix these errors or rerun synthesis and back-end flow from scratch.

- **Static Timing Analysis:** Tempus (version 14.10) from Cadence and PrimeTime (version 2014.12) from Synopsys. For more accurate results in terms of timing after syntheis and after layout generation these two tools were used in order to ensure that the final design has no timing violation and it is ready to send to the foundry for fabrication.

- **Layout generation:** System on Chip (SoC) Encounter (version 14.13) from Cadence completes the VLSI design flow. It performs floorplanning, power planning, placement, routing, clock tree synthesis, power analysis, DRC checks, in order to produce the final Graphic Data System (GDS)II file for the final tape-out of the circuit. The GDSII file is afterwards send to the foundry in order to begin the fabrication process of the System on Chip (SoC)

2.5.1 Deal With Problems

All these tools described before are delivered with their huge documentation and manuals but they just describe each command of each tool with no direction for the design flow at least the minimum one that has to be followed. This problem applies to all EDA tools from three biggest vendors used in this thesis (Synopsys, Cadence and Mentor graphics) because they are expensive programs, mostly used in companies, and therefore, it is difficult to get information about how to solve specific problems. The first help came from the department because of the agreement it has with Europractice which legally provides the latest versions of EDA tools for academic use, as well as official tutorials, labs sessions and lectures from Cadence and Synopsys. After spending many hours of training sessions, there were still some issues i faced that many google searches helped me to solve. Summarizing the main help provided by:

- **Synopsys tools:** After checking the manuals, Solvenet can be used. It is the online resource for Synopsys tool support and downloads, that offers access to the Synopsys knowledge data base containing up-to-date product manual, webinars and labs.
- **Cadence tools:** Cadence provides lab and lecture sessions to universities that have an agreement through Europractice. iLs provides lab sessions that covers pretty much the whole design flow with Cadence EDA tools.
- For Mentor Graphics Modelsim some tutorials and scripts from google search were very useful and helped me understand in short time the tool.
- The rest of the problems solved from google search some other websites/forums that can be really helpful for clarifying some concepts. A small sample of them are: www.edacafe.com, www.edaboard.com, www.deepchip.com.

Chapter 3

ARM AMBA AHB

3.1 General on ARM AMBA

The ARM [7] Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in System on Chip (SoC) designs. It facilitates development of multi-processor designs with large numbers of controllers and peripherals. Since its inception, the scope of AMBA has, despite its name, gone far beyond micro controller devices. Today, AMBA is widely used on a range of ASIC and SoC parts including applications processors used in modern portable mobile devices like smartphones. AMBA is a registered trademark of ARM Ltd. AMBA was introduced by ARM in 1996. The first AMBA buses were Advanced System Bus (ASB) and Advanced Peripheral Bus (APB). In its second version, AMBA 2, ARM added AMBA Advanced High-performance Bus (AHB) that is a single clock-edge protocol. In 2003, ARM introduced the third generation, AMBA 3, including AXI to reach even higher performance interconnect and the Advanced Trace Bus (ATB) as part of the CoreSight on-chip debug and trace solution. In 2010 the AMBA 4 specifications were introduced starting with AMBA 4 AXI4, then in 2011 extending system wide coherency with AMBA 4 ACE. In 2013 the AMBA 5 Coherent Hub Interface (CHI) specification was introduced, with a re-designed high-speed transport layer and features designed to reduce congestion [1].

3.1.1 Design Principles

The important aspect of a SoC is not only which components or blocks it houses, but also how they interconnect. AMBA is a solution for the blocks to interface with each other [?]. The objective of the AMBA specification is to:

- facilitate right-first-time development of embedded microcontroller products with one or more CPUs, GPUs or signal processors
- be technology independent, to allow reuse of IP cores, peripheral and system macrocells across diverse IC processes
- encourage modular system design to improve processor independence, and the development of reusable peripheral and system IP libraries
- minimize silicon infrastructure while supporting high performance and low power on-chip communication.

3.1.2 AMBA Protocol Specifications

The AMBA specification defines an on-chip communications standard for designing high-performance embedded micro-controllers. It is supported by ARM Limited with wide cross-industry participation. A simple transaction on the AHB consists of an address phase and a subsequent data phase (without wait states: only two bus-cycles). Access to the target device is controlled through a MUX (non-tristate), thereby admitting bus-access to one bus-master at a time. AHB-lite protocol is a subset of AHB formally defined in the AMBA 3 specifcation. This design simplifies the design with a single master [1].

3.2 ARM AMBA AHB-Lite

AHB-Lite [7] is a subset of the full AHB specification and is intended for use in designs where only a single bus master is used. This can either be a simple single master system, as show in Figure 3.1, or a multi-layer AHBsystem where there is only one AHB master per layer.

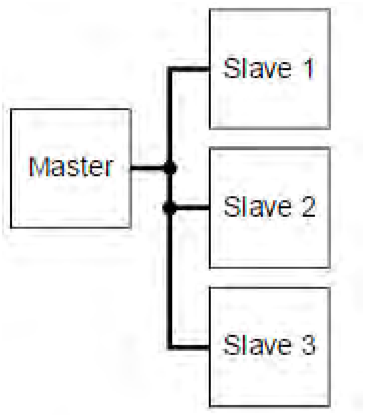


Figure 3.1: AHB-lite single master system

AHB-Lite simplifies the AHB specification by removing the protocol required for multiple bus masters, and includes:

- request and grant protocol to the arbiter
- SPLIT and RETRY responses from slaves.

Masters designed to the AHB-Lite interface specification can be significantly simpler in terms of interface design, compared to a full AHB master. AHB-Lite enables faster design and verification of these masters and the addition of a standard off-the-shelf bus mastering wrapper can be used to convert an AHB-Lite master for use in a full AHB system.

Any master that is already designed to the full AHB specification can be used in an AHB-Lite system with no modification. The majority of AHB slaves can be used interchangeably in either an AHB or AHB-Lite system. This is because AHB slaves that do not use either the SPLIT or RETRY response are automatically compatible with both the full AHB and the AHB-Lite specification. It is only existing AHB slaves that do use SPLIT and RETRY responses that require an additional standard off-the-shelf wrapper to be used in an AHB-Lite system. Any slave designed for use in an AHB-Lite system works in both a full AHB and an AHB-Lite design.

The AHB-Lite specification differs from the full AHB specification in the following ways:

- It is a single-master system. There is only one source of address, control, and write data, so no master-to-slave multiplexor is required.
- There is no arbitration. The AHB-Lite master always has control of the bus.
- There is no master HBUSREQ output. If such an output exists on a master, it is left unconnected.
- There is no master HGRANT input. If such an input exists on a master, it is tied HIGH.
- There is no SPLIT or RETRY slave responses. The AHB-Lite master deals only with a slave ERROR response.
- The AHB-Lite lock signal HMASTLOCK, compared with HLOCK for full AHB, and it has the same timing as the address bus and other control signals. If a master has an HLOCK output, it can be retimed to generate HMASTLOCK.
- The AHB-Lite lock signal, HMASTLOCK, must remain stable throughout a burst of transfers, in the same way that other control signals must remain constant throughout a burst.

Using the AHB-Lite interface makes the bus transfers generated by the AHB-Lite file reader easier to understand and easier to debug. Because the AHB-Lite is a single master protocol, an AHB-Lite master always has control of the bus. Unlike AHB, AHB-Lite has no request phase. Consequently, the AHB-Lite bus might be subject to wait states during the request phase of the AHB bus.

An **AHB-Lite master** has the same signal interface as a full AHB bus master, except that it does not support HBUSREQx and HGRANTx. The Lock functionality is still required because the master might be performing a transfer to a multi-port slave. The slave must be given an indication that no other transfer should occur to the slave when the master requires locked access. An AHB-Lite master is not required to support either the SPLIT or RETRY response and only the OKAY and ERROR responses are required, so the AHB-Lite master interface does not require the HRESP input.

The **advantage** of using the AHB-Lite protocol is that the bus master does not have to support the following cases:

- Losing ownership of the bus. The clock enable for the master can simply be derived from the HREADY signal on the bus.
- Early terminated bursts. There is no requirement for the master to rebuild a burst due to early termination, because the master always has access to the bus.
- SPLIT or RETRY transfer responses. There is no requirement for the master to retain the address of the last transfer to be able to restart a previous transfer.

A standard wrapper is available to **convert** an AHB-Lite master to make it a full AHB master. This wrapper adds support for the features described in AHB-Lite advantages. Because the AHB-Lite master has no bus request signal available, the wrapper generates this directly from the HTRANS signals.

AHB slaves that do not use either the SPLIT or RETRY response can be used in either a full AHB or AHB-Lite system. Any slave that does use SPLIT or RETRY responses can be used in an AHB-Lite system by adding a standard wrapper. This wrapper provides the ability to store the previous transfer in the case of a SPLIT and RETRY response and restart the transfer when appropriate. This wrapper is very similar to that required to convert an AHB-Lite master for use in a full AHB system. For compatibility with Multi-layer AHB, it is required that all AHB-Lite slaves still retain support for early terminated bursts. Figure 3.2 shows a more detailed block diagram, including Decoder and slave-to-master multiplexor connections.

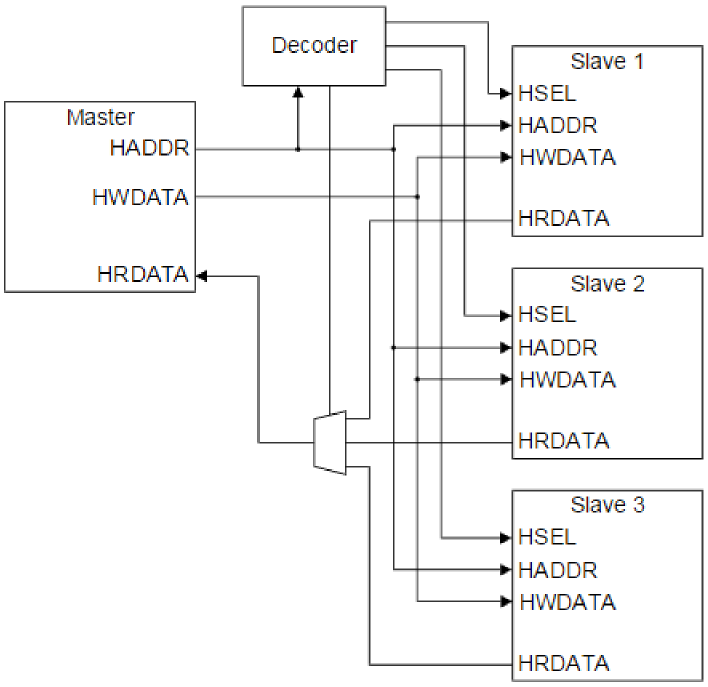


Figure 3.2: AHB-Lite components

3.2.1 Introduction to AHB-Lite slave

3.2.1.1 About the protocol

AMBA AHB [1]-Lite addresses the requirements of high-performance synthesizable designs. It is a bus interface that supports a single bus master and provides high-bandwidth operation.

AHB-Lite implements the features required for high-performance, high clock frequency systems including:

- burst transfers
- single-clock edge operation
- non-tristate implementation
- wide data bus configurations, 64, 128, 256, 512, and 1024 bits.

The most common AHB-Lite slaves are internal memory devices, external memory interfaces, and high bandwidth peripherals. Although low-bandwidth peripherals can be included as AHB-Lite slaves, for system performance reasons they typically reside on the AMBA Advanced Peripheral Bus (APB). Bridging between this higher level of bus and APB is done using a AHB-Lite slave, known as an APB bridge.

Figure 3.3 shows a single master AHB-Lite system design with one AHB-Lite master and three AHB-Lite slaves. The bus interconnect logic consists of one address decoder and a slave-to-master multiplexor. The decoder monitors the address from the master so that the appropriate slave is selected and the multiplexor routes the corresponding slave output data back to the master.

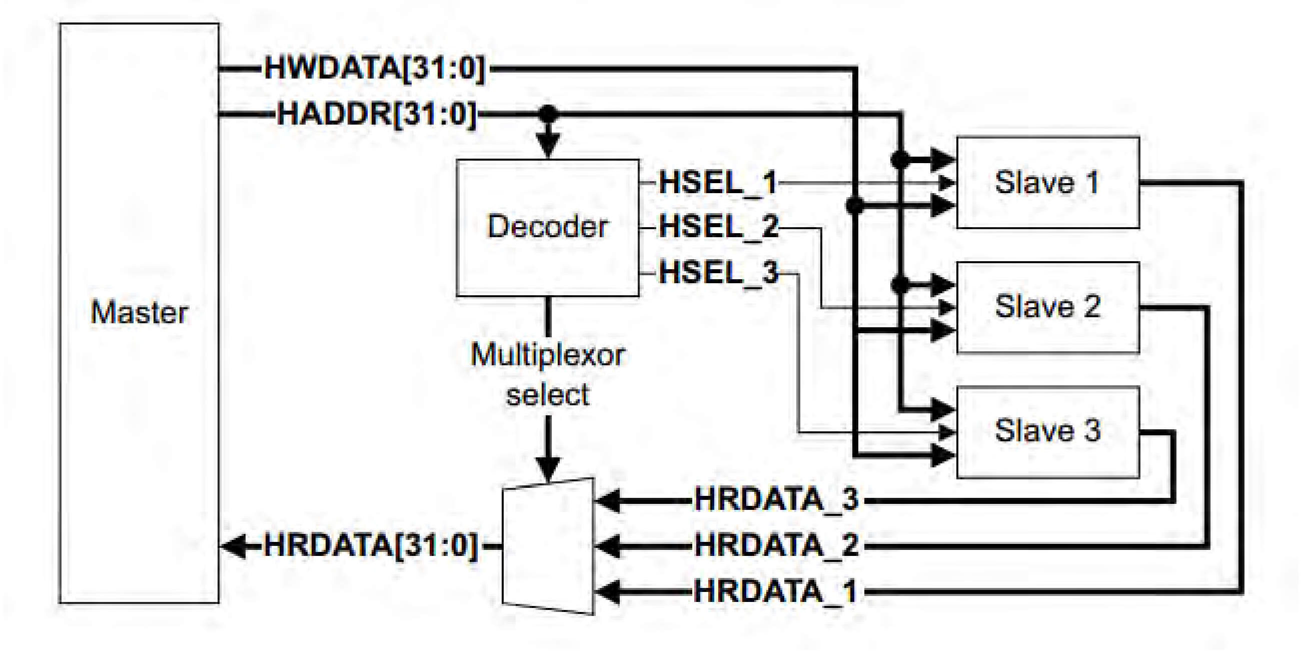


Figure 3.3: AHB-Lite block diagram

3.2.1.2 Master

An AHB-Lite master provides address and control information to initiate read and write operations. Figure 3.4 shows an AHB-Lite master interface.

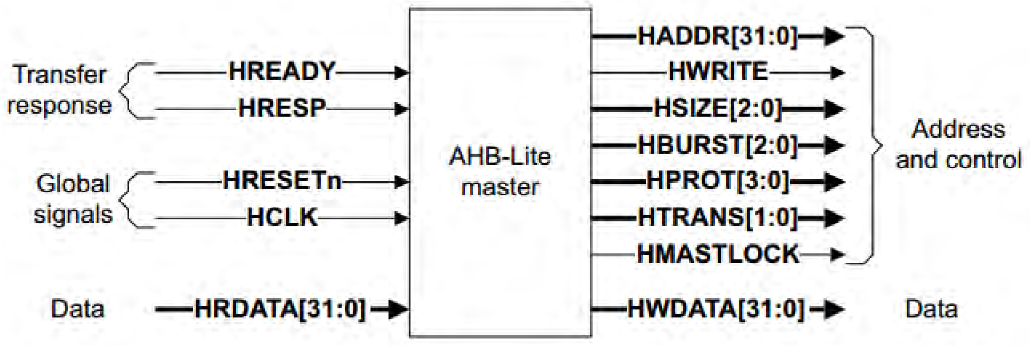


Figure 3.4: Master interface

3.2.1.3 Slave

An AHB-Lite slave responds to transfers initiated by masters in the system. The slave uses the HSELx select signal from the decoder to control when it responds to a bus transfer. The slave signals back to the master:

- the success
- failure
- or waiting of the data transfer.

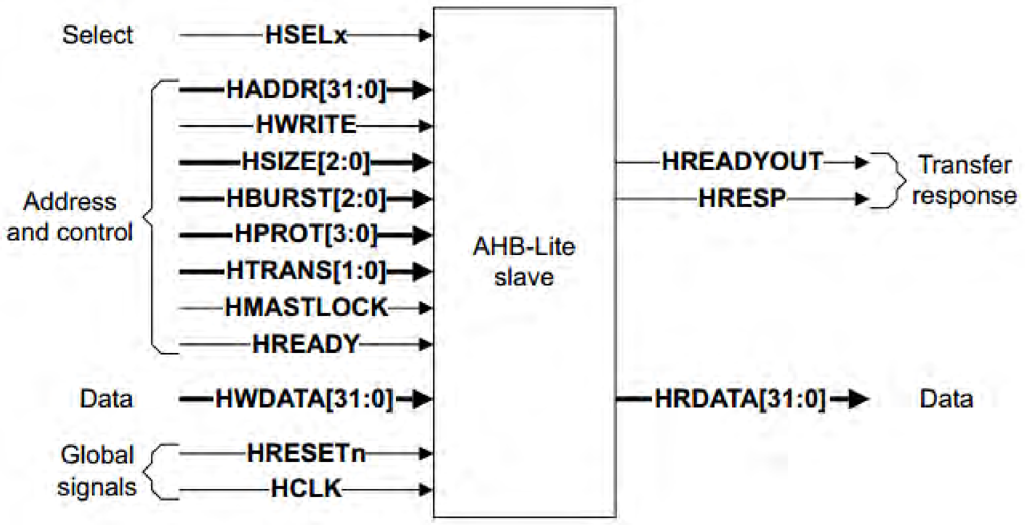


Figure 3.5: Slave interface

3.2.1.4 Decoder

This component decodes the address of each transfer and provides a select signal for the slave that is involved in the transfer. It also provides a control signal to the multiplexor. A single centralized decoder is required in all AHB-Lite implementations that use two or more slaves.

3.2.1.5 Mutiplexor

A slave-to-master multiplexor is required to multiplex the read data bus and response signals from the slaves to the master. The decoder provides control for the multiplexor. A single centralized multiplexor is required in all AHB-Lite implementations that use two or more slaves.

3.2.1.6 Operation

The master starts a transfer by driving the address and control signals. These signals provide information about the address, direction, width of the transfer, and indicate if the transfer forms part of a burst. Transfers can be:

- single
- incrementing bursts that do not wrap at address boundaries
- wrapping bursts that wrap at particular address boundaries.

The write data bus moves data from the master to a slave, and the read data bus moves data from a slave to the master. Every transfer consists of:

Address phase	one address and control cycle
Data phase	one or more cycles for the data

A slave cannot request that the address phase is extended and therefore all slaves must be capable of sampling the address during this time. However, a slave can request that the master extends the data phase by using HREADY. This signal, when LOW, causes wait states to be inserted into the transfer and enables the slave to have extra time to provide or sample data. The slave uses HRESP to indicate the success or failure of a transfer

3.2.2 Signal Description

In this section will analyze the signals that this circuit use and where their source as well as their destination is (master, multiplexor, decoder). As you will notice, they are the modified signals are used at the circuit according to some corrections were made for reducing the die size area. This was necessary because the funds for the fabrication process were limited. Finally, regardless of the modifications, as you can see at next chapters the final modified protocol is fully functional according to its bus modified width.

3.2.2.1 General Signals

Name	**Source**	**Destination**
clk	Clock source	The bus clock times all bus transfers. All signal timings are related to the rising edge of CLK
rst	Reset	The bus reset signal is active LOW and resets the system and the bus. This is the only active LOW AHB-Lite signal

Table 3.1: Global Signals

3.2.2.2 Master Signals

Name	**Destination**	**Description**
HADDR[7:0]	Slave and decoder	The 8-bit system address bus
HBURST[2:0]	Slave	The burst type indicates if the transfer is a single transfer or forms part of a burst. Fixed length bursts of 4, 8, and 16 beats are supported. The burst can be incrementing or wrapping. Incrementing bursts of undefined length are also supported.
HMASTLOCK (clk)	Slave	When HIGH, this signal indicates that the current transfer is part of a locked sequence. It has the same timing as the address and control signals.
HPROT[3:0]	Slave	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wants to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access, and if the transfer is a privileged mode access or user mode access. For masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable. The specific signal is not used at this design.
HSIZE[2:0]	Slave	Indicates the size of the transfer, that is typically byte, halfword, or word. The protocol allows for larger transfer sizes up to a maximum of 1024 bits.
HTRANS[1:0]	Slave	Indicates the transfer type of the current transfer. This can be: • IDLE • BUSY • NONSEQUENTIAL • SEQUENTIAL
HWDATA[7:0]	Slave	The write data bus transfers data from the master to the slaves during write operations. A minimum data bus width of 8 bits is recommended. However, this can be extended to enable higher bandwidth operation.
HWRITE	Slave	Indicates the transfer direction. When HIGH this signal indicates a write transfer and when LOW a read transfer. It has the same timing as the address signals, however, it must remain constant throughout a burst transfer

Table 3.2: Master Signals

3.2.2.3 Slave Signals

Name	Destination	Description
HRDATA[7:0]	Multiplexor	During read operations, the read data bus transfers data from the selected slave to the multiplexor. The multiplexor then transfers the data to the master. A minimum data bus width of 32 bits is recommended. However, this can be extended to enable higher bandwidth operation. In this design a bus width of 8 bits is used.
HREADY	Multiplexor	When **HIGH**, the HREADY signal indicates that a transfer has finished on the bus. This signal can be driven **LOW** to extend a transfer.
HMASTCLOCK (clk)	Slave	When HIGH, this signal indicates that the current transfer is part of a locked sequence. It has the same timing as the address and control signals.
HRESP	Multiplexor	The transfer response, after passing through the multiplexor, provides the master with additional information on the status of a transfer. When LOW, the **HRESP** signal indicates that the transfer status is OKAY. When HIGH, the **HRESP** signal indicates that the transfer status is ERROR. In this design were made some modifications considering the HRESP signal. See the subsection 4.2.1 at page 35 for more information.

Table 3.3: Slave Signals

3.2.2.4 Decoder Signals

Name	Destination	Description
HSELx	Slave	Each AHB-Lite slave has its own slave select signal HSELx and this signal indicates that the current transfer is intended for the selected slave. When the slave is initially selected, it must also monitor the status of HREADY to ensure that the previous bus transfer has completed, before it responds to the current transfer. The HSELx signal is a combinational decode of the address bus.[1]

Table 3.4: Decoder Signals

[1]The letter x used in HSELx must be changed to a unique identifier for each AHB-Lite slave in a system

3.2.2.5 Multiplexor Signals

Name	**Destination**	**Description**
HRDATA[7:0]	Master	Read data bus, selected by the decoder.[2]
HREADY	Master and Slave	When HIGH, the **HREADY** signal indicates to the master and all slaves, that the previous transfer is complete.
HRESP	Master	Transfer response, selected by the decoder.[2]

Table 3.5: Multiplexor Signals

3.2.3 AHB-Lite Slave Transfers

In this section is described the basic transfers of AHB-Lite protocol.

3.2.3.1 Basic Transfers

An AHB-Lite transfer consists of two phases:

Address	Lasts for a single **HCLK** cycle unless its extended by the previous bus transfer.
Data	That might require several **HCLK** cycles. Use the **HREADY** signal to control the number of clock cycles required to complete the transfer.

HWRITE controls the direction of data transfer to or from the master. Therefore, when:

- **HWRITE** is HIGH, it indicates a write transfer and the master broadcasts data on the write data bus, **HWDATA[7:0]**
- **HWRITE** is LOW, a read transfer is performed and the slave must generate the data on the read data bus, **HRDATA[7:0]**

The simplest transfer is one with no wait states, so the transfer consists of one address cycle and one data cycle. Figure 3.6 shows a simple read transfer and Figure 3.7 shows a simple write transfer.

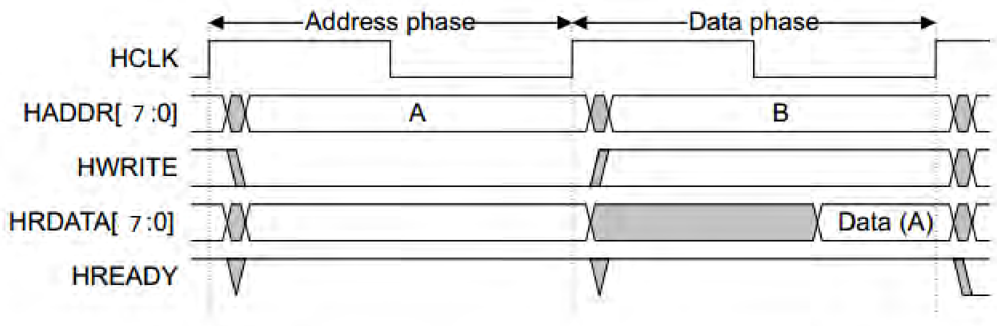


Figure 3.6: Read transfer

[2]Because the HRDATA[7:0] and HRESP signals pass through the multiplexor and retain the same signal naming, the full signal description for these two signals are provided in 3.3 on page 24.

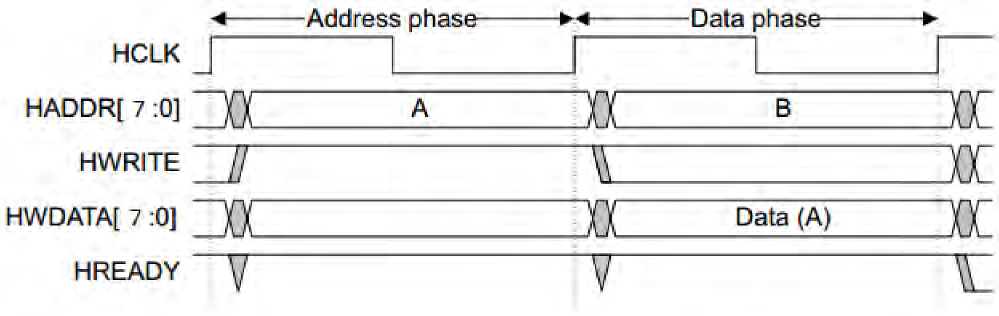


Figure 3.7: Write transfer

In a simple transfer with no wait states:

1. The master drives the address and control signals onto the bus after the rising edge of **clk**.
2. The slave then samples the address and control information on the next rising edge of **clk**.
3. After the slave has sampled the address and control it can start to drive the appropriate **HREADY** response. This response is sampled by the master on the third rising edge of **clk**.

This simple example demonstrates how the address and data phases of the transfer occur during different clock cycles. The address phase of any transfer occurs during the data phase of the previous transfer. This overlapping of address and data is fundamental to the pipelined nature of the bus and enables high performance operation while still providing adequate time for a slave to provide the response to a transfer.

A slave can insert wait states into any transfer to enable additional time for completion. Figure 3.8 shows a read transfer with two wait states and Figure 3.9 shows a write transfer with one wait state.

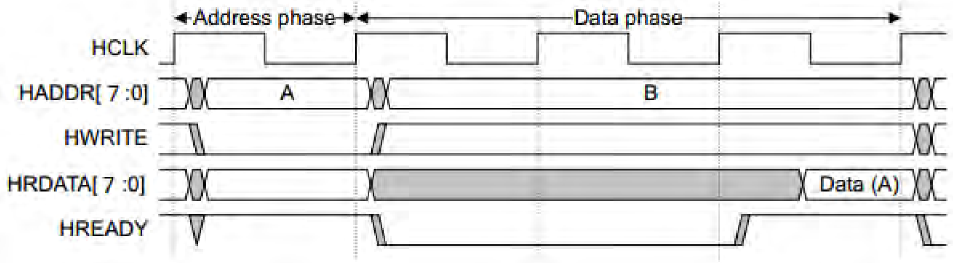


Figure 3.8: Read transfer with two wait states

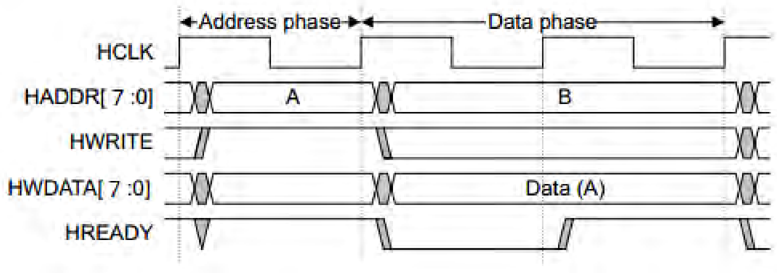


Figure 3.9: Write transfer with one wait state

Note: For write operations the master holds the data stable throughout the extended cycles. For read transfers the slave does not have to provide valid data until the transfer is about to complete.

When a transfer is extended in this way it has the side-effect of extending the address phase of the next transfer. Figure 3.10 shows three transfers to unrelated addresses, A, B, and C with an extended address phase for address C.

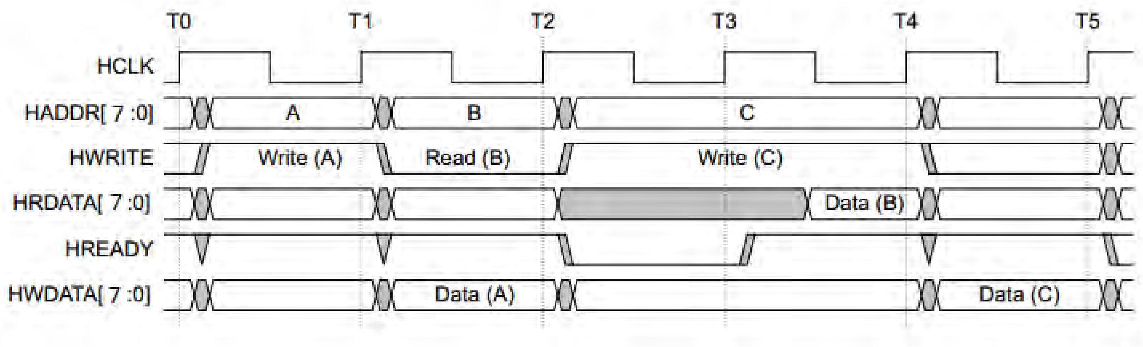


Figure 3.10: Multiple transfers

In Figure 3.10:

- the transfers to addresses A and C are zero wait state
- the transfer to address B is one wait state
- extending the data phase of the transfer to address B has the effect of extending the address phase of the transfer to address C

3.2.3.2 Transfer Types

Transfers can be classified into one of four types, as controlled by HTRANS[1:0]. Table 3.6 lists these.

HTRANS[1:0]	**Type**	**Description**
b00	IDLE	Indicates that no data transfer is required. A master uses an IDLE transfer when it does not want to perform a data transfer. It is recommended that the master terminates a locked transfer with an IDLE transfer. Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer must be ignored by the slave.
b01	BUSY	The BUSY transfer type enables masters to insert idle cycles in the middle of a burst. This transfer type indicates that the master is continuing with a burst but the next transfer cannot take place immediately. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst. Only undefined length bursts can have a BUSY transfer as the last cycle of a burst. Slaves must always provide a zero wait state OKAY response to BUSY transfers and the transfer must be ignored by the slave.
b10	NONSEQ	Indicates a single transfer or the first transfer of a burst. The address and control signals are unrelated to the previous transfer. Single transfers on the bus are treated as bursts of length one and therefore the transfer type is NON-SEQUENTIAL.
b11	SEQ	The remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer. The control information is identical to the previous transfer. The address is equal to the address of the previous transfer plus the transfer size, in bytes, with the transfer size being signaled by the HSIZE[2:0] signals. In the case of a wrapping burst the address of the transfer wraps at the address boundary.

Table 3.6: Transfer type encoding

3.2.3.3 Transfer Size

HSIZE[2:0] indicates the size of a data transfer. Table 3.7 lists the possible transfer sizes.

HSIZE[2:0]	**Size (bits)**	**Description**
b000	8	Byte
b001	16	Halfword
b010	32	Word
b011	64	Doubleword
b100	128	4-word line
b101	256	8-word line
b110	512	-
b111	1024	-

Table 3.7: Transfer size encoding

Note: The transfer size set by **HSIZE** must be less than or equal to the width of the data bus. For example, with a 32-bit data bus, HSIZE must only use the values b000, b001, or b010.

The **HSIZE** signals have exactly the same timing as the address bus. However, they must remain constant throughout a burst transfer.

3.2.3.4 Burst Operation

Bursts of 4, 8, and 16-beats, undefined length bursts, and single transfers are defined in this protocol. It supports incrementing and wrapping bursts:

- Incrementing bursts access sequential locations and the address of each transfer in the burst is an increment of the previous address.
- Wrapping bursts wrap when they cross an address boundary. The address boundary is calculated as the product of the number of beats in a burst and the size of the transfer. The number of beats are controlled by **HBURST** and the transfer size is controlled by textbfHSIZE. For example, a four-beat wrapping burst of word (4-byte) accesses wraps at 16-byte boundaries. Therefore, if the start address of the transfer is 0x34, then it consists of four transfers to addresses 0x34, 0x38, 0x3C, and 0x30.

HBURST[2:0] controls the burst type. Table 3.8 lists the possible burst types.

HBURST[2:0]	**Type**	**Description**
b000	SINGLE	Single burst
b001	INCR	Incrementing burst of undefined length
b010	WRAP4	4-beat wrapping burst
b011	INCR4	4-beat incrementing burst
b100	WRAP8	8-beat wrapping burst
b101	INCR8	8-beat incrementing burst
b110	WRAP16	16-beat wrapping burst
b111	INCR16	16-beat incrementing burst

Table 3.8: Burst signal encoding

Masters must not attempt to start an incrementing burst that crosses a 1KB address boundary. Masters can perform single transfers using either:

- SINGLE burst
- undefined length burst that has a burst of length one.

Note: The burst size indicates the number of beats in the burst and not the number of bytes transferred. Calculate the total amount of data transferred in a burst by multiplying the number of beats by the amount of data in each beat, as indicated by **HSIZE[2:0]**.

All transfers in a burst must be aligned to the address boundary equal to the size of the transfer. For example, you must align word transfers to word address boundaries (**HADDR[1:0]** = b00), and halfword transfers to halfword address boundaries (**HADDR[0]** = 0). The address for IDLE transfers must also be aligned, otherwise during simulation it is likely that bus monitors could report spurious warnings.

<u>Burst termination after a BUSY transfer</u>

After a burst has started, the master uses BUSY transfers if it requires more time before continuing with the next transfer in the burst. During an undefined length burst, INCR, the master might insert BUSY transfers and then decide that no more data transfers are required. Under these circumstances, it is acceptable for the master to then perform a NONSEQ or IDLE transfer that then effectively terminates the undefined length burst. The protocol does not permit a master to end a burst with a BUSY transfer for fixed length bursts of type:

- incrementing INCR4, INCR8, and INCR16
- or wrapping WRAP4, WRAP8, and WRAP16.

These fixed length burst types must terminate with a SEQ transfer. The master is not permitted to perform a BUSY transfer immediately after a SINGLE burst. SINGLE bursts must be followed by an IDLE transfer or a NONSEQ transfer.

Early burst termination
Bursts can be terminated by either:

- Slave error response
- Multi-layer interconnect termination

Slave error response
If a slave provides an ERROR response then the master can cancel the remaining transfers in the burst. However, this is not a strict requirement and it is also acceptable for the master to continue the remaining transfers in the burst.

If the master does not complete that burst then there is no requirement for it to rebuild the burst when it next accesses that slave. For example, if a master only completes three beats of an eight-beat burst then it does not have to complete the remaining five transfers when it next accesses that slave.

Multi-layer interconnect termination
Although masters are not permitted to terminate a burst request early, slaves must be designed to work correctly if the burst is not completed. When a multi-layer interconnect component is used in a multi-master system then it can terminate a burst so that another master can gain access to the slave. The slave must terminate the burst from the original master and then respond appropriately to the new master if this occurs.

3.2.3.5 Waited Transfers

Slaves use **HREADY** to insert wait states if they require more time to provide or sample the data. During a waited transfer, the master is restricted to what changes it can make to the transfer type and address. These restrictions are described in the following sections:

- Transfer type changes during wait states
- Address changes during wait states

Transfer type changes during wait states When the slave is requesting wait states, the master must not change the transfer type, except as described in:

- IDLE transfer
- BUSY transfer, fixed length burst
- BUSY transfer, undefined length burst

IDLE transfer During a waited transfer, the master is permitted to change the transfer type from IDLE to NONSEQ. When the **HTRANS** transfer type changes to NONSEQ the master must keep **HTRANS** constant, until **HREADY** is HIGH.

BUSY transfer, fixed length burst During a waited transfer for a fixed length burst, the master is permitted to change the transfer type from BUSY to SEQ. When the **HTRANS** transfer type changes to SEQ the master must keep **HTRANS** constant, until **HREADY** is HIGH.

BUSY transfer, undefined length burst During a waited transfer for an undefined length burst, INCR, the master is permitted to change from BUSY to any other transfer type, when **HREADY** is LOW. The burst continues if a SEQ transfer is performed but terminates if an IDLE or NONSEQ transfer is performed.

Address changes during wait states When the slave is requesting wait states, the master can only change the address once, except as described in:

- During an IDLE transfer
- After an ERROR response

During an IDLE transfer During a waited transfer, the master is permitted to change the address for IDLE transfers. When the **HTRANS** transfer type changes to NONSEQ the master must keep the address constant, until **HREADY** is HIGH.

After an ERROR response During a waited transfer, if the slave responds with an ERROR response then the master is permitted to change the address when **HREADY** is LOW.

3.2.4 Slave Response Signaling

This section describes the slave response signaling.

3.2.4.1 Slave Transfer Responses

After a master has started a transfer, the slave controls how the transfer progresses. A master cannot cancel a transfer after it has commenced. A slave must provide a response that indicates the status of the transfer when it is accessed. The transfer status is provided by the **HRESP** signal. Table 3.9 lists the **HRESP** states.

HRESP	**Response**	**Description**
0	OKAY	The transfer has either completed successfully or additional cycles are required for the slave to complete the request. The **HREADY** signal indicates whether the transfer is pending or complete.
1	ERROR	An error has occurred during the transfer. The error condition must be signaled to the master so that it is aware the transfer has been unsuccessful. A two-cycle response is required for an error condition with **HREADY** being asserted in the second cycle.

Table 3.9: HRESP signal

Table 3.9 shows that the complete transfer response is a combination of the **HRESP** and **HREADY** signals. Table 3.10 lists the complete transfer response based on the status of these two signals

This means the slave can complete the transfer in the following three ways:

	HREADY	
HRESP	**0**	**1**
0	Transfer pending	Successful transfer completed
1	ERROR response, first cycle	ERROR response, second cycle

Table 3.10: Transfer response

- immediately complete the transfer
- insert one or more wait states to enable time to complete the transfer
- signal an error to indicate that the transfer has failed.

Transfer done A successful completed transfer is signaled when **HREADY** is HIGH and **HRESP** is OKAY.

Transfer pending A typical slave uses **HREADY** to insert the appropriate number of wait states into the data phase of the transfer. The transfer then completes with **HREADY** HIGH and an OKAY response to indicate the successful completion of the transfer. When a slave inserts a number of wait states prior to completing the response, it must drive **HRESP** to OKAY.

Note: In general, every slave must have a predetermined maximum number of wait states that it inserts before it backs off the bus. This enables you to calculate the latency for accessing the bus. It is recommended that slaves do not insert more than 16 wait states, to prevent any single access locking the bus for a large number of clock cycles. However, this recommendation is not applicable to some devices, for example, a serial boot ROM. This type of device is usually only accessed during system startup and the impact on system performance is negligible if greater than 16 wait states are used.

ERROR response

A slave uses the ERROR response to indicate some form of error condition with the associated transfer. Usually this denotes a protection error such as an attempt to write to a read-only memory location.

Although an OKAY response can be given in a single cycle, the ERROR response requires two cycles. To start the ERROR response, the slave drives **HRESP** HIGH to indicate ERROR while driving **HREADY** LOW to extend the transfer for one extra cycle. In the next cycle **HREADY** is driven HIGH to end the transfer and **HRESP** remains driven HIGH to indicate ERROR.

The two-cycle response is required because of the pipelined nature of the bus. By the time a slave starts to issue an ERROR response then the address for the following transfer has already been broadcast onto the bus. The two-cycle response provides sufficient time for the master to cancel this next access and drive **HTRANS[1:0]** to IDLE before the start of the next transfer.

If the slave requires more than two cycles to provide the ERROR response then additional wait states can be inserted at the start of the transfer. During this time **HREADY** is LOW and the response must be set to OKAY.

If a slave provides an ERROR response then the master can cancel the remaining transfers in the burst. However, this is not a strict requirement and it is also acceptable for the master to continue the remaining transfers in the burst.

3.2.5 Data Buses

This section describes the AHB-Lite data buses. It contains the following sections:

3.2.5.1 Data Buses

Separate read and write data buses are required to implement an AHB-Lite system without using tristate drivers. Although the recommended minimum data bus width is specified as 32 bits, you can change this (8 bits at the specific design). The data buses are described in:

- HWDATA
- HRDATA
- Endianness

HWDATA

The master drives the write data bus during write transfers. If the transfer is extended then the master must hold the data valid until the transfer completes, as indicated by **HREADY** HIGH. For transfers that are narrower than the width of the bus, for example a 4-bit transfer on a 8-bit bus, the master only has to drive the appropriate byte lanes. The slave selects the write data from the correct byte lanes.

HRDATA

The appropriate slave drives the read data bus during read transfers. If the slave extends the read transfer by holding **HREADY** LOW then the slave only has to provide valid data in the final cycle of the transfer, as indicated by **HREADY** HIGH. For transfers that are narrower than the width of the bus, the slave only requires to provide valid data on the active byte lanes. The master selects the data from the correct byte lanes. A slave only has to provide valid data when a transfer completes with an OKAY response. ERROR responses do not require valid read data.

Endianness

It is essential that all modules are of the same endianness and also that any data routing or bridges are of the same endianness for the system to function correctly. Dynamic endianness is not supported, because in the majority of embedded systems, this leads to a significant redundant silicon overhead. It is recommended that only modules designed for use in a wide variety of applications are made bi-endian, with either a configuration pin or internal control bit to select the endianness. For more application-specific blocks, fixing the endianness to either little-endian or big-endian results in a smaller, lower power, higher performance interface.

3.2.6 Clock and Reset

This section describes the AHB-Lite data buses. It contains the following sections:

3.2.6.1 Clock

Each AHB-Lite component uses a single clock signal, clk. All input signals are sampled on the rising edge of clk. All output signal changes must occur after the rising edge of clk.

3.2.6.2 Reset

The reset signal, **rst**, is the only active LOW signal in the AHB-Lite protocol and is the primary reset for all bus elements. The reset can be asserted asynchronously, but is asserted synchronously after the rising edge of **clk**. During reset all masters must ensure the address and control signals are at valid levels and that **HTRANS[1:0]** indicates IDLE. During reset all slaves must ensure that **HREADYOUT** is HIGH.

Chapter 4

Front End Flow

4.1 ARM Specifications, Circuit Description/Architecture

At the previous chapters (chapter 3 and chapter 2) we describe extensively the ARM AHB-lite slave specifications and the functionality as well as the values of each signal separately. The final result for this thesis would be a single ARM AHB-lite slave SoC circuit that would operate as a single module. After the circuit fabrication, the SoC will have the ability to read and write data to its memory module and export the output of this operation according to the specifications from ARM [7].

4.2 RTL Coding

As mentioned before, the initial RTL code was originated from opencores website. At first some modifications needed to be done which are described at section 4.2.1 because the circuit was at beta form and after a quick simulation it did not applied the ARM AHB-lite slave specifications. Finally, in this thesis, because the main purpose was the fabrication of the circuit, it was necessary to create a top level module that would include the pads of the signal pins of the circuit. A figure is shown below (figure 4.1):

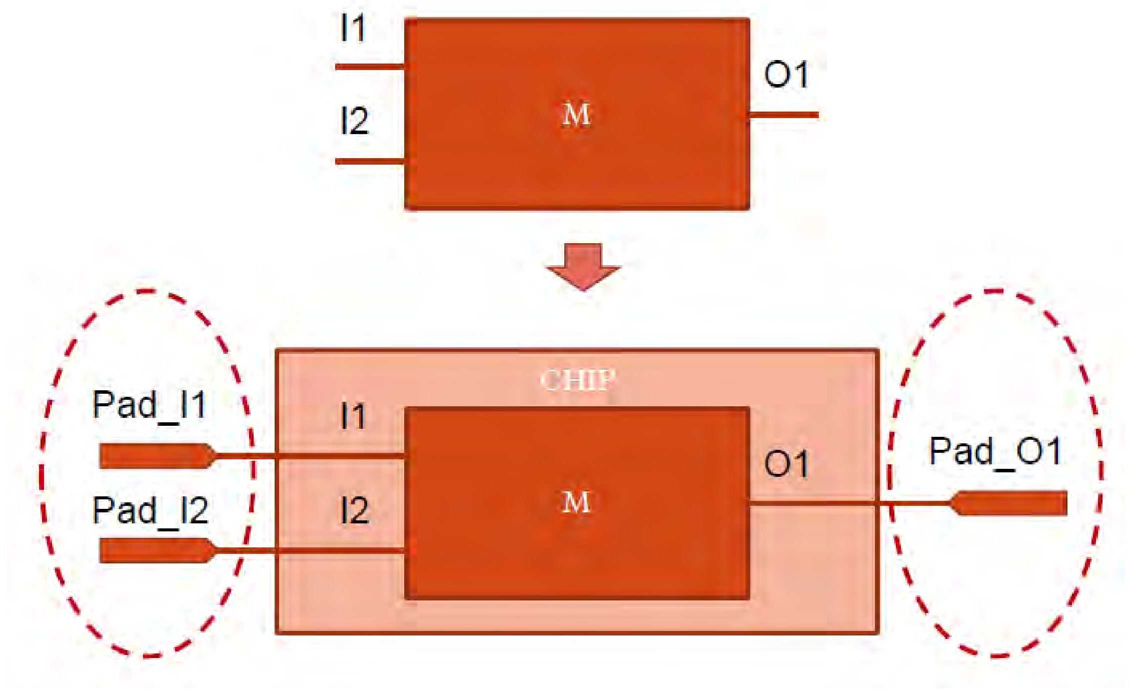


Figure 4.1: From pins to pads

4.2.1 Modifications

In this section we will describe the modifications needed to be done to the initial circuit as well as the modifications made to implement the error response of the slave according to ARM specification files. Concerning the modifications from the initial circuit the following modifications needed to be done:

- Read and write functionality. After simulating the initial RTL code we realized that the pipeline functionality concerning the write transfer was not implemented in the right way. So, first of all we added the pipeline write functionality (one cycle would refer to the address and the next one to the data itself of the data that we want to write to the memory) to the circuit.

- Implement the memory as a register file and modifications in order to have a capacity of 256 stored data of 256 different addresses of 8 bits of width each.

Furthermore, concerning the error response of the slave, we modified the initial implementation that referred to an error response of a single address of hexadecimal value of 'FF' to a wider range of values. The implementation of error signal for this thesis after a discussion and a final agreement with my advisor was as follows: The error signal sould contain three different cases:

- **Error response with five wait cycles:** HRESP would be high after five clock cycles of an error address read from slave.
- **Error response with three wait cycles:** HRESP would be high after three clock cycles of an error address read from slave.
- **Error response with no wait cycles:** HRESP would be high immediate after an error address read from slave.

To achieve this we had to divide the HADDR signal (which refers to the address that a master wants to access to read or write) with the binary number 100 in order to take the modulo result of this operation. The modulo result operation of HADDR signal with 100 would give us the wait cycles we had to wait in order to observe an error response from slave. So, we had the three different cases mentioned above and the modulo_result signal values which was a signal to help us to distinguish these cases more clear. These cases are (referred to binary values):

- **five wait cycles:** modulo_result equals to 011
- **three wait cycles:** modulo_result equals to 010
- **no wait cycles:** modulo_result equals to 001

During the wait cycles the next address that should a master send have to remain steady till the HREADY signal becomes high.

Finally, a modify to the signal width needed to be done in order to achieve the desired minimum area of the circuit without any consequence to the functionality of the circuit. These modifications included the downsize from 32 to 8 bits to the signals referred to the address and the data of the circuit. These signals were: HADDR, HWDATAand HRDATA. The only consequence, was that it would operate to a smaller range of address and data transfers.

4.2.2 Code Fragments

In this section we will quote some code fragments and will explain what these code fragments implement in our design.

Listing 4.1: `ahb_slave_mem.v`

```
module ahb_slave_mem (clk, reset, WR, RD, ADDR_WR, ADDR_RD, DIN, DOUT);

    parameter                        MEM_WORDS = 256;
    input                          clk;
    input                          reset;
    input                          WR;
    input                          RD;
    input   [7:0]                  ADDR_WR;
    input   [7:0]                  ADDR_RD;
    input   [7:0]                  DIN;
    output  [7:0]                  DOUT;

    reg  [7:0]                     DOUT;
    reg  [7:0]                     Mem[MEM_WORDS-1:0];

    always @(posedge clk)
      if (WR)
        Mem[ADDR_WR] <= DIN;

    always @(posedge clk or posedge reset)
      if (reset)
        DOUT <= {8{1'b0}};
      else if (RD)
        DOUT <= Mem[ADDR_RD];

endmodule
```

As we can see at the code segment 4.1, the memory of our circuit is modeled as a register file with 256 memory addresses and width of 8 each to store data of width 8 as well.

The segment 4.2 describes the read transfer in our design. As we mentioned at section 3.2.3.1, during the read operation of our circuit, we have the address phase and the data phase. That means that at the first clock cycle of the transfer the slave gets the address that the master wants to read data and at the immediate next cycle the slave returns the data to the master of the given address.

Listing 4.2: `ahb_slave_ram_read`

```
//ARM protocol read segment implementation
    always @(posedge clk or posedge reset)
      if (reset)
        begin
         RD_pre_d <= 1'b0;
         ADDR_RD <= {8{1'b0}};
        end
      else if (HREADY | HRESP)
        begin
         RD_pre_d <= RD_pre;
         ADDR_RD <= ADDR_RD_pre;
        end
```

The segment 4.3 describes the write transfer in our design. As we mentioned at section 3.2.3.1, during the write operation of our circuit, we have the address phase and the data phase. That means that at the first clock cycle of the transfer the slave gets the address that the master wants to write data and at the immediate next cycle the slave gets the data from the master of the given address in order to store the data to the address given in its memory.

Listing 4.3: `ahb_slave_ram_write`

```
//ARM protocol write segment implementation
   always @(posedge clk or posedge reset)
      if (reset)
        begin
         WR_pre_d <= 1'b0;
         ADDR_WR <= {8{1'b0}};
        end
      else if (HREADY | HRESP)
        begin
         WR_pre_d <= WR_pre;
         ADDR_WR <= ADDR_WR_pre;
        end
```

The segment 4.4 describes the write transfer in our design. As we mentioned at section 3.2.4.1, when we have an address send from master to slave that will cause an error signal response we have two signals that involved in this situation, **HRESP** and **HREADY**. The implementation below, is our implementation that was a result of discussions of the way we would model the logic behind the error signal response in our circuit. First of all we set three counters for each case of error response, which described at section 4.2.1. After the cycle that the address read from slave, master can send another address, **but**, that address will be read when **HREADY** becomes 1, i.e. after the second cycle of **HRESP** is 1. The two cycles of **HRESP** at 1 and **HREADY** being 0 at the first cycle of **HRESP** is 1 and 1 at the second cycle that **HRESP** is 1 is the error response of our AHB-lite slave. So, with the verilog code segment below we counter after how many clock cycles this interaction of **HRESP** and **HREADY** has to begin. If the counter does not reach to the desired number it continues to counter till the number of wait cycles that it has to wait. When it reaches the desired number of wait cycles, first we reset the counters to zero, afterwards the **HRESP** signal becomes 1 and the error signal that is a flag for the **HREADY** signal remains 0. At the next cycle that the **HRESP** signal is high we set error signal to 1, so the **HREADY** becomes high again.

Listing 4.4: `ahb_slave_ram.v`

```
always @(posedge clk or posedge reset)
      if (reset)
      begin
        HRESP = 1'b0;
        counter = 3'b000;
        counter1 = 1'b0;
        counter5 = 1'b0;
        counter3 = 1'b0;
        error = 1'b0;
      end
           else if ((|HTRANS) & (modulo_result_error == HRESP_addr_n_w_c) & (HRESP == 1'b0))
                begin            //counter for the wait cycles of each possibly
                   counter1 = 1'b1;  //error address transaction
                   counter1 = 1'b0;
                   counter = 3'b000;
                   HRESP = 1'b1;
                   error = 1'b0;
              end

            else if ((|HTRANS) & (modulo_result_error == HRESP_addr_w_c_3) & (HRESP == 1'b0))
             begin
               counter3 = 1'b1;
               if((counter == 3'b011) & (counter3 == 1'b1))
                 begin
                   counter3 = 1'b0;
                   counter = 3'b000;
                   HRESP = 1'b1;
                   error = 1'b0;
                   end
                 else if ((counter3 == 1'b1) & (HRESP == 1'b0))
                   begin
                   error = 1'b0;
                   HRESP = 1'b0;
                   counter = counter + 1'b1;
                   end
             end

             else if ((|HTRANS) & (modulo_result_error == HRESP_addr_w_c_5) & (HRESP == 1'b0))
             begin
                 counter5 = 1'b1;
                 if ((counter == 3'b101) & (counter5 == 1'b1))
                   begin
                   counter5 = 1'b0;
                   counter = 3'b000;
                   HRESP = 1'b1;
                   error = 1'b0;
                   end
                  else if ((counter5 == 1'b1) & (HRESP == 1'b0))
                   begin
                   error = 1'b0;
                   HRESP = 1'b0;
                   counter = counter + 1'b1;
                   end
             end

      else if ((HREADY) & (HRESP == 1'b1))
       begin
         HRESP = 1'b0;
         error = 1'b1;
       end

      else
       error = 1'b1;
```

For further understanding of how the signals operate and how the interact at the segment 4.5 we present the signal interaction. The code for stall signal is for manual wait cycles in our design. We first define the address response constants for each case of error response in our circuit. Afterwards we define the constants for the transfer type of slave. Moving on, we define the **STALL** signal for controlling the **HREADY** signal (as we can see from the **HREADY** assignment). Finally, we define the signals and how they interact with other signals. For example, **WR_pre** which is the indicator of a write transfer, is high when we have **SEQ** or **NSEQ** transfer and **HWRITE** is high, which indicates that we want to write to the memory of slave.

Listing 4.5: `ahb_slave_ram_signals`

```
parameter  HRESP_addr_n_e = 2'b00;   //address for response no error
parameter  HRESP_addr_n_w_c = 2'b01; //address for response error no wait cycle
parameter  HRESP_addr_w_c_3 = 2'b10; //address for response error 3 wait cycle
parameter  HRESP_addr_w_c_5 = 2'b11; //address for response error 5 wait cycle


parameter  TRANS_IDLE = 2'b00; //IDLE
parameter  TRANS_STALL  = 2'b01; //BUSY
parameter  TRANS_NONSEQ = 2'b10; //NON_SEQ
parameter  TRANS_SEQ  = 2'b11; //SEQ

//stall code for manual wait cycle
```

```
always @(posedge clk or posedge reset)
 if (reset)
  STALL <= 1'b0;
 else
  STALL <= STALL_pre;


//important internal signals
assign  HREADY = ~error ? 1'b0 : STALL;
assign  HRDATA = HREADY & data_phase & ~HRESP ? DOUT : 'd0;
//internal signals
assign  WR_pre  = HWRITE & ((HTRANS == TRANS_NONSEQ) | (HTRANS == TRANS_SEQ));
assign  WR = WR_pre_d & HREADY & HSEL;
assign  RD_pre = (~HWRITE) & ((HTRANS == TRANS_NONSEQ) | (HTRANS == TRANS_SEQ));
assign  RD = RD_pre_d & HREADY & HSEL;
assign  ADDR_WR_pre = {8{WR_pre}} & HADDR;
assign  ADDR_RD_pre      = {8{RD_pre}} & HADDR;
//error assign for three occasions of wait cycles
assign  modulo_result = HADDR % 3'b100;
```

Moving on, a small example of the top level with pad definition is described at code fragment 4.6.

Listing 4.6: `ahb_slaveuniquify`

```
//scan pads
   ITP_V5 PAD_scan_in (.Y(Scan_in),              .PAD(PI_Scan_In));
   ITP_V5 PAD_scan_enable (.Y(Scan_enable),      .PAD(PI_Scan_enable));
   BU16P_V5 PAD_scan_out (.PAD(PO_Scan_Out) , .A(Scan_out));

   //Input pads
   //clk, reset pads North side pads

   ITCK4P_V5 PAD_clk (.Y(clk),                   .PAD(PI_clk));

   ITP_V5 PAD_reset (.Y(reset),          .PAD(PI_reset));

   ITP_V5 PAD_HSEL (.Y(HSEL),                    .PAD(PI_HSEL));


   ITP_V5 PAD_HADDR7 (.Y(HADDR[7]),              .PAD(PI_HADDR[7]));
   ITP_V5 PAD_HADDR6 (.Y(HADDR[6]),              .PAD(PI_HADDR[6]));
   ITP_V5 PAD_HADDR5 (.Y(HADDR[5]),              .PAD(PI_HADDR[5]));

   ITP_V5 PAD_HADDR4 (.Y(HADDR[4]),              .PAD(PI_HADDR[4]));

   ITP_V5 PAD_HADDR3 (.Y(HADDR[3]),              .PAD(PI_HADDR[3]));
   ITP_V5 PAD_HADDR2 (.Y(HADDR[2]),              .PAD(PI_HADDR[2]));
   ITP_V5 PAD_HADDR1 (.Y(HADDR[1]),              .PAD(PI_HADDR[1]));



//Output pads
```

```
BU16P_V5 PAD_HRESP  (.PAD(PO_HRESP),             .A(HRESP));

BU16P_V5 PAD_HREADY  (.PAD(PO_HREADY)  ,         .A(HREADY));

BU16P_V5 PAD_HRDATA0  (.PAD(PO_HRDATA[0]),      .A(HRDATA[0]));
BU16P_V5 PAD_HRDATA1  (.PAD(PO_HRDATA[1]),      .A(HRDATA[1]));
BU16P_V5 PAD_HRDATA2  (.PAD(PO_HRDATA[2]),      .A(HRDATA[2]));
BU16P_V5 PAD_HRDATA3  (.PAD(PO_HRDATA[3]),      .A(HRDATA[3]));
BU16P_V5 PAD_HRDATA4  (.PAD(PO_HRDATA[4]),      .A(HRDATA[4]));
BU16P_V5 PAD_HRDATA5  (.PAD(PO_HRDATA[5]),      .A(HRDATA[5]));
BU16P_V5 PAD_HRDATA6  (.PAD(PO_HRDATA[6]),      .A(HRDATA[6]));
BU16P_V5 PAD_HRDATA7  (.PAD(PO_HRDATA[7]),      .A(HRDATA[7]));
```

4.3 Functional Simulation

As it was mentioned in section 2.5, Questasim, Simvision and VCSMX is used to verify the correct behavior of the design **RTL Description:** At this early stage, the above EDA tools is used to check that the design from RTL code accomplishes with the specifications and product requirements. To achieve that it was necessary to create a testbench file that was able to test every possible case of any transaction that could happen to the slave according to the specifications from ARM. To create that big testbench file that included as many possible cases as studying the ARM AMBA AHB-lite slave specifications and according to the figures that explained the signals that had to trigger in each case and their respective values.

At this point it is worth mentioning that mostly for simulations was used VCSMX from Synopsys. The tool was robust and reliable and very easy to use. There is an option of changing the signals names and the values that we would like to see at the waveform and we could see more easily the mapping of the signals and the attribute that the specification file describes. At the following sections you will see some figures from these simulations and some comparisons between the EDA simulation tools. Finally, the testbenches that have been employed are the same in every phase and are explained in section 4.3.1.

4.3.1 Test Bench

A testbench is a verilog code that is used to verify the functional correctness of a HDL model. It can be seen as wrapper where the top entity of the Device Under Test (DUT) is instantiated in order to apply stimulus to the DUT and verify the corresponding outputs. As we will see to the figures that follows, the initial values of the addresses we use during the testbench written from the circuit itself at the early stage of the simulation. As all other EDA tools we use in this thesis, simulation tools also have a Tcl/Tk scripting option which allows us to save a significant amount of time when using the tool more than once. In general, a simulation cannot be correct from the first run, so a large number of rerunning the tools and simulate the results, till we reach the desired waveforms, is needed. For instance, for running VCSMX we will need to obtain the commands:

```
vcs -full64 ahb_slave.v ahb_slave_ram_fin.v ahb_slave_mem.v error_tb_arm.v -debug_all -2005
./simv -gui
```

The first command compiles the necessary source files (testbench and RTL) and produces an executable simv file that we run afterwards to visualize the waveforms. After the graphical environment starts we can navigate, add the desired signals to waveform viewer and start the simulation process. Then we can also save our session of waveforms to reuse it with the order we want and the modifications we made considering the radix for example of the signals at waveforms. These commands can merge to a single script file with the name ahb_slave_lite.tcl for example. the command to run this script and not type them down again and again is:

source ahb_slave_lite.tcl

Bellow this point some figures and the explanation of what each figure shows us will be given. Furthermore, some figures of three simulation tools results comparison will be shown.

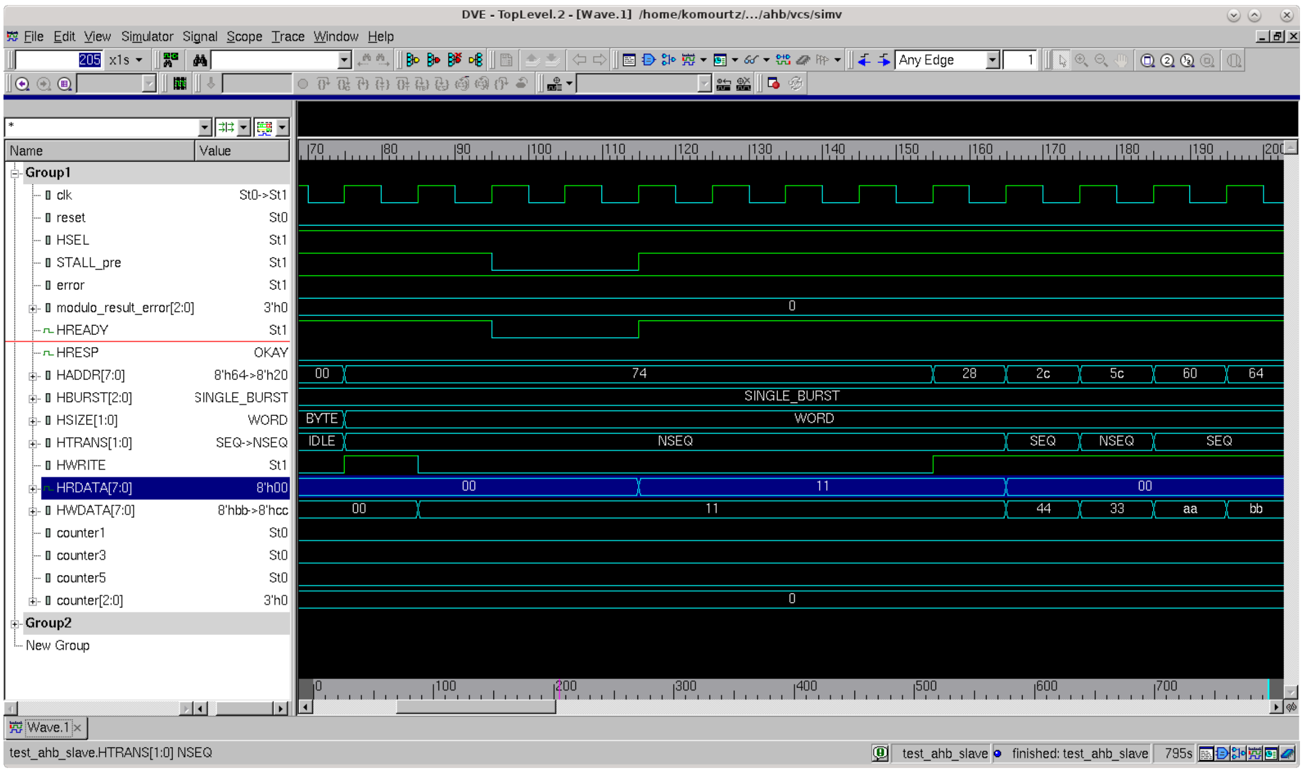
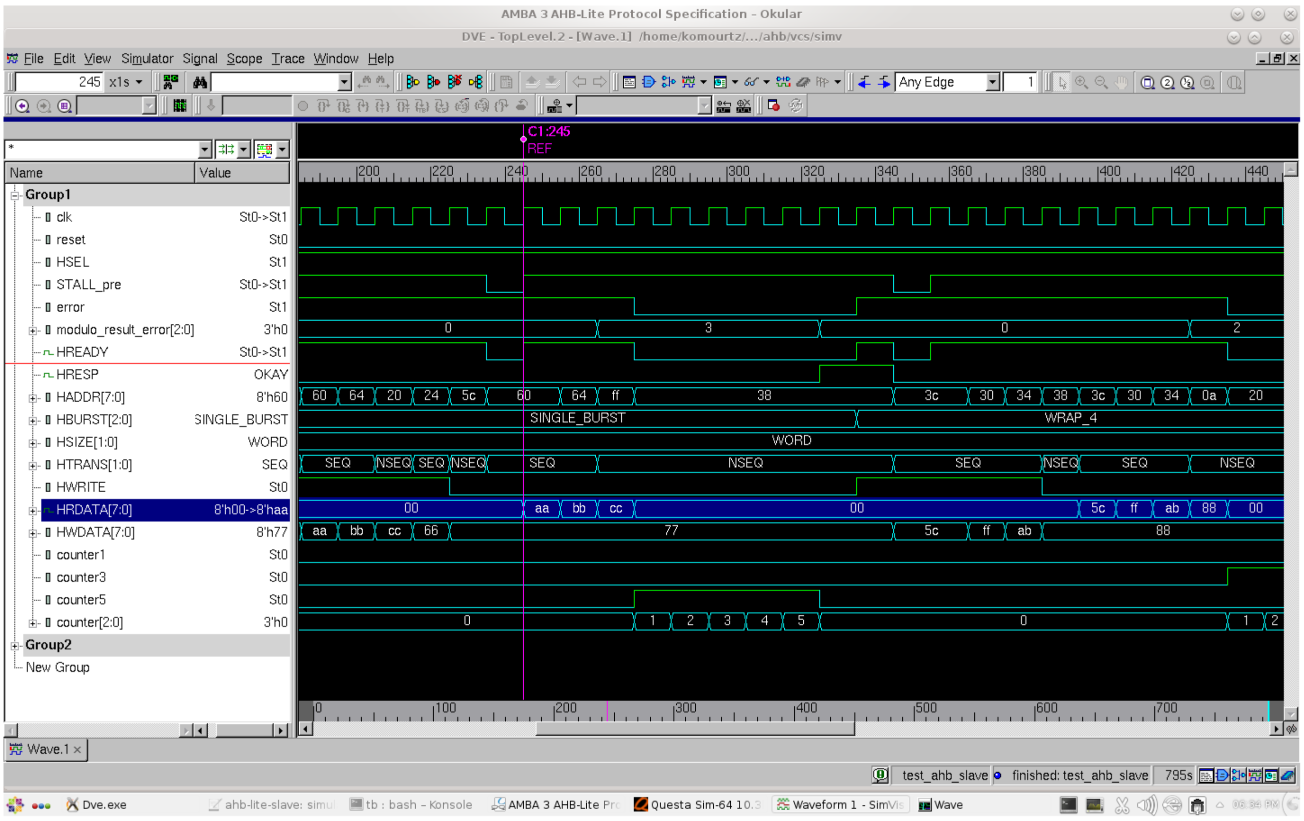


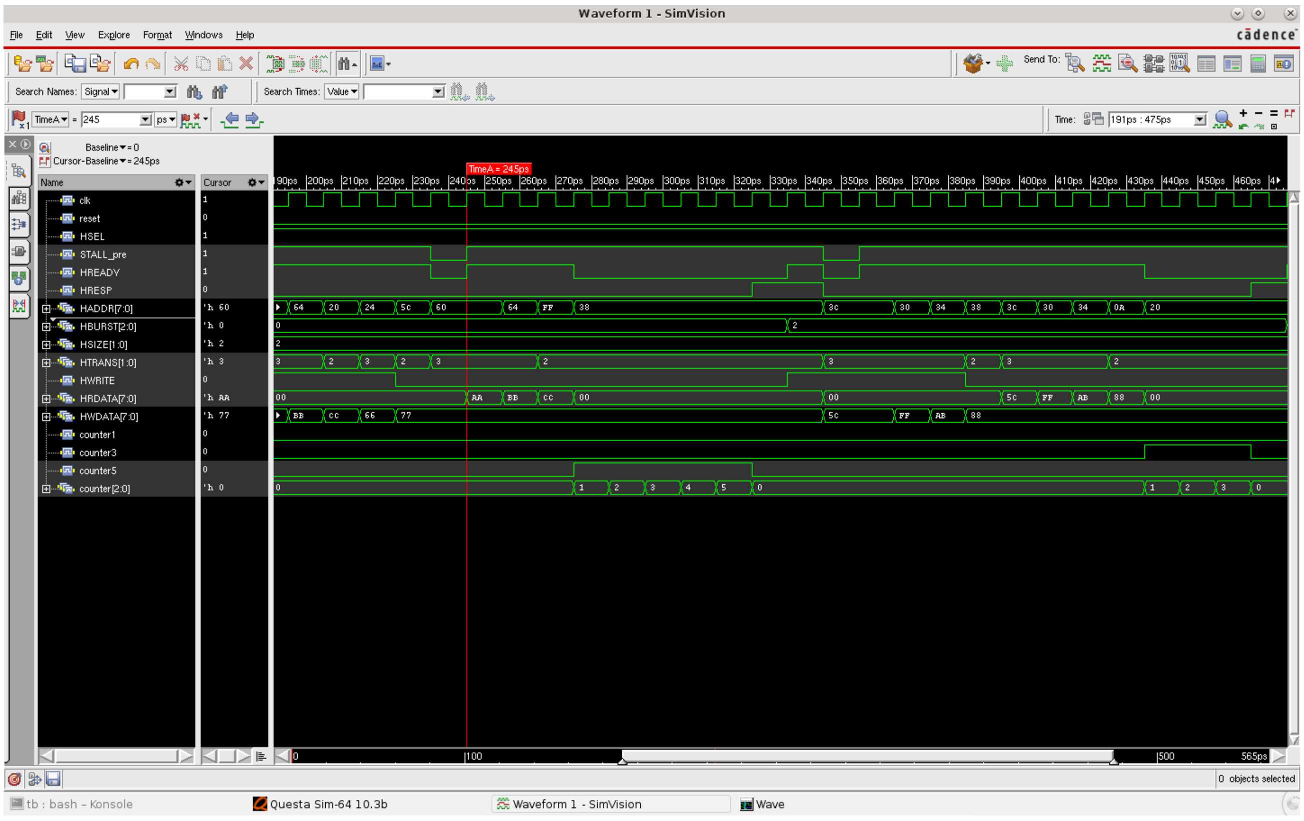
Figure 4.2: Synopsys VCSMX initial waveform

In figure 4.2 we see the initial screen of VCSMX from Synopsys, where the signal group and the waveforms are shown. In figure 4.3 we see the initial screen of VCSMX from Synopsys versus the initial waveform screen from Simvision of Cadence, where the signal group and the waveforms are shown. As we can see, the two tools have identical waveforms.

In figure 4.4 we see the initial screen of VCSMX from Synopsys versus the initial waveform screen from Mentor Graphics Questasim, where the signal group and the waveforms are shown. As we can see, the two tools have identical waveforms.. In figure 4.5 we see the initial screen of Simvision from Cadence versus the initial waveform screen from Mentor Graphics Questasim, where the signal group and the waveforms are shown. As we can see, the two tools have identical waveforms.

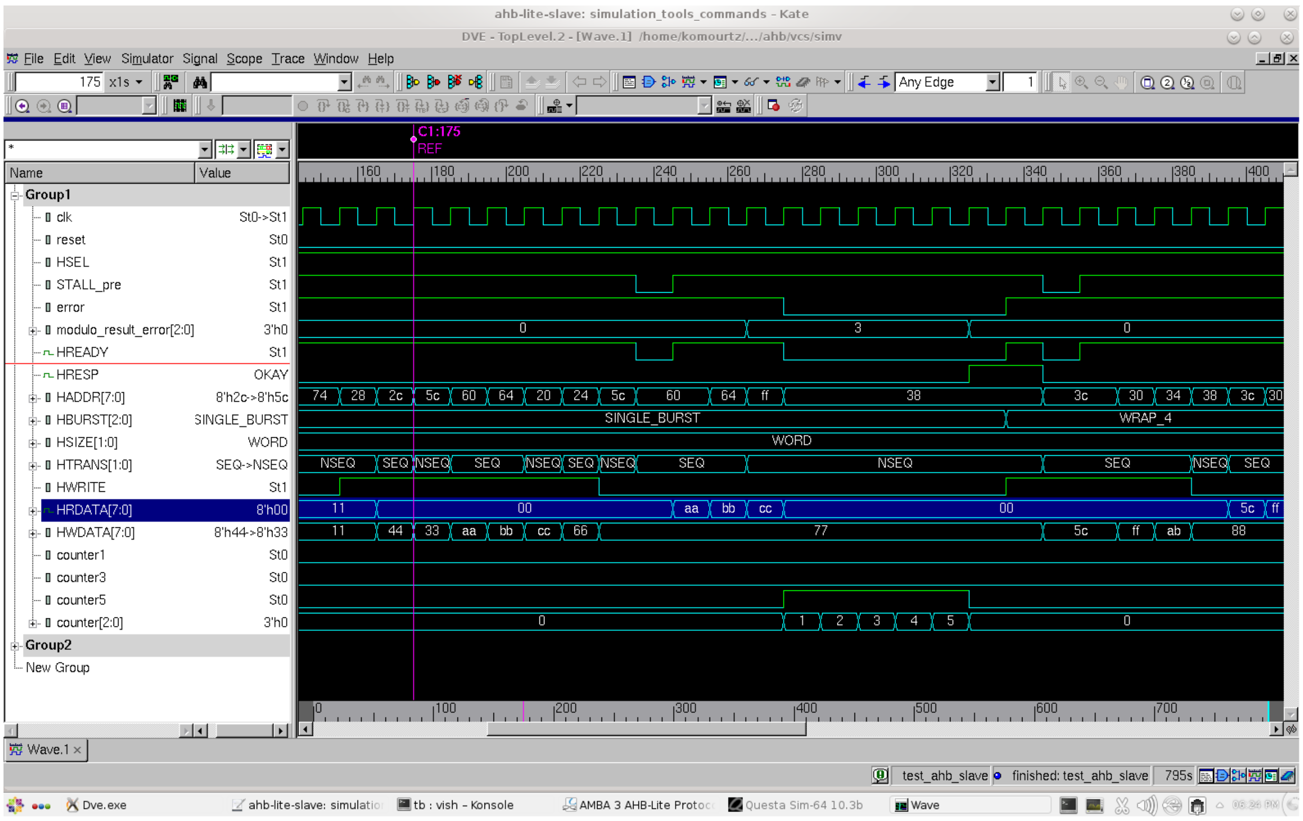


(a) Synopsys VCSMX

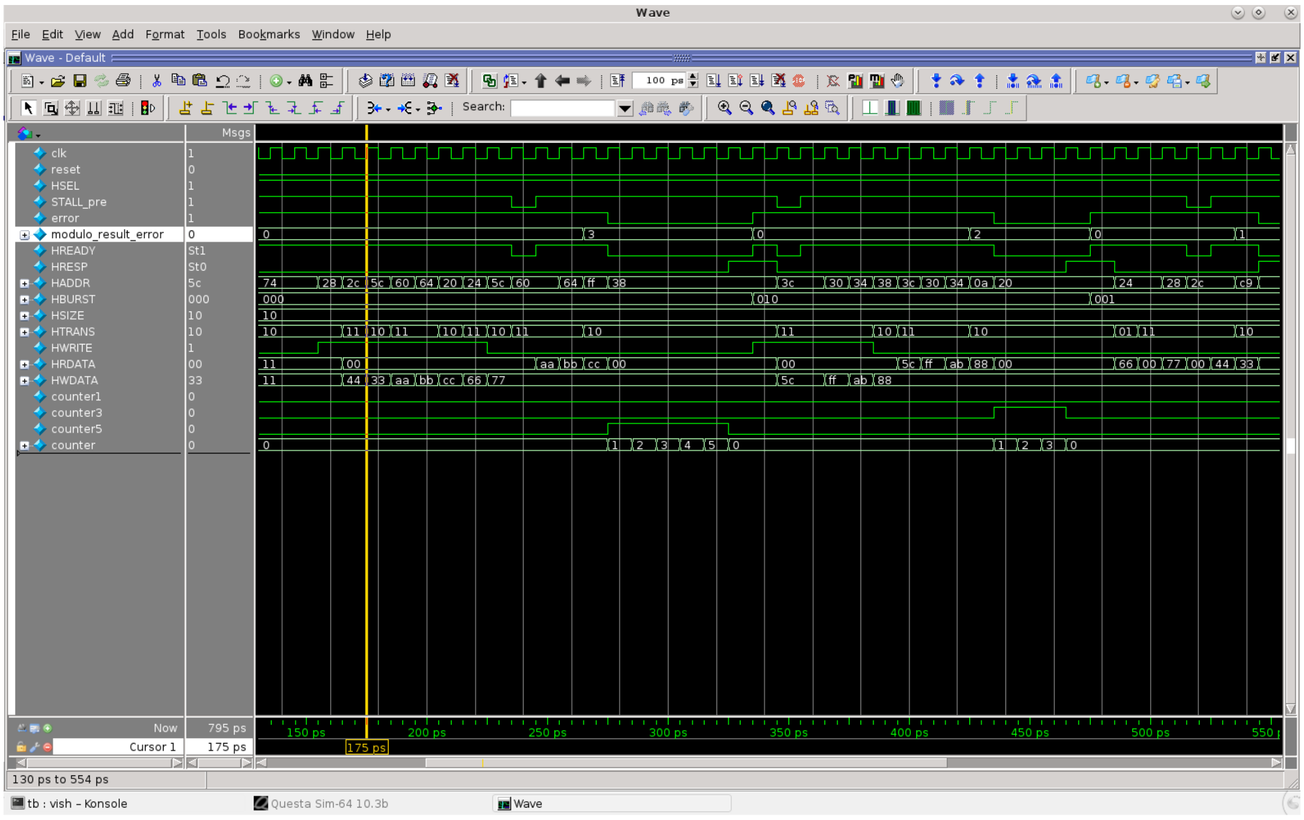


(b) Cadence Simvision

Figure 4.3: Synopsys VCSMX vs Cadence Simvision initial waveform

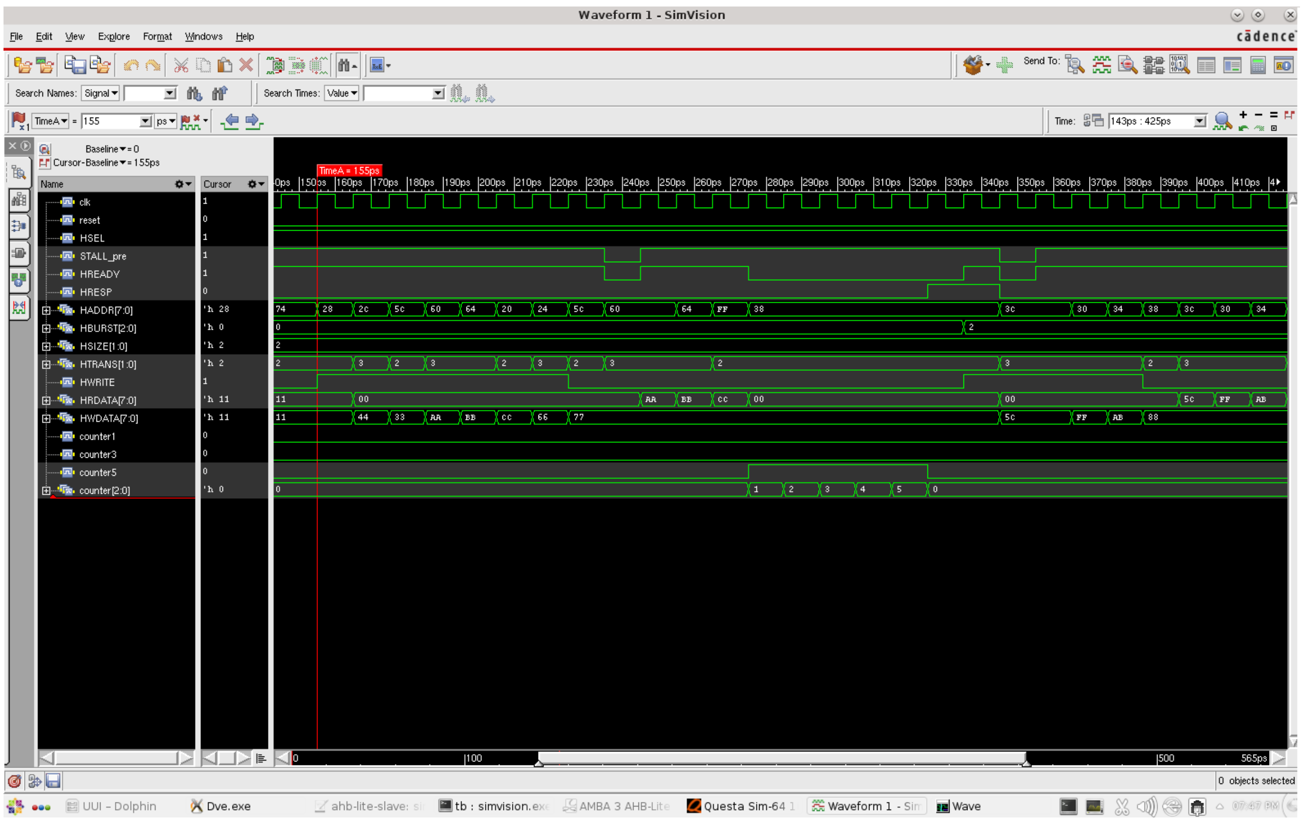


(a) Synopsys VCSMX

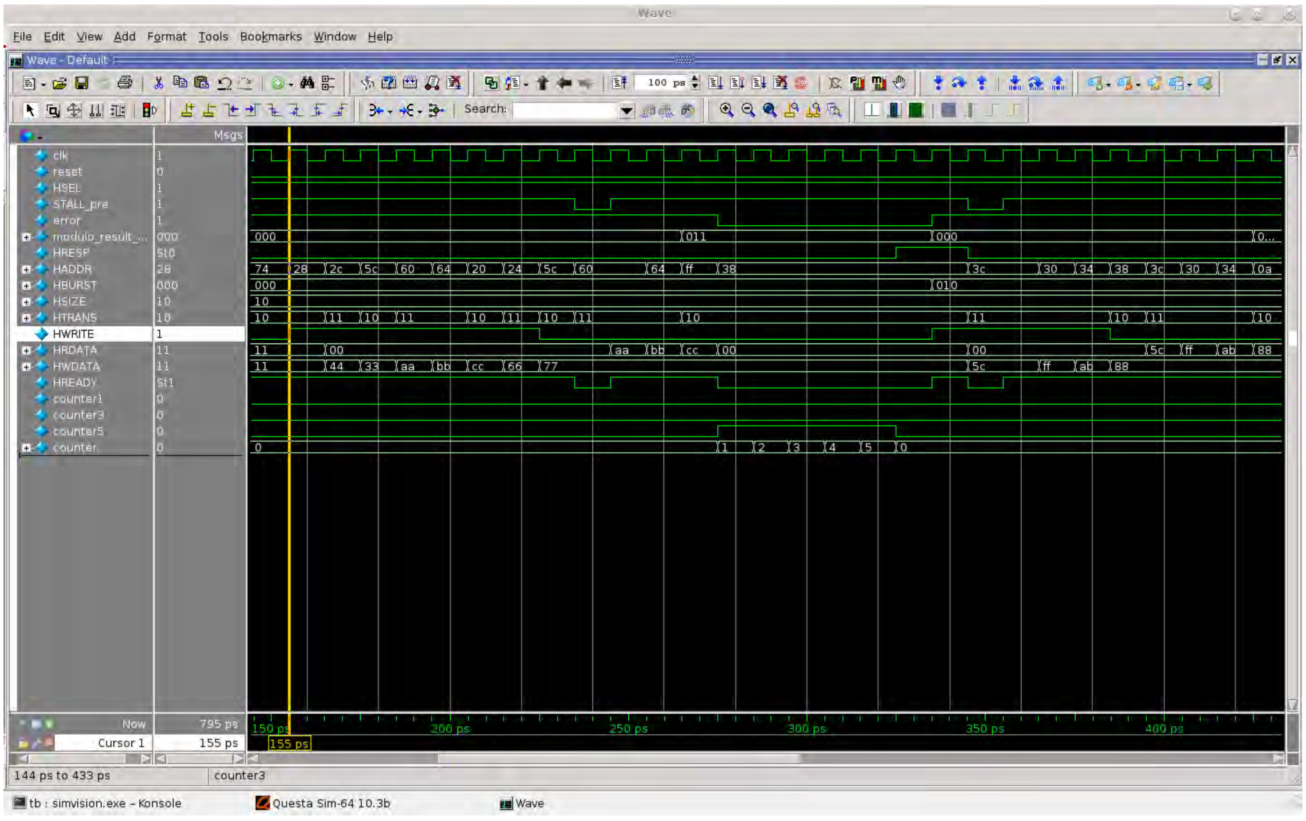


(b) Mentor Graphics Questasim

Figure 4.4: Synopsys VCSMX vs Cadence Simvision initial waveform



(a) Cadence Simvision



(b) Mentor Graphics Questasim

Figure 4.5: Cadence Simvision vs Mentor Graphics Questasim initial waveform

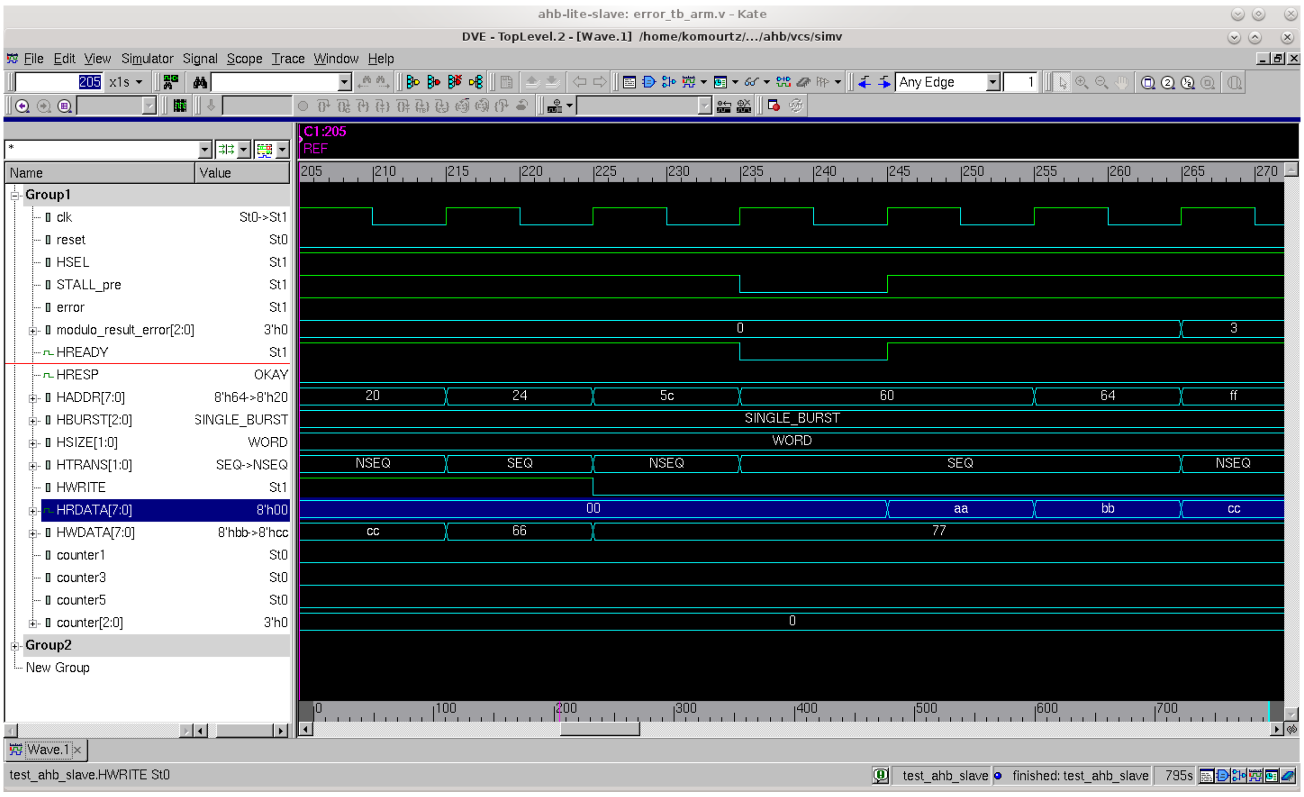


Figure 4.6: Undefined length bursts, INCR

Figure 4.6 shows:

- The first burst is a write consisting of two word transfers starting at address 0x20. These transfer addresses increment by four.

- The second burst is a read consisting of three word transfers starting at address 0x5C. These transfer addresses increment by four.

Figure 4.7 shows an error response of the address binary:11111111. It means that with the implementation of the logic behind error response (section 4.2.1), after the modulo calculation we have a result of modulo_result equals to 011. So, after five cycles (as we can see with the help of counter signal) we have the error response for this address. We have to note here that the immediate address after the address hex(ff) is stable (master has to keep it stable) for the time that HREADY is low and we can change that after HREADY is again high(after the slave reads the address).

Moving on to figure 4.8 we can see a four beat wrapping burst. Because the burst is a four-beat burst of word transfers, the address wraps at 16-byte boundaries, and the transfer to address 0x3C is followed by a transfer to address 0x30. After the transaction done, we read the address to examine that the slave successfully write the addresses and the date we gave and we can read them with no problem.

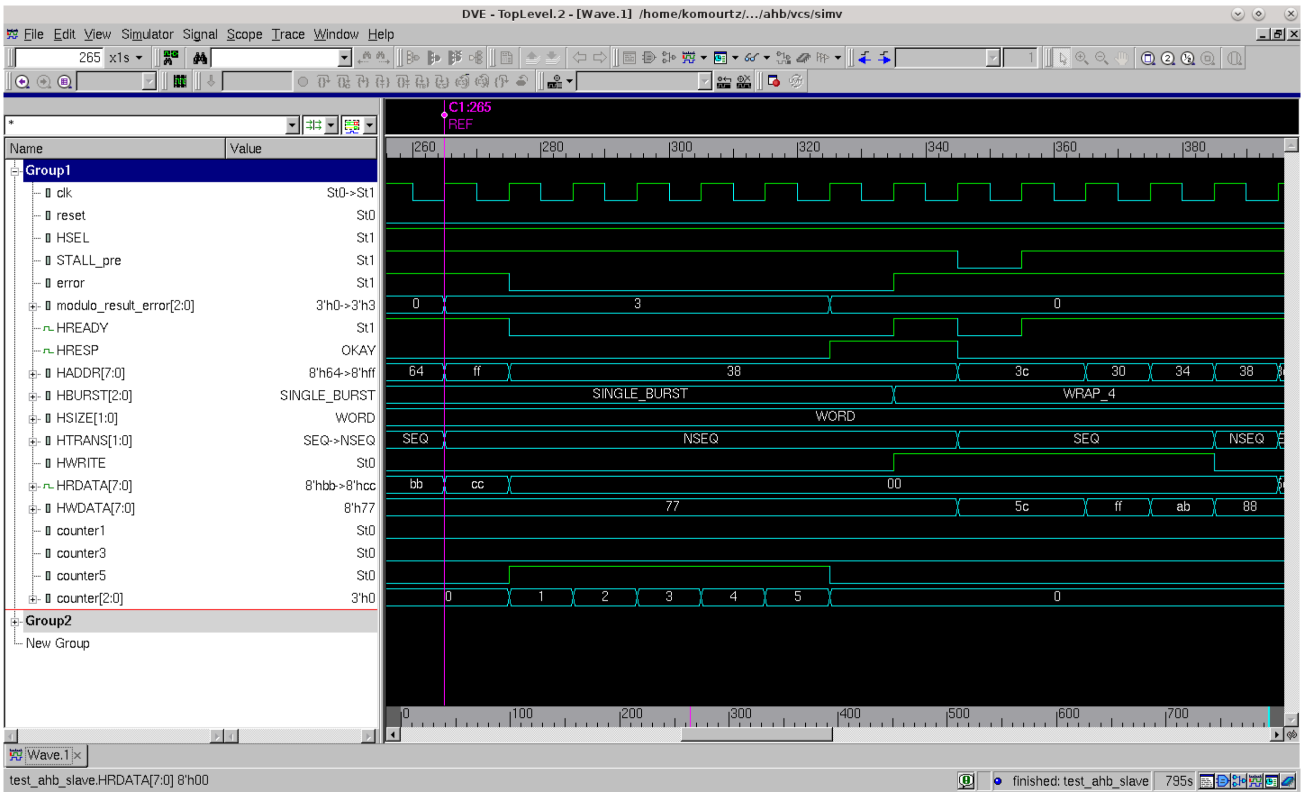


Figure 4.7: Error response with five wait cycles

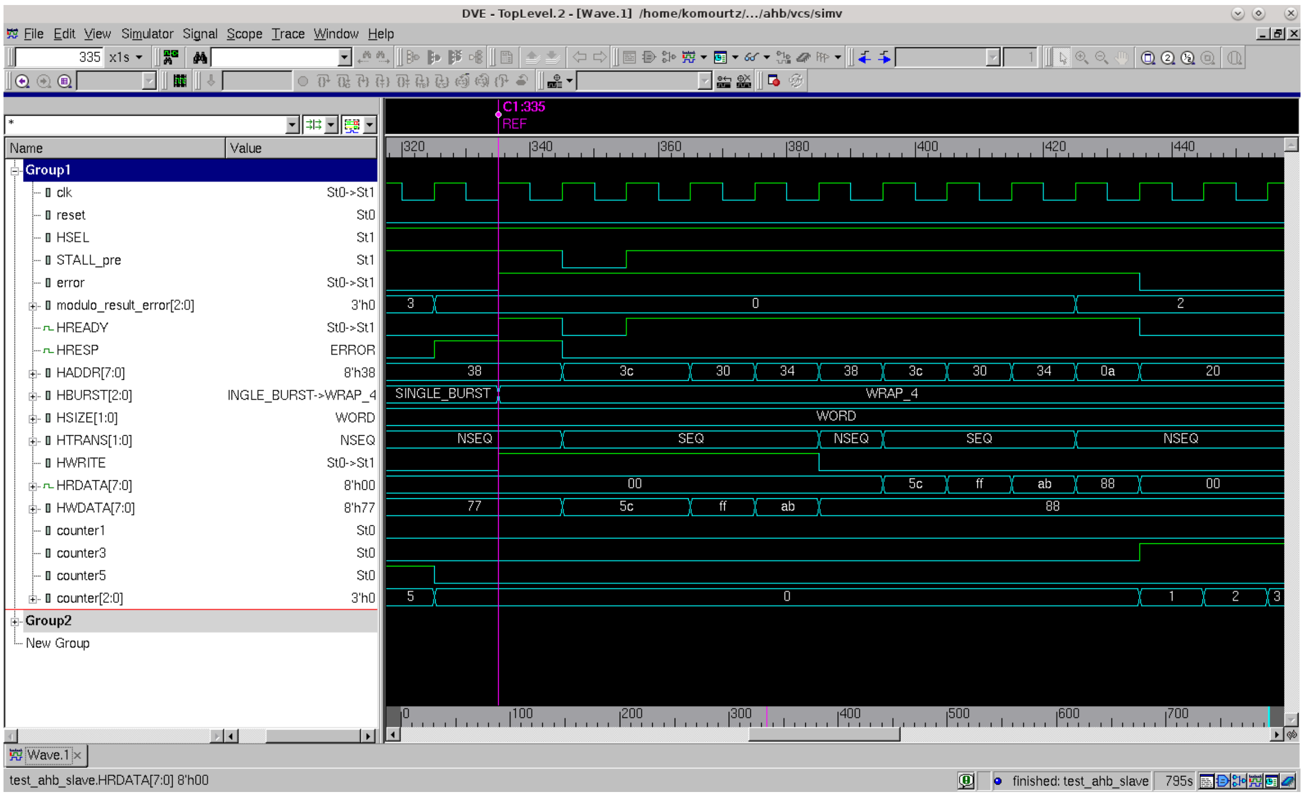


Figure 4.8: Four-beat wrapping burst

Figure 4.9 shows an error response of the address binary:00001010. It means that with the implementation of the logic behind error response (section 4.2.1), after the modulo calculation we have a result of modulo_result equals to 010. So, after three cycles (as we can see with the help of counter signal) we have the error response for this address. We have to note here that the immediate address after the address hex(ff) is stable (master has to keep it stable) for the time that HREADY is low and we can change that after HREADY is again high(after the slave reads the address).

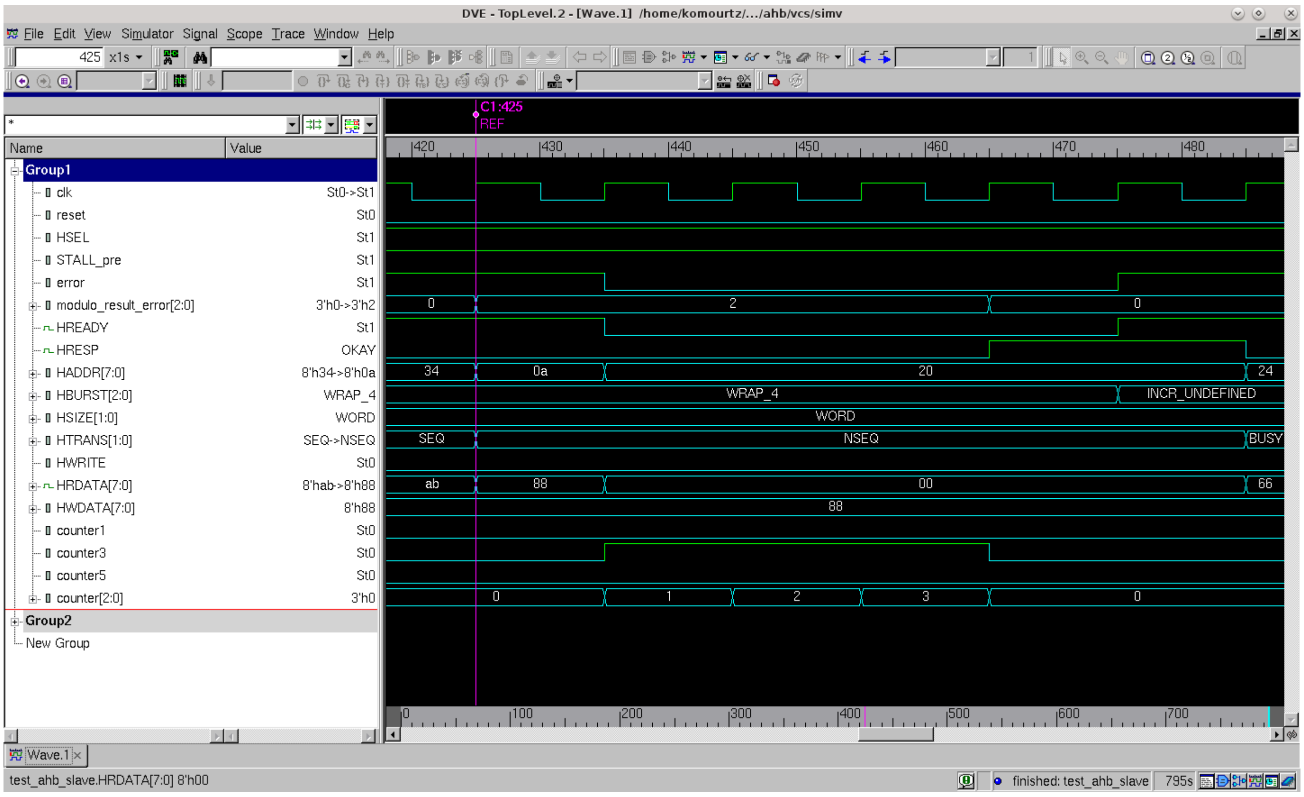


Figure 4.9: Error response with three wait cycles

Figure 4.10 shows the use of the NONSEQ, BUSY, and SEQ transfer types. More specific we have:

- **1st clock cycle:** The 4-beat read starts with a NONSEQ transfer.
- **2nd clock cycle:** The master is unable to perform the second beat and inserts a BUSY transfer to delay the start of the second beat. The slave provides the read data for the first beat.
- **3rd clock cycle:** The master is now ready to start the second beat, so a SEQ transfer is signaled. The master ignores any data that the slave provides on the read data bus.
- **4th clock cycle:** The master performs the third beat. The slave provides the read data for the second beat.
- **5th clock cycle:** The master performs the last beat. The slave is unable to complete the transfer and uses **HREADY** to insert a single wait state.
- **6th clock cycle:** The slave provides the read data for the third beat.

- **7th clock cycle:** The slave provides the read data for the last beat.

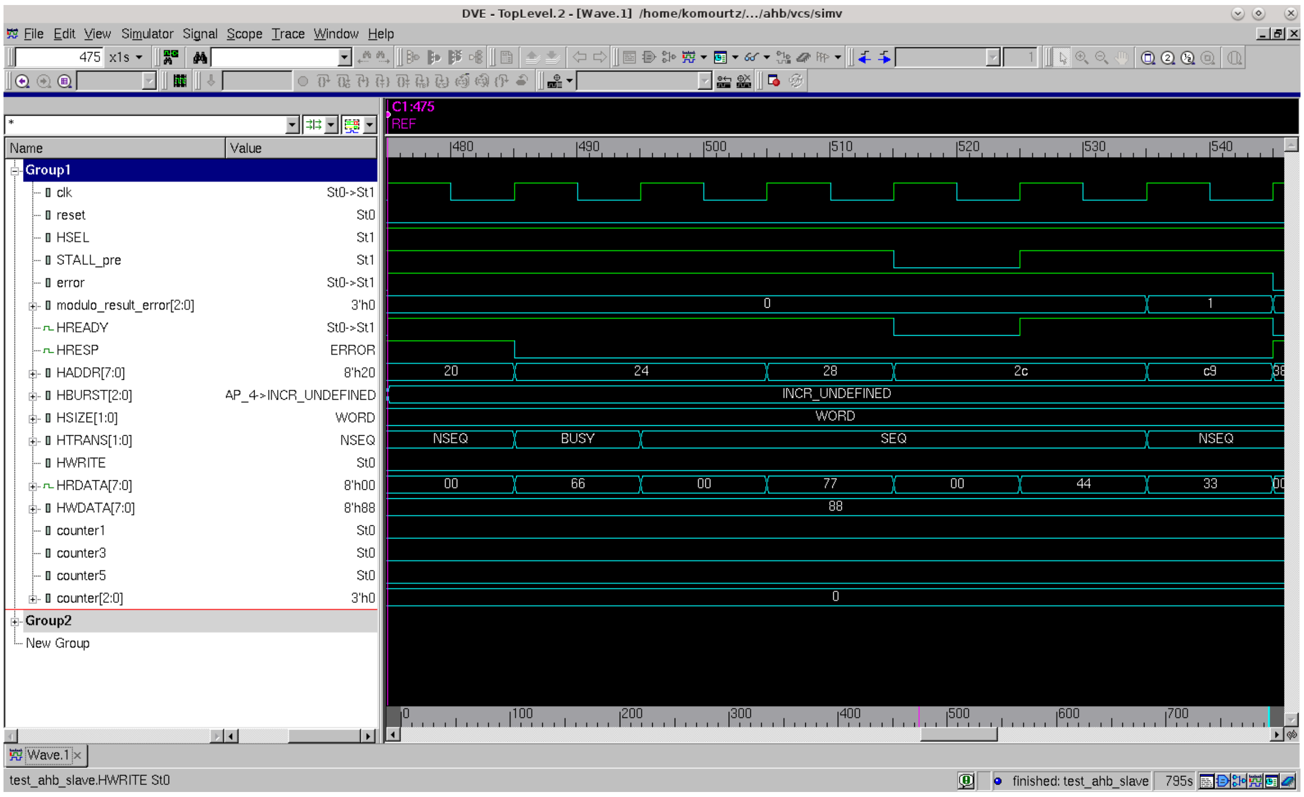
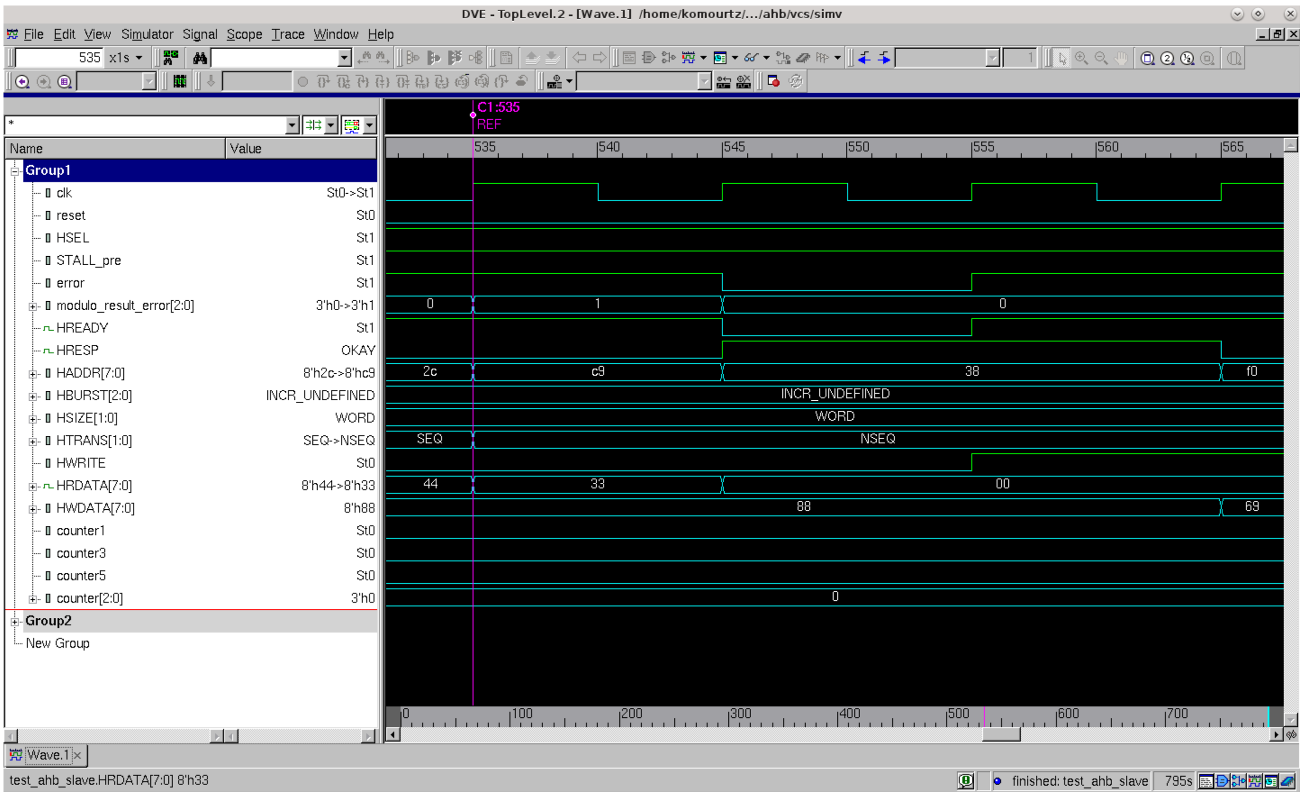


Figure 4.10: NSEQ, BUSY, and SEQ transfer types

Afterwards, in figure 4.11 shows an error response of the address binary:11001001. It means that with the implementation of the logic behind error response (section 4.2.1), after the modulo calculation we have a result of modulo_result equals to 001. So, after zero cycles (immediate error response as we can see with the help of counter signal) we have the error response for this address. We have to note here that the immediate address after the address hex(ff) is stable (master has to keep it stable) for the time that HREADY is low and we can change that after HREADY is again high(after the slave reads the address).


DVE - TopLevel.2 - [Wave.1] /home/komourtz/.../ahb/vcs/simv

Name	Value
Group1	
clk	St0->St1
reset	St0
HSEL	St1
STALL_pre	St1
error	St1
modulo_result_error[2:0]	3'h0->3'h1
HREADY	St1
HRESP	OKAY
HADDR[7:0]	8'h2c->8'hc9
HBURST[2:0]	INCR_UNDEFINED
HSIZE[1:0]	WORD
HTRANS[1:0]	SEQ->NSEQ
HWRITE	St0
HRDATA[7:0]	8'h44->8'h33
HWDATA[7:0]	8'h88
counter1	St0
counter3	St0
counter5	St0
counter[2:0]	3'h0
Group2	
New Group	

test_ahb_slave.HRDATA[7:0] 8'h33

finished: test_ahb_slave 795s

Figure 4.11: Error response with no wait cycles

In figure 4.12 shows a read after write with a write of 2 addresses and after immediate read them. The figure 4.13 shows an immediate read after write of a single address-single data.

The last figure (Figure 4.14) shows what happens if the HSEL signal becomes zero (the slave is not chosen to operate). The slave then does not produce any output signals e.g. HRDATA even if he gets address and data.

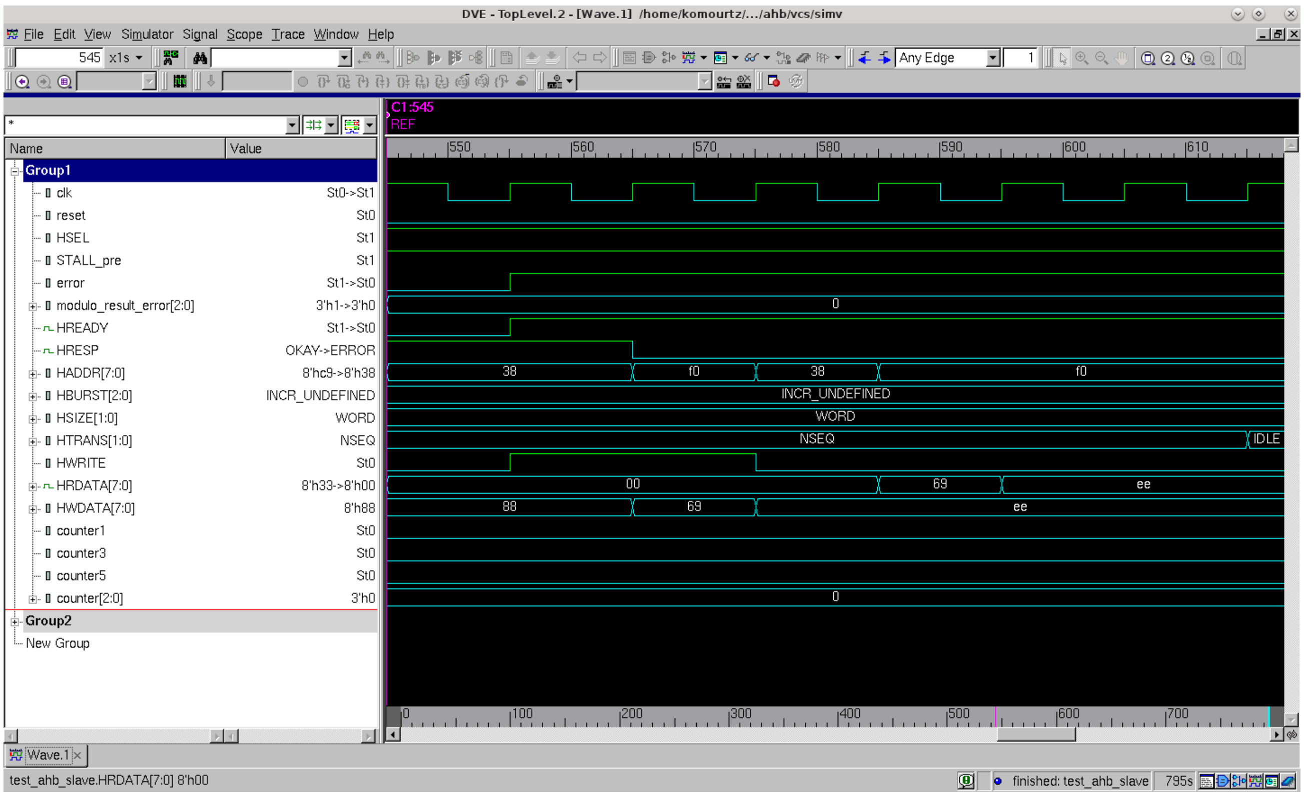


Figure 4.12: Read after write with two addresses

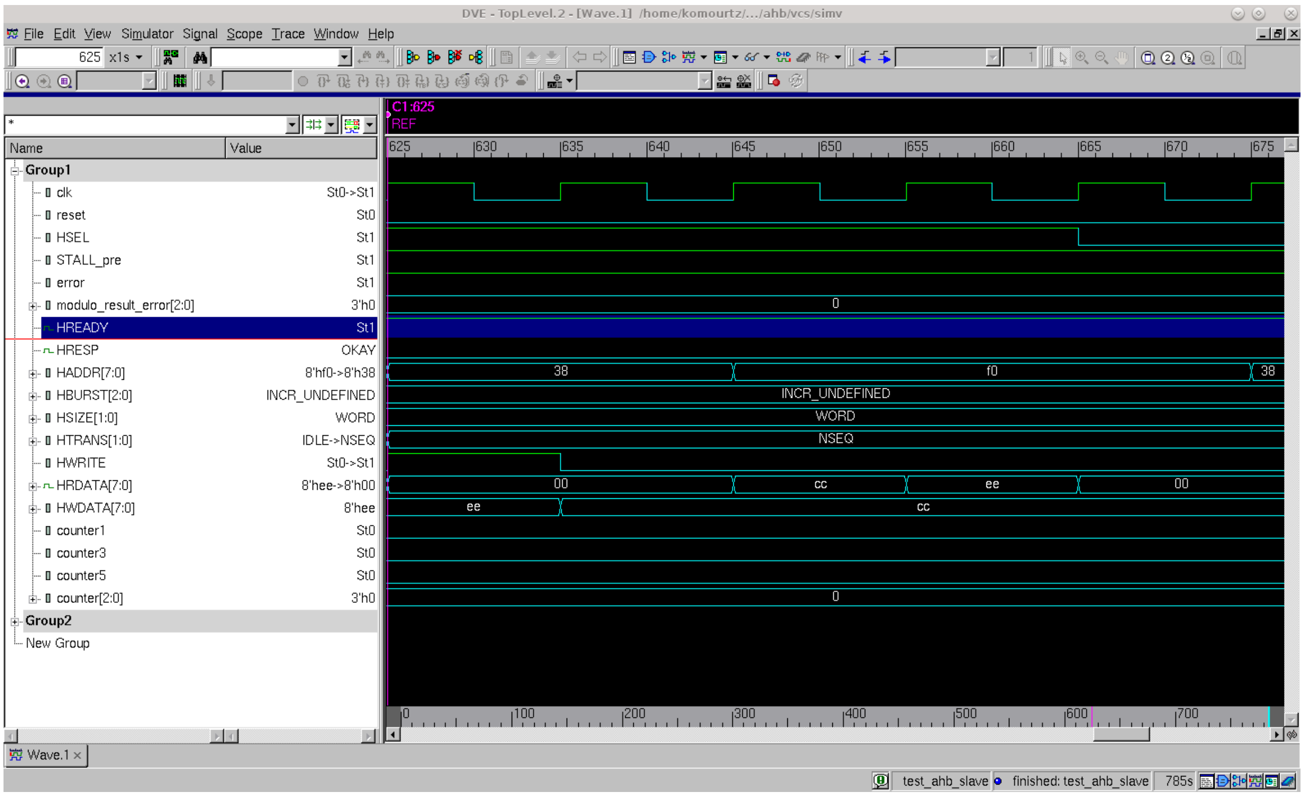


Figure 4.13: Read after write single address

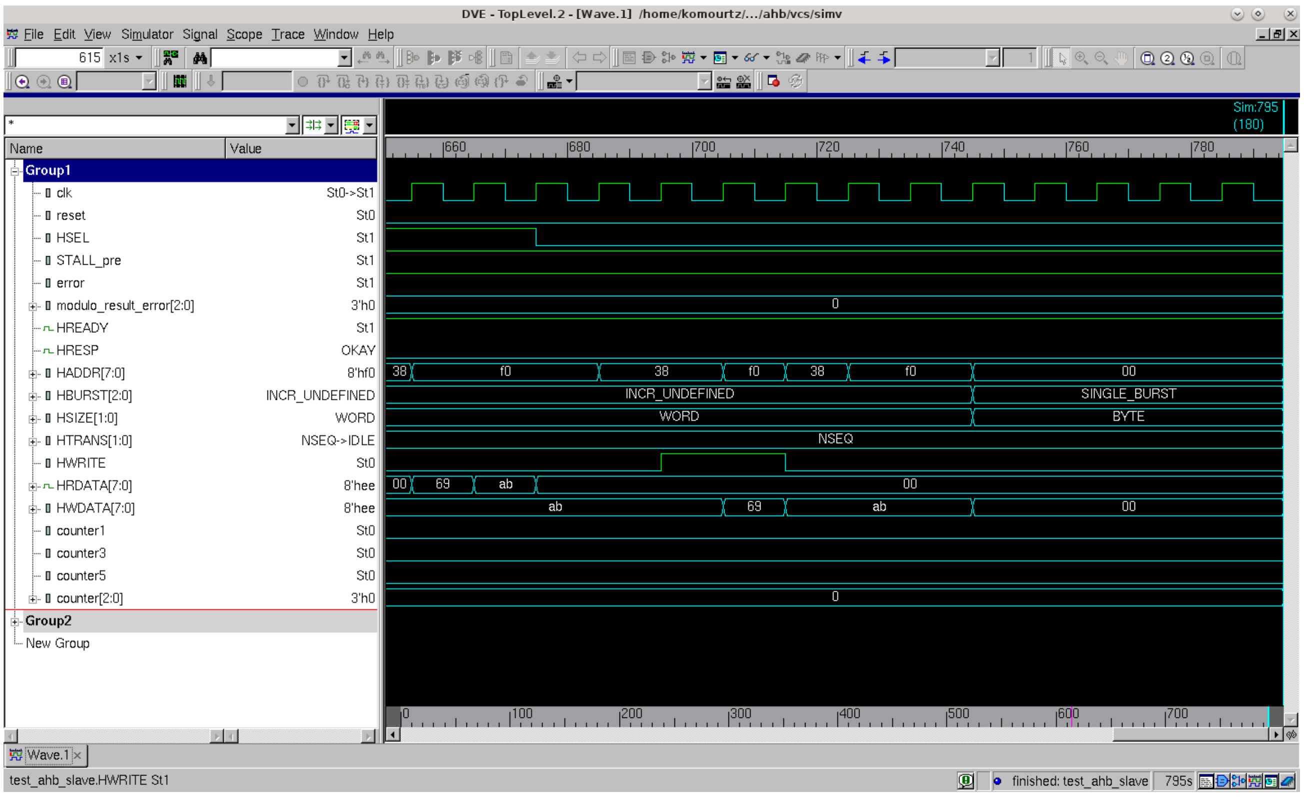


Figure 4.14: HSEL = 0

4.4 Logic Synthesis

In this chapter will be discussed some issues related with performing the design synthesis step using Design Compiler. A nice definition of synthesis can be found on [8]:

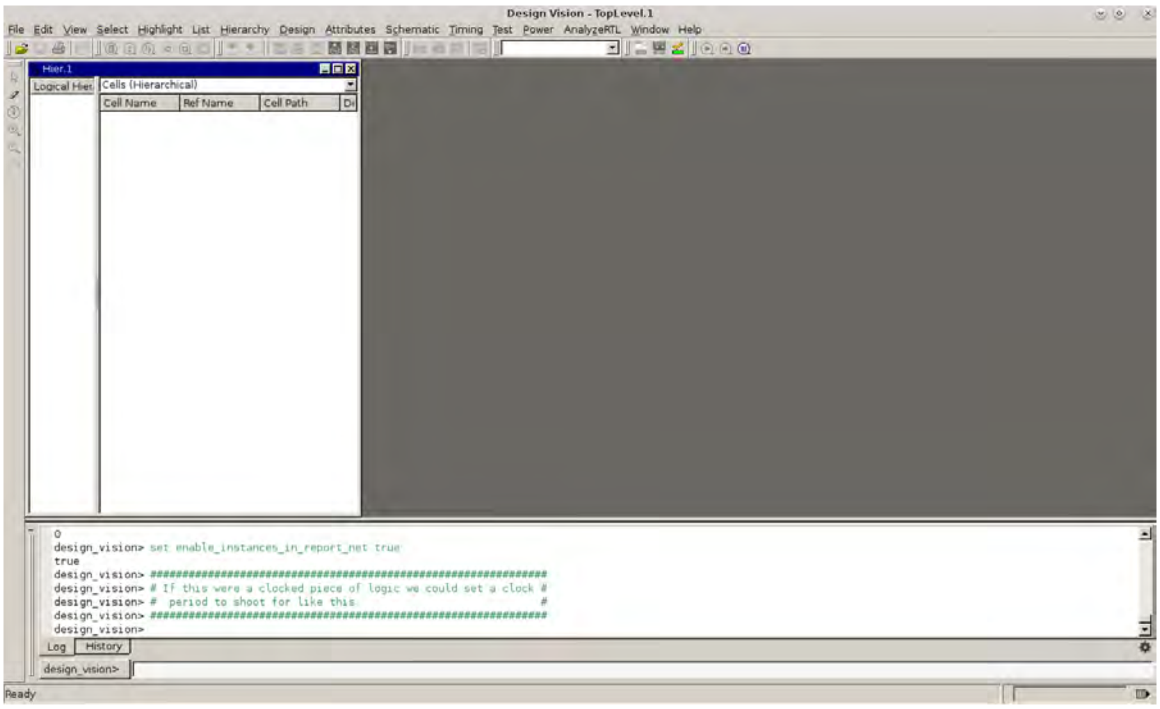
"Synthesis is the process of taking a design written in a hardware description language, such as VHDL, and compiling it into a netlist of interconnected gates which are selected from a user-provided library of various gates."

A figure of the initial screen of Synopsys Design Compiler is shown in the figure 4.15

4.4.1 Libraries Specification

The Synopsys synthesis tool when invoked, through Design compiler command, reads a startup file, which must be present in the current working directory. This startup file is **synopsys_dc.setup** file. In this thesis this file is integrated to the Tcl script that specifies all the constraints and specifications for the design to run. There should be two startup files present, one in the current working directory and other in the root directory in which Synopsys is installed. The local startup file in the current working directory should be used to specify individual design specifications. This file does not contain design dependent data. Its function is to load the Synopsys technology independent libraries and other parameters. The user in the startup files specifies the design dependent data. The settings provided in the current working directory override the ones specified in the root directory.

This step presents setup of basic library information. Design Compiler uses technology,


Design Vision - TopLevel.1
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
Hier.1
Logical Hier
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path
Log
History
Ready

```
0
design_vision> set enable_instances_in_report_net true
true
design_vision> #############################################################
design_vision> # If this were a clocked piece of logic we could set a clock #
design_vision> #  period to shoot for like this                             #
design_vision> #############################################################
design_vision>
```

```
design_vision>
```

Figure 4.15: Synopsys Design Compiler

symbol, and synthetic or Design Ware libraries to implement synthesis and to display synthesis results graphically. We should specify the link, target, symbol, and synthetic libraries for Design Compiler by using the link_library, target_library, symbol_library, and synthetic_library commands. There are four important parameters that should be setup before one can start using the tool.

Search path: This parameter is used to specify the synthesis tool all the paths that it should search when looking for a synthesis technology library for reference during synthesis or for the initial RTL files.

Link Library: Design Compiler uses the link library to resolve references. For a design to be complete, it must connect to all the library components and designs it references. This process is called linking the design or resolving references. The link_library variable specifies a list of libraries and design files that Design Compiler can use to resolve references. When you load a design into memory, Design Compiler also loads all libraries specified in the link_library variable.

Target Library: This library is used mainly for mapping all the logic gates from the target library. It also calculates the timing of the circuit, using the vendor-supplied timing data for these gates. The target_library specification should only contain those standard cell libraries that you want Design Compiler to use when mapping design standard cells. Standard cells are cells such as combinational logic and registers. The target_library specification should not include any DesignWare libraries or macro libraries such as I/O pads or memories. The target_library is a subset of the link_library and listed first in your list of link libraries.

Symbol Library: It is the library that contains all the definitions of the graphic symbols that represent library cells in the design schematics, when you generate the design schematic, Design Compiler performs a one-to-one mapping of cells in the netlist to cells in the symbol library.

Synthetic Library: The user does not need to specify the standard synthetic library (standard.sldb), which implements the built-in HDL operators. These operators include +, -, *, etc. and the operations defined by if and case statements. The Design Compiler software automatically uses this library without even loading in the setup file (.synopsys_dc.setup). If you are using additional DesignWare libraries, you must specify these libraries by using the synthetic_library variable (for optimization purposes) and the link_library variable (for cell resolution purposes).

Table 4.1 shows the default variables of library types at Synopsys Design Compiler.

Library type	Variable	Default	File extension
Target Library	target_library	{"your_library.db"}	.db
Link Library	link_library	{"", "your_library.db"}	.db
Symbol Library	symbol_library	{"your_library.sdb"}	.sdb
DesignWare Library	synthetic_library	{}	.sldb

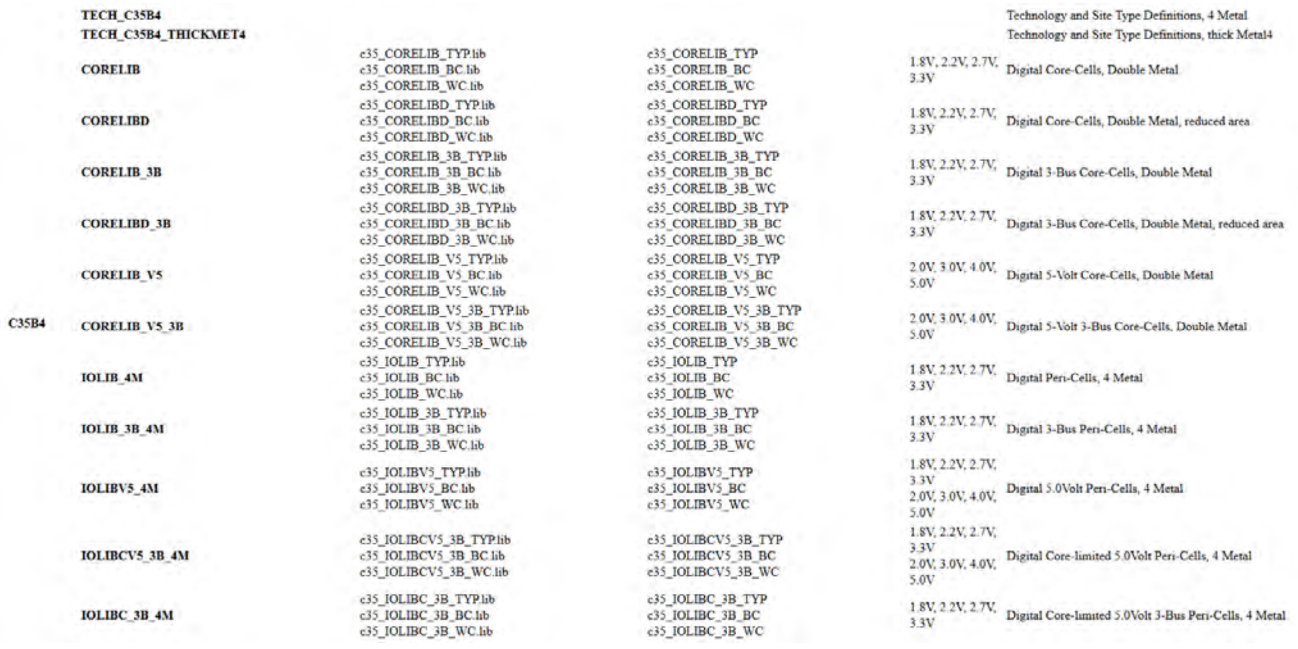
Table 4.1: Library variables

All of the different libraries that we talked about previously should be located in a special format in a file under a specific name (.synopsys_dc.setup) which must be located in the working directory of the user so it can be invoked by the tool. In fragment 4.7 is a sample of the (.synopsys_dc.setup) file that is integrated into the script of Design Compiler:

Listing 4.7: `.synopsys_dc.setup`

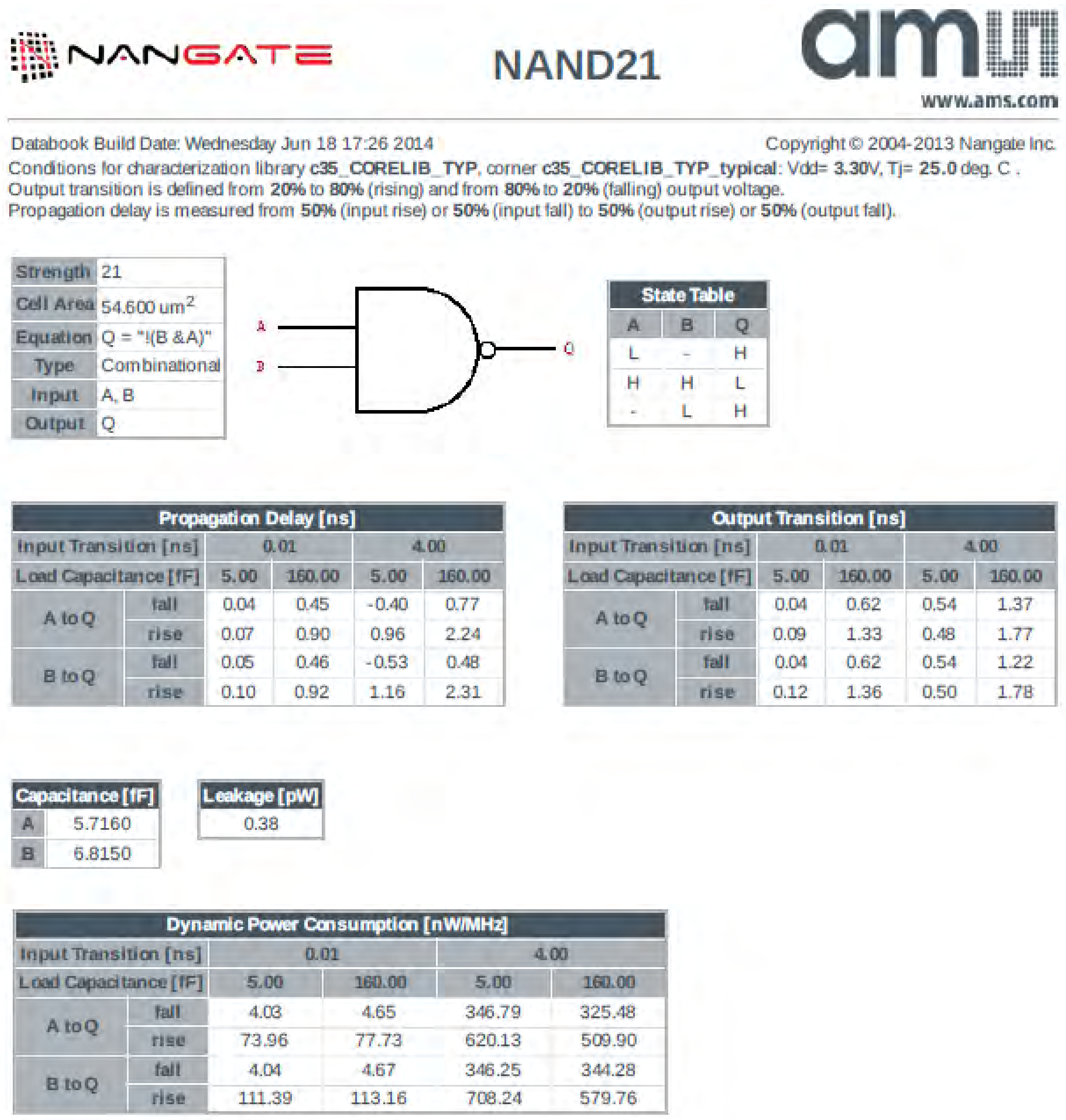
```
#########################################################
#/* All verilog files, separated by spaces        */#
#########################################################
set my_verilog_files "ahb_slave.v ahb_slave_mem.v ahb_slave_ram_fin.v slaveuniquify.v"
###########################
#/* Top-level Module */##
###########################
set my_toplevel slaveuniquify
set report_default_significant_digits 4

set lib_root "~/AMS/C35B4C3"
set project_root "~/km_fab_project"
set tool_root "/home1/eda/Synopsys_2015/synopsys/2014-15/RHELx86/SYN_2014.09-SP2"

#/***********************************************/
#/* No modifications needed below              */
#/***********************************************/
set search_path ". $lib_root/synopsys/c35v5_5.0V
                   $project_root/ahb
                   $tool_root/libraries/syn"

set target_library "c35_CORELIB_V5_WC.db"
set target_library [concat  $target_library c35_CORELIB_V5_BC.db]
set target_library [concat  $target_library c35_IOLIBV5_WC.db]
set target_library [concat  $target_library c35_IOLIBV5_BC.db]
set synthetic_library "standard.sldb dw_foundation.sldb"
set link_library "* $target_library $synthetic_library"
```

We had two sets of libraries, the ones with the D (c35_CORELIBD) that were we had all kind of cells/gates (AND, OR, NOR, NAND, as well as storing elements like flip-flops etc.) and the ones with the no D (c35_CORELIB) that were consists of the ones with 5 Voltage on their cells and with the same set of cells/gates and the ones with Voltage below 5 Volts (1.8-3.3 Voltage) that contained only the cells negative unate (NAND, NOR, etc.) and not the positive unate cells (AND, OR, etc.). Because gates like NAND, NOR has an integrated inverter they have smaller delay (which is our main constraint for the design) rather that gates like AND that the tool has to place an inverter right after the gate and it in general cause bigger delay than before in our circuit. Below we will present a comparison between the three different libraries on a single NAND gate with 2 inputs and 1 output. Figure 4.16 shows us the different libraries that our design library set contains (C35B4C3). The other 3 figures shows us the delays as well as the power consumption of NAND gates.



	TECH_C35B4				Technology and Site Type Definitions, 4 Metal
	TECH_C35B4_THICKMET4				Technology and Site Type Definitions, thick Metal4
	CORELIB	c35_CORELIB_TYP.lib c35_CORELIB_BC.lib c35_CORELIB_WC.lib	c35_CORELIB_TYP c35_CORELIB_BC c35_CORELIB_WC	1.8V, 2.2V, 2.7V, 3.3V	Digital Core-Cells, Double Metal
	CORELIBD	c35_CORELIBD_TYP.lib c35_CORELIBD_BC.lib c35_CORELIBD_WC.lib	c35_CORELIBD_TYP c35_CORELIBD_BC c35_CORELIBD_WC	1.8V, 2.2V, 2.7V, 3.3V	Digital Core-Cells, Double Metal, reduced area
	CORELIB_3B	c35_CORELIB_3B_TYP.lib c35_CORELIB_3B_BC.lib c35_CORELIB_3B_WC.lib	c35_CORELIB_3B_TYP c35_CORELIB_3B_BC c35_CORELIB_3B_WC	1.8V, 2.2V, 2.7V, 3.3V	Digital 3-Bus Core-Cells, Double Metal
	CORELIBD_3B	c35_CORELIBD_3B_TYP.lib c35_CORELIBD_3B_BC.lib c35_CORELIBD_3B_WC.lib	c35_CORELIBD_3B_TYP c35_CORELIBD_3B_BC c35_CORELIBD_3B_WC	1.8V, 2.2V, 2.7V, 3.3V	Digital 3-Bus Core-Cells, Double Metal, reduced area
	CORELIB_V5	c35_CORELIB_V5_TYP.lib c35_CORELIB_V5_BC.lib c35_CORELIB_V5_WC.lib	c35_CORELIB_V5_TYP c35_CORELIB_V5_BC c35_CORELIB_V5_WC	2.0V, 3.0V, 4.0V, 5.0V	Digital 5-Volt Core-Cells, Double Metal
C35B4	CORELIB_V5_3B	c35_CORELIB_V5_3B_TYP.lib c35_CORELIB_V5_3B_BC.lib c35_CORELIB_V5_3B_WC.lib	c35_CORELIB_V5_3B_TYP c35_CORELIB_V5_3B_BC c35_CORELIB_V5_3B_WC	2.0V, 3.0V, 4.0V, 5.0V	Digital 5-Volt 3-Bus Core-Cells, Double Metal
	IOLIB_4M	c35_IOLIB_TYP.lib c35_IOLIB_BC.lib c35_IOLIB_WC.lib	c35_IOLIB_TYP c35_IOLIB_BC c35_IOLIB_WC	1.8V, 2.2V, 2.7V, 3.3V	Digital Peri-Cells, 4 Metal
	IOLIB_3B_4M	c35_IOLIB_3B_TYP.lib c35_IOLIB_3B_BC.lib c35_IOLIB_3B_WC.lib	c35_IOLIB_3B_TYP c35_IOLIB_3B_BC c35_IOLIB_3B_WC	1.8V, 2.2V, 2.7V, 3.3V	Digital 3-Bus Peri-Cells, 4 Metal
	IOLIBV5_4M	c35_IOLIBV5_TYP.lib c35_IOLIBV5_BC.lib c35_IOLIBV5_WC.lib	c35_IOLIBV5_TYP c35_IOLIBV5_BC c35_IOLIBV5_WC	1.8V, 2.2V, 2.7V, 3.3V 2.0V, 3.0V, 4.0V, 5.0V	Digital 5.0Volt Peri-Cells, 4 Metal
	IOLIBCV5_3B_4M	c35_IOLIBCV5_3B_TYP.lib c35_IOLIBCV5_3B_BC.lib c35_IOLIBCV5_3B_WC.lib	c35_IOLIBCV5_3B_TYP c35_IOLIBCV5_3B_BC c35_IOLIBCV5_3B_WC	1.8V, 2.2V, 2.7V, 3.3V 2.0V, 3.0V, 4.0V, 5.0V	Digital Core-limited 5.0Volt Peri-Cells, 4 Metal
	IOLIBC_3B_4M	c35_IOLIBC_3B_TYP.lib c35_IOLIBC_3B_BC.lib c35_IOLIBC_3B_WC.lib	c35_IOLIBC_3B_TYP c35_IOLIBC_3B_BC c35_IOLIBC_3B_WC	1.8V, 2.2V, 2.7V, 3.3V	Digital Core-limited 5.0Volt 3-Bus Peri-Cells, 4 Metal

Figure 4.16: Libraries different flavors



NANGATE NAND21 ams www.ams.com

Databook Build Date: Wednesday Jun 18 17:26 2014 Copyright © 2004-2013 Nangate Inc.
Conditions for characterization library **c35_CORELIB_TYP**, corner **c35_CORELIB_TYP_typical**: Vdd= **3.30**V, Tj= **25.0** deg. C .
Output transition is defined from **20%** to **80%** (rising) and from **80%** to **20%** (falling) output voltage.
Propagation delay is measured from **50%** (input rise) or **50%** (input fall) to **50%** (output rise) or **50%** (output fall).

Strength	21
Cell Area	54.600 um^2
Equation	Q = "!(B &A)"
Type	Combinational
Input	A, B
Output	Q

State Table		
A	B	Q
L	-	H
H	H	L
-	L	H

Propagation Delay [ns]					
Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		5.00	160.00	5.00	160.00
A to Q	fall	0.04	0.45	-0.40	0.77
	rise	0.07	0.90	0.96	2.24
B to Q	fall	0.05	0.46	-0.53	0.48
	rise	0.10	0.92	1.16	2.31

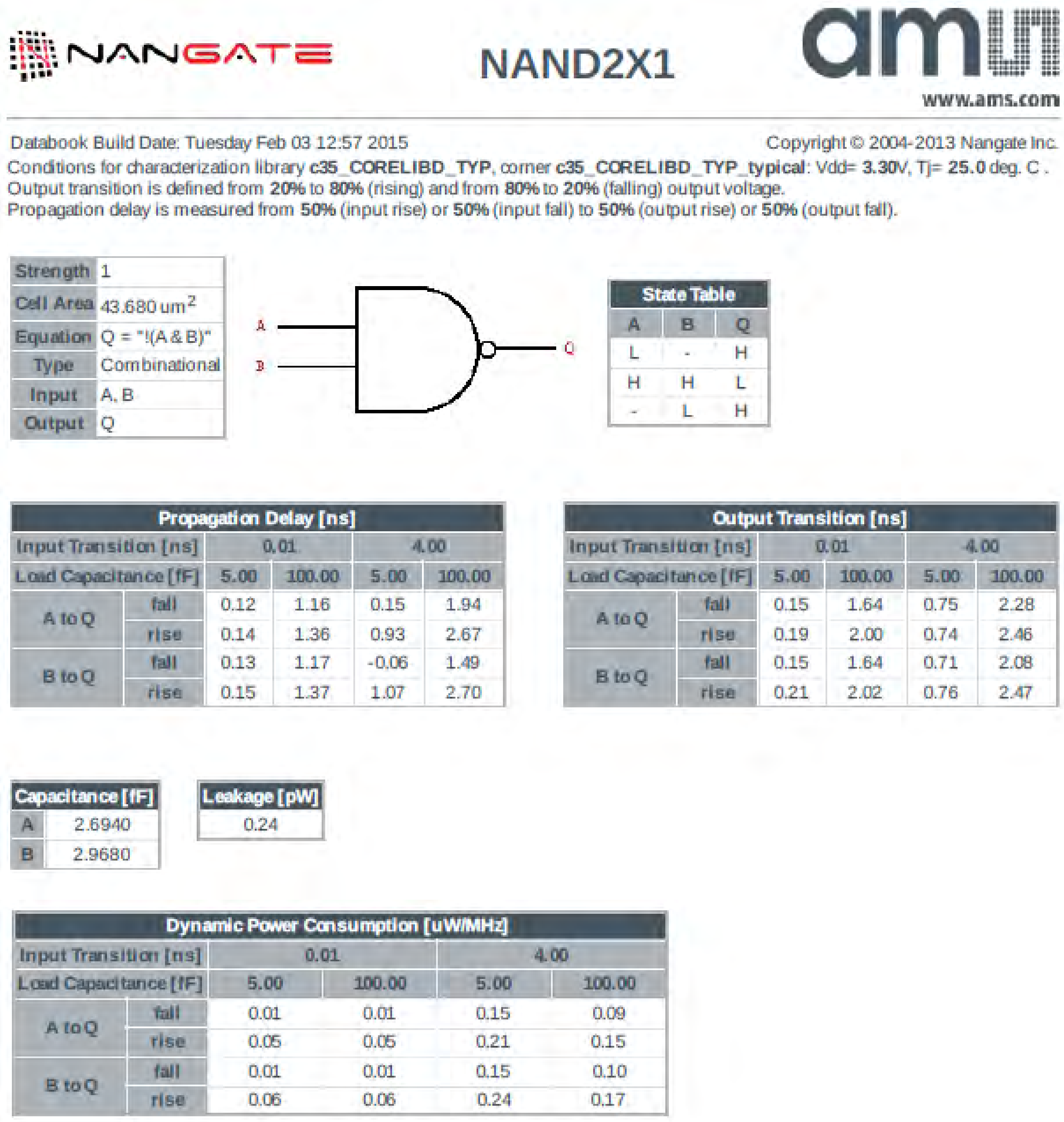
Output Transition [ns]					
Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		5.00	160.00	5.00	160.00
A to Q	fall	0.04	0.62	0.54	1.37
	rise	0.09	1.33	0.48	1.77
B to Q	fall	0.04	0.62	0.54	1.22
	rise	0.12	1.36	0.50	1.78

Capacitance [fF]	
A	5.7160
B	6.8150

Leakage [pW]
0.38

Dynamic Power Consumption [nW/MHz]					
Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		5.00	160.00	5.00	160.00
A to Q	fall	4.03	4.65	346.79	325.48
	rise	73.96	77.73	620.13	509.90
B to Q	fall	4.04	4.67	346.25	344.28
	rise	111.39	113.16	708.24	579.76

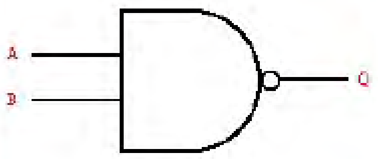
Figure 4.17: Corelib 2x1 Nand Gate



NANGATE **NAND2X1** ams www.ams.com

Databook Build Date: Tuesday Feb 03 12:57 2015 Copyright © 2004-2013 Nangate Inc.
Conditions for characterization library **c35_CORELIBD_TYP**, corner **c35_CORELIBD_TYP_typical**: Vdd= **3.30**V, Tj= **25.0** deg. C .
Output transition is defined from **20%** to **80%** (rising) and from **80%** to **20%** (falling) output voltage.
Propagation delay is measured from **50%** (input rise) or **50%** (input fall) to **50%** (output rise) or **50%** (output fall).

Strength	1
Cell Area	43.680 um^2
Equation	Q = "!(A & B)"
Type	Combinational
Input	A, B
Output	Q



State Table		
A	B	Q
L	-	H
H	H	L
-	L	H

Propagation Delay [ns]					
Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		5.00	100.00	5.00	100.00
A to Q	fall	0.12	1.16	0.15	1.94
	rise	0.14	1.36	0.93	2.67
B to Q	fall	0.13	1.17	-0.06	1.49
	rise	0.15	1.37	1.07	2.70

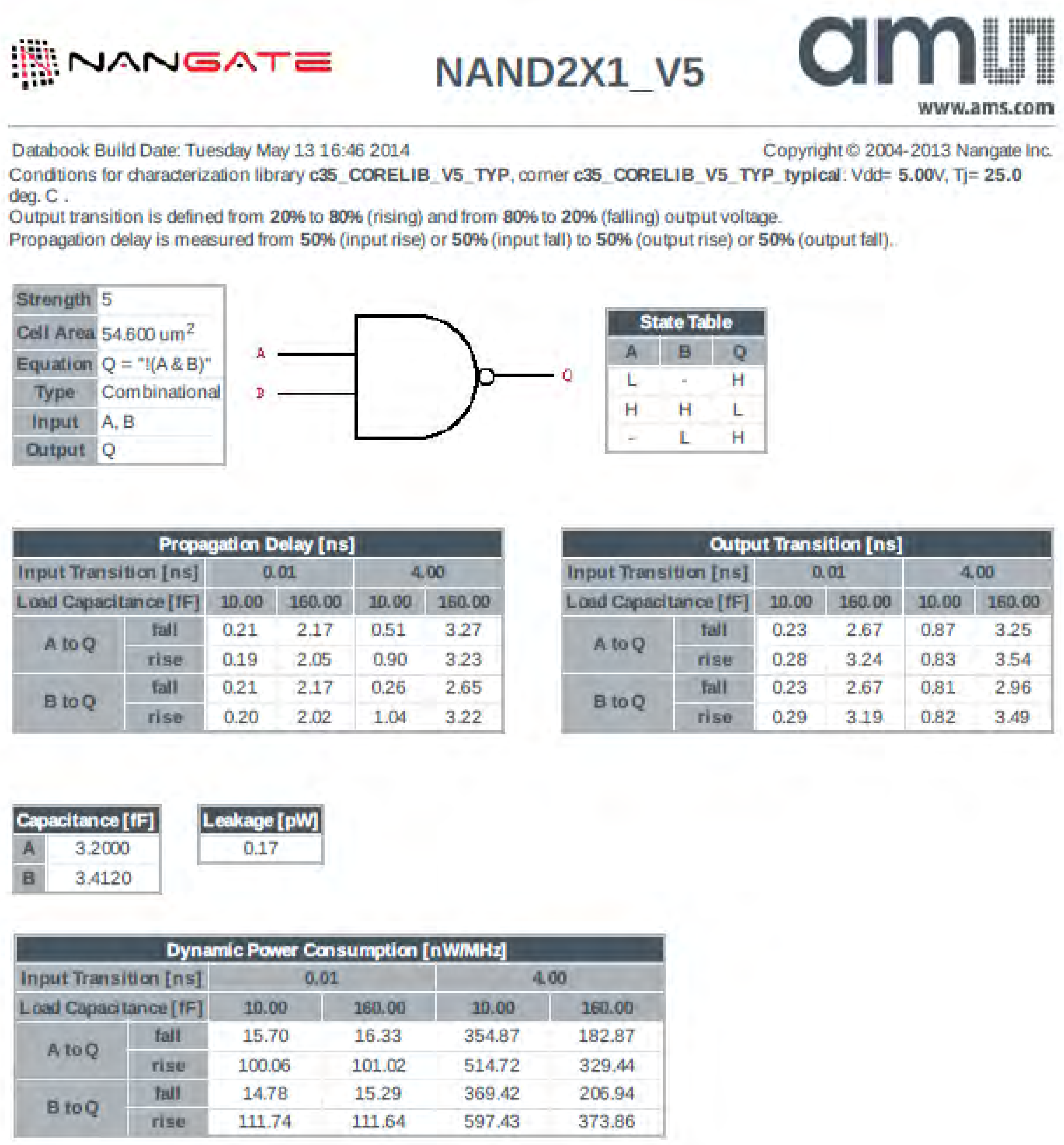
Output Transition [ns]					
Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		5.00	100.00	5.00	100.00
A to Q	fall	0.15	1.64	0.75	2.28
	rise	0.19	2.00	0.74	2.46
B to Q	fall	0.15	1.64	0.71	2.08
	rise	0.21	2.02	0.76	2.47

Capacitance [fF]	
A	2.6940
B	2.9680

Leakage [pW]
0.24

Dynamic Power Consumption [uW/MHz]					
Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		5.00	100.00	5.00	100.00
A to Q	fall	0.01	0.01	0.15	0.09
	rise	0.05	0.05	0.21	0.15
B to Q	fall	0.01	0.01	0.15	0.10
	rise	0.06	0.06	0.24	0.17

Figure 4.18: Corelibd 2x1 Nand Gate



NANGATE NAND2X1_V5 ams www.ams.com

Databook Build Date: Tuesday May 13 16:46 2014 Copyright © 2004-2013 Nangate Inc.
Conditions for characterization library **c35_CORELIB_V5_TYP**, corner **c35_CORELIB_V5_TYP_typical**: Vdd= **5.00**V, Tj= **25.0** deg. C .
Output transition is defined from **20%** to **80%** (rising) and from **80%** to **20%** (falling) output voltage.
Propagation delay is measured from **50%** (input rise) or **50%** (input fall) to **50%** (output rise) or **50%** (output fall).

Strength	5
Cell Area	54.600 um^2
Equation	Q = "!(A & B)"
Type	Combinational
Input	A, B
Output	Q

State Table

A	B	Q
L	-	H
H	H	L
-	L	H

Propagation Delay [ns]

Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		10.00	160.00	10.00	160.00
A to Q	fall	0.21	2.17	0.51	3.27
	rise	0.19	2.05	0.90	3.23
B to Q	fall	0.21	2.17	0.26	2.65
	rise	0.20	2.02	1.04	3.22

Output Transition [ns]

Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		10.00	160.00	10.00	160.00
A to Q	fall	0.23	2.67	0.87	3.25
	rise	0.28	3.24	0.83	3.54
B to Q	fall	0.23	2.67	0.81	2.96
	rise	0.29	3.19	0.82	3.49

Capacitance [fF]	
A	3.2000
B	3.4120

Leakage [pW]
0.17

Dynamic Power Consumption [nW/MHz]

Input Transition [ns]		0.01		4.00	
Load Capacitance [fF]		10.00	160.00	10.00	160.00
A to Q	fall	15.70	16.33	354.87	182.87
	rise	100.06	101.02	514.72	329.44
B to Q	fall	14.78	15.29	369.42	206.94
	rise	111.74	111.64	597.43	373.86

Figure 4.19: Corelib Nand 5Volts 2x1

4.4.2 Read Design

After setting the startup file for Design Compiler and specifying all the required libraries and RTL files that needed for the design we have to proceed in reading the design and create the WORK folders for our design.

4.4.2.1 Analyze and Elaborate

Executing analyze command does the following:

- Reads an HDL source file
- Checks it for errors (without building generic logic for the design)
- Creates HDL library objects in an HDL-independent intermediate format
- Stores the intermediate files in a location you define

If the analyze command reports errors, fix them in the HDL source file and run analyze again. After a design is analyzed, you must reanalyze it only when you change it. Executing elaborate command does the following:

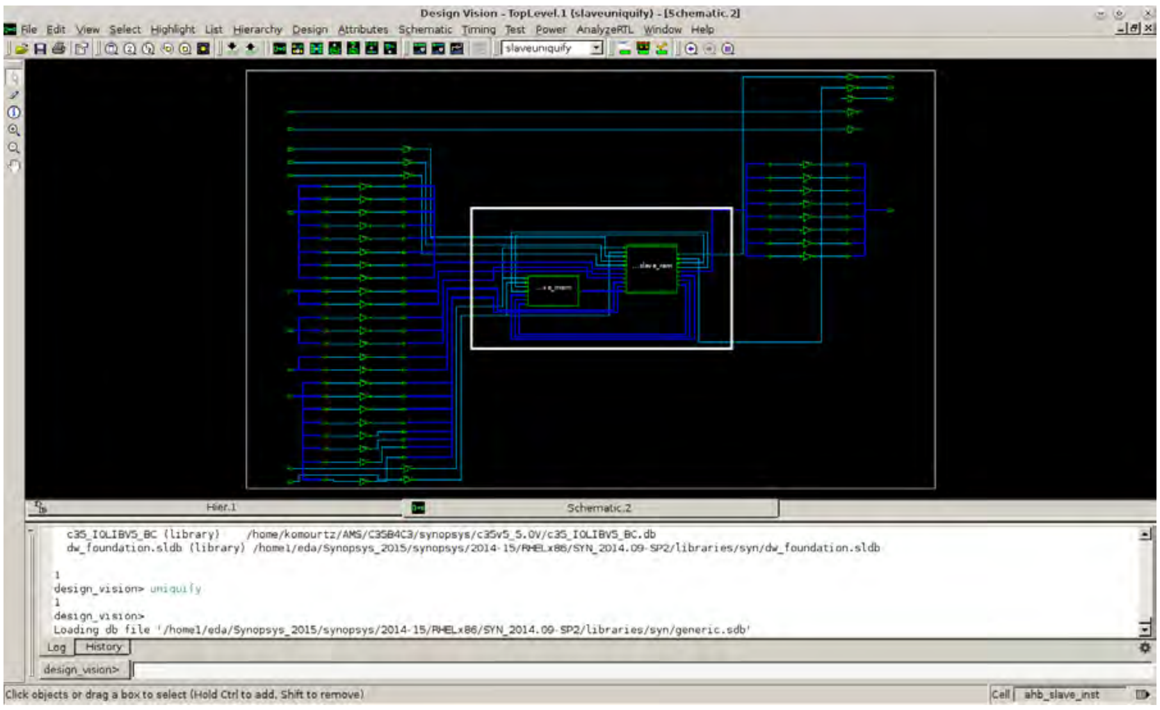
- Translates the design into a technology-independent design (GTECH) from the intermediate files produced during analysis.
- Allows changing of parameter values defined in the source code.
- Allows verilog architecture selection.
- Replaces the HDL arithmetic operators in the code with DesignWare components.
- Automatically executes the link command, which resolves design references.

After the elaboration the design is like the figure shown below (Figure 4.20):

4.4.2.2 Read File

Executing read_file command does the following:

- Reads several different formats (.ddc, .vhd, .v, .db).
- Performs the same operations as analyze and elaborate in a single step
- Creates .mr and .st intermediate files for VHDL
- Does not execute the link command automatically.
- Does not create any intermediate files for Verilog (However, you can have the read_file command create intermediate files by setting the hdlin_auto_save_templates variable to true.


Design Vision - TopLevel.1 (slaveuniquify) - [Schematic.2]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
slaveuniquify
Hier.1
Schematic.2
Log
History
Cell
ahb_slave_inst

```
  c35_IOLIBV5_BC (library)    /home/komourtz/AMS/C35B4C3/synopsys/c35v5_5.0V/c35_IOLIBV5_BC.db
  dw_foundation.sldb (library) /home1/eda/Synopsys_2015/synopsys/2014-15/RHELx86/SYN_2014.09-SP2/libraries/syn/dw_foundation.sldb

1
design_vision> uniquify
1
design_vision>
Loading db file '/home1/eda/Synopsys_2015/synopsys/2014-15/RHELx86/SYN_2014.09-SP2/libraries/syn/generic.sdb'
```

design_vision>

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

Figure 4.20: Top level module after elaboration

4.4.3 Define Design Environment

After reading the design files, and before doing any optimization we have to define the design environment of our design. Design environment is the operating environment where the design is expected to operate in should be defined. The design environment defines the environment in which the design is expected to operate. Design environment includes parameters such as operating conditions (specify variations in temperature, voltage, and manufacturing process), wire load models (WLM; used to specify the effect of the interconnects on the timing and area), and system interface characteristics (input drives, I/O loads, and fanout loads). The environment model directly affects design synthesis results. In Design Compiler, the model is defined by a set of attributes and constraints that you assign to the design, using specific dc_shell commands.

Operating conditions

Most technology libraries used has different and predefined operating conditions. We have to report the library using command report_lib to list the operating conditions defined in the library. The library should be loaded first in memory before running the command report_lib. In order to see the list of libraries loaded in memory, use command list_libs. Operating condition describes the Voltage, Temperature, Interconnect model and process of the design. Each operating condition predefined with its specific temperature, voltage, interconnect model. There are mostly common in most technology libraries WORST, BEST and TYPICAL but the names are library dependent. Users should ask the vendor which is the best operating condition to be used.

By changing the value of the operating condition command, full ranges of process variations are covered. The WORST case operating condition is generally used during pre-layout synthesis phase, thereby optimizing the design for maximum setup-time. The BEST case condition is commonly used to fix the hold-time violations. The TYPICAL case is mostly ignored, since analysis at WORST and BEST case also covers the TYPICAL case. The next figure describes the relationship between the different operating conditions in our design and their affect on the design itself (Figure 4.21)

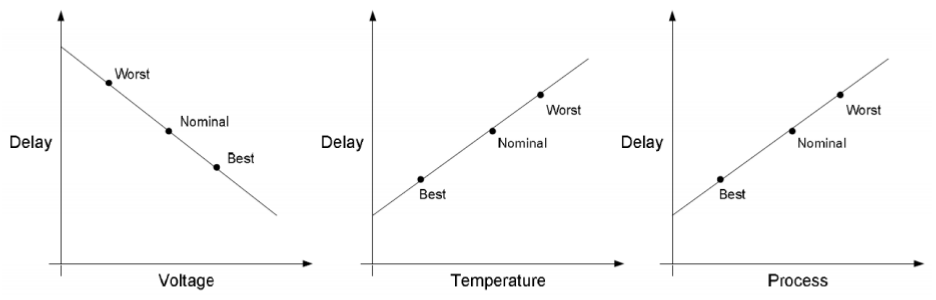


Figure 4.21: Operating conditions comparison

Wire-Load

Wire load modeling allows the user to estimate the effect of wire length and fanout on the resistance, capacitance, and area of nets. Design Compiler uses these physical values to calculate wire delays and circuit speeds. The vendors usually develop wire load model, depending on some statistical information. The models include values for area, capacitance, and resistance per unit length.

Design Compiler supports three modes for determining which wire load model to use for nets that cross hierarchical boundaries:

- **TOP:** This wire load model is used as if there is no hierarchy, in which the compiler models all the nets in the top level or in the sub-designs using the wire load model specified for the top level model.
- **ENCLOSED:** Design Compiler uses the wire load model of the smallest design in the hierarchy. If the design enclosing the net has no wire load model specified, the tool keep tracing in upward direction until it finds a wire load model. ENCLOSED wire load model is more accurate than top model when cells in the same design are placed near to each other during layout.
- **SEGMENTED:** Design Compiler determines the wire load model of each segment of a net by the design encompassing the segment. Nets crossing boundaries are divided into segments. For each net segment, Design Compiler uses the wire load model of the design containing the segment.

If the design has a net that has no wire load model specified, the tool keep tracing in upward direction until it finds a wire load model. Below is show an image of the wire-load models (figure 4.22)

In our design, the wire-load model that each module uses, is automatically chosen from the area of the design has after compilation. For the top module the tool chooses to use the 30K wire-load model while in the sub modules of the design the tool chooses to use the 10K wire load model.

Table 4.2 shows the different results considering Power, Area, Slack with different available wire load models from the c35_CORELIB_WC design lib. We run the same script and we change the wire load model on each run. The clock period was at 12 ns at all runs, so the slack (positive or negative) is with this touchstone. The results are the following:

Drive-Load

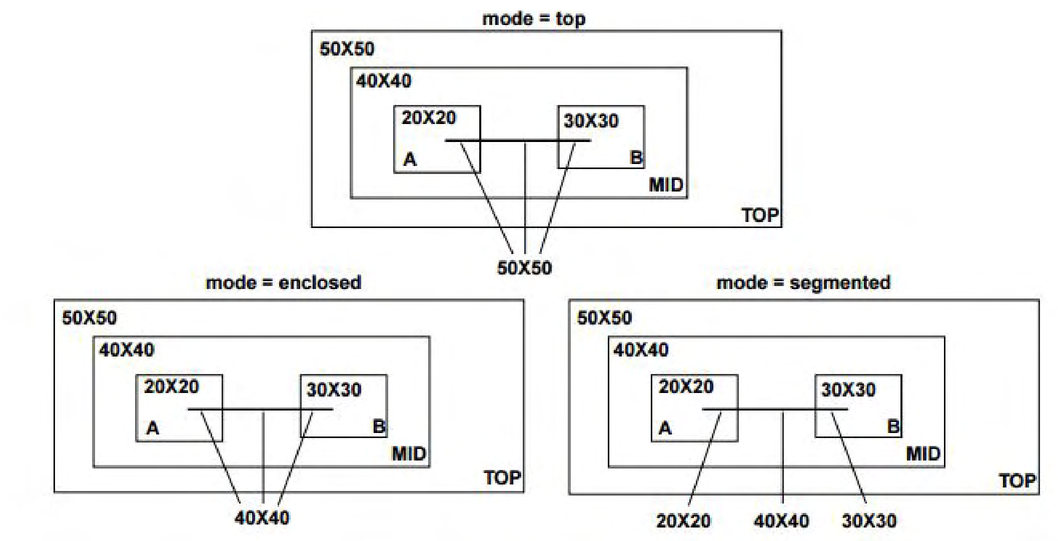


Figure 4.22: Wire load models

Wire load model	10k	30k	100k	pad_wire_laod
Internal Power (mW)	44.391083	44.110794	44.436543	44.249599
Switching Power (mW)	24.530565	27.468857	33.805050	55.361717
Leakage Power (pW)	2.404035e+07	2.403981e+07	2.466101e+07	
Worst negative slack (ns)	0.40160	0.550897	0.276166	0.011813
Area (um^2)	2594710.012337	2627976.697029	2705388.421537	2954650.821393

Table 4.2: Wire load models comparison

- **Load:** Each output can specify the drive capability that determines how many loads can be driven within a particular time. Each input can have a load value specified that determines how much it will slow a particular driver. Signals that are arriving later than the clock can have an attribute that specifies this fact. The load attribute specifies how much capacitance load exists on a particular output signal. The load value is specified in the units of the technology library in terms of picofarads or standard loads, etc.

- **Drive:** The drive specifies the drive strength at the input port. It is specified as a resistance value. This value controls how much current a particular driver can source. The larger a driver is, i.e 0 resistance, the faster a particular path will be, but a larger driver will take more area, so the designer needs to trade off speed and area for the best performance

Below is shown an image of what Drive-Load (4.23)

4.4.4 Set Design Constraints

When Design Compiler comes to optimize the design, it uses two types of constraints:

- **Design rule constraints:** These are implicit constraints; the technology library defines them. These constraints are requirements for a design to function correctly, and they apply to any design using the library. You can make these constraints more restrictive than optimization constraints.

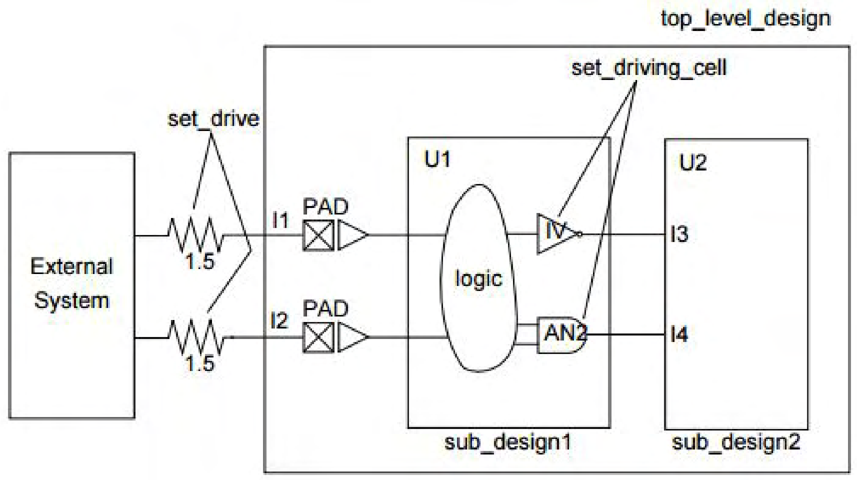


Figure 4.23: Driving characteristics

- **Optimization constraints:** These are explicit constraints; you define them. Optimization constraints apply to the design on which you are working for meeting the design goals. They must be realistic. Design Compiler tries to meet both design rule constraints and optimization constraints, but design rule constraints take precedence.

4.4.5 Design Rule Constraints

Design rule constraints reflect technology-specific restrictions the design must meet in order to function as intended. The design rule constraints include:

- Maximum transition time
- Maximum fanout
- Minimum and maximum capacitance
- Cell degradation

DRCs are applied to the nets of the design in association to the pins of the cells from the technology library. Design compiler cannot violate the DRCs, even if it means to violate the optimization constraints (area or speed). User can apply design rule constraints more restrictive than the default constraints set by the technology library, but these constraints cannot be less restrictive.

4.4.6 Design Optimization Constraints

These types of constraints are being set to meet the design goals or requirements in terms of area, power and speed. It is recommended that designers specify realistic constraints, since unrealistic specification results in excess area, increased power and/or degradation in timing. The optimization constraints include

- Timing constraints (performance and speed)
- Input and output delays (synchronous paths)
- Minimum and maximum delay (asynchronous paths)

- Maximum area (number of gates)

The most important design constraint for a synchronous design is the system clock. The constraints to model the behavior of the system clock and the clock network are clock period, clock uncertainty, clock latency, and clock transition time.

- **Period:** The time at which the clock pulse repeats itself.
- Uncertainty: Clock uncertainty can be used to model the variation of the clock propagation times in the different branches of the clock tree. This variation is due to different lengths of these branches (clock skew) but uncertainty can also be used to model clock jitter (variation in clock period length).
- **Latency:** Clock latency models the time it takes the clock signal to propagate from the clock source to the clock capture point (register clock pins).
- **Transition time:** Clock transition time can be used to model the time it takes a clock signal to change state.

create_clock: This command is used to define the clock that will be used in the design. This command does not imply that the synthesizer will create a circuit to implement the waveform. It tells the synthesizer that the specified periodic waveform will be input at a clock port and it needs to make a circuit that will work with it.

set_clock_latency: This is used to specify that there will be a delay of unit time delays through buffers that are not modeled in the design. It is very useful in hierarchical designs, when constraining sub-modules. This is primarily used during the pre-layout synthesis and timing analysis. The estimated delay number is an approximation of the delay produced by the clock tree network insertion (done during the layout phase).

set_clock_uncertainty: This command is used for setting a margin for setup and hold time of the clock. When clock arrives simultaneously to every register, the arrival time won't be the same at every register. There will be skew. During the pre-layout phase one can add more time margin as compared to the post-layout phase. This is to specify the worst case sum of clock skew and jitter.

set_input_delay: This command is used to specify the arrival of the signal coming from outside that will not arrive at the beginning of the clock period. The compiler should know this. It is used at the input ports to specify the time it takes for the data to be stable after the clock edge. It is the delay provided at the input port by the external logic. (Input delay = FF delay + external Combinational delay)

set_output_delay: This command is used to specify the time taken by the signal to be available before the clock edge. It is used at the output ports to specify the time it takes for the data to be stable before the clock edge. It is the delay at the output port that has been caused by the external logic delay. (Maximum Output delay = Setup time of FF + max. external logic delay) (Minimum Output delay = min. external logic delay – hold time of FF)

set_false_path: Design Compiler does not report false paths in the timing report or consider them during timing optimization. Use the set_false_path command to specify a false path. Use this command to ignore paths that are not timing-critical, that can mask other paths that must be considered during optimization, or that never occur in normal operation.

set_fix_multiple_port_nets -all: Design Compiler by default makes single clock cycle timing a requirement for all the paths. Design Compiler automatically infers single-cycle timing from clock waveforms and from input delay and output delay information. Single-cycle timing means that data should reach from start point to end point in a single clock cycle. You can change the default behavior for multi-cycle paths. A multi-cycle path is a timing path that is not expected to propagate a signal in one cycle. Multi-cycle paths are exceptions to the default single-cycle timing. You can set multi-cycle paths to direct Design Compiler to allow multiple clock cycles for data to propagate along a path. You can always reset paths to single-cycle timing.

set_max_delay: This command defines the maximum delay required in terms of time units for a particular path. In general, it is used for the blocks that contain combinational logic only. However, it may also be used to constrain a block that is driven by multiple clocks, each with a different frequency.

set_min_delay: This command is the opposite of the set_max_delay command, and is used to define the minimum delay required in terms of time units for a particular path.

set_max_area: This command specifies the maximum allowable area for the current design. Design Compiler computes the area of a design by adding the areas of each component on the lowest level of the design hierarchy (and the area of the nets). Maximum area represents the number of gates in the design, not the physical area the design occupies.

4.4.7 Compile strategy

Synopsys recommends the following compilation strategies that depend entirely on how your design is structured and defined. It is up to user discretion to choose the most suitable compilation strategy for a design. There are mainly two ways for compiling the design: **compile** and **compile_ultra**. Because after running many tests we observed that using compile_ultra caused the design having in the end of synthesis Design Rule Costs problems (The design rule cost is a indication of how many cells violate one of the standard cell library design rules constraints.), we decided to use the simple compile command with the high optimization on map, area, power. The compile cost function consists of design rule costs and optimization costs. By default, Design Compiler prioritizes costs in the following order:

1. Design rule costs
   - (a) Connection class
   - (b) Multiple port nets
   - (c) Maximum transition time
   - (d) Maximum fanout
   - (e) Maximum capacitance
   - (f) Cell degradation
2. Optimization costs
   - (a) Maximum delay
   - (b) Minimum delay
   - (c) Maximum power
   - (d) Maximum area
   - (e) Minimum porosity

The compile cost function considers only those components that are active on your design. Design Compiler evaluates each cost function component independently, in order of importance. When evaluating cost function components, Design Compiler considers only violators (positive difference between actual value and constraint) and works to reduce the cost function to 0. Design Compiler tries to meet all constraints but, by default, gives emphasis to design rule constraints because design rule constraints are requirements for functional designs. Using the default priority, Design Compiler fixes design rule violations even at the cost of violating your delay or area constraints

4.4.7.1 Top-Down Hierarchical compile method

Top-Down hierarchical compile method was generally used to synthesize very small designs (less than 10K gates). Using this method, the source is compiled by reading the entire design. Based on the design specifications, the constraints and attributes are applied, only at the top level. Although, this method provides an easy push-button approach to synthesis, it was extremely memory intensive and viable only for very small designs. The advantages of this method are:

- Only top level constraints are needed.
- Better results due to optimization across entire design.

While the drawbacks are:

- Long compile times.
- Incremental changes to the sub-blocks require complete re-synthesis. design.
- Does not perform well, if design contains multiple clocks or generated clocks.

4.4.7.2 Bottom-Up Hierarchical compile method

The designer manually specifies the timing requirements for each block of the design, thereby producing multiple synthesis scripts for individual blocks. The synthesis is usually performed bottom-up i.e., starting at the lowest level and ascending to the topmost level of the design. This method targets medium to very large designs and does not require large amounts of memory. The advantages of this method are:

- Easier to manage the design because of individual scripts.
- Incremental changes to the sub-blocks do not require complete re-synthesis of the entire design.
- Does not suffer from design style e.g., multiple and generated clocks are easily managed
- Good quality results in general because of flexibility in targeting and optimizing individual blocks.

While the drawbacks are:

- Tedious to update and maintain multiple scripts
- Critical paths seen at the top-level may not be critical at lower level.
- The design may need to be incrementally compiled in order to fix the DRCs.

4.4.8 Optimizing the Design

Optimization is the Design Compiler synthesis step that maps the design to an optimal combination of specific target library cells, based on the design's functional, speed, and area requirements. Design Compiler provides options that enable you to customize and control optimization. Design Compiler performs the following three levels of optimization:

- Architectural optimization
- Logic-level optimization
- Gate-level optimization

4.4.8.1 Architectural Optimization

Architectural optimization works on the HDL description. It includes such high-level synthesis tasks as:

- Sharing common sub-expressions
- Sharing resources
- Selecting DesignWare implementations
- Reordering operators
- Identifying arithmetic expressions for data-path synthesis (DC Ultra only).

4.4.8.2 Logic-Level Optimization

Logic-level optimization works on the GTECH netlist. It consists of the following two processes:

- **Flattening:** Flattening is a common academic term for reducing logic to a 2-level AND/OR representation. DC uses this approach to remove all intermediate variables and parenthesis (using Boolean distributive laws) in order to optimize the design. This option is set to "false" by default. It is useful for speed optimization because it leads to just two levels of combinational logic.
- **Structuring:** Structuring is used for designs containing regular structured logic, for e.g., a carry-look-ahead adder. It is enabled by default for timing only. When structuring, DC adds intermediate variables that can be factored out. This enables sharing of logic that in turn results in reduction of area. Structuring comes in two flavors: timing (default) and Boolean optimization. The latter is a useful method of reducing area, but has a greater impact on timing.

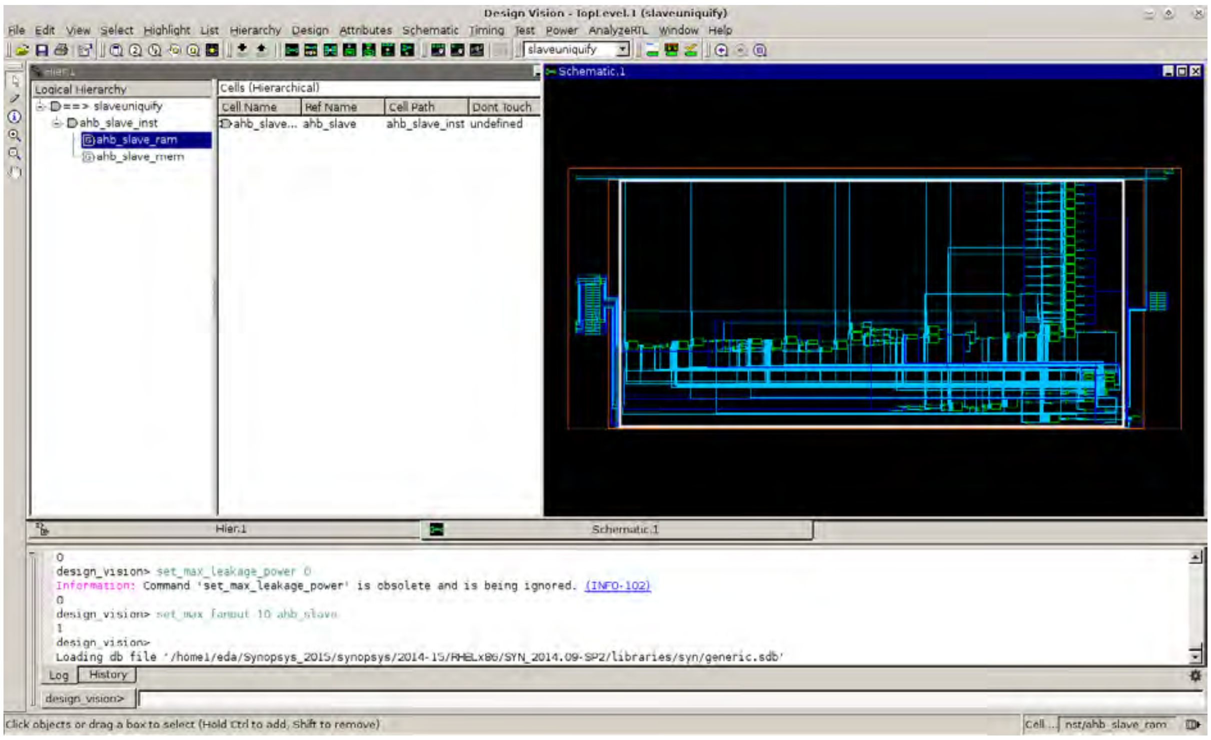
4.4.8.3 Gate Level Optimization

Gate-level optimization works on the generic netlist created by logic synthesis to produce a technology-specific netlist. It includes the following processes:

- **Mapping:** This process uses gates (combinational and sequential) from the target technology libraries to generate a gate-level implementation of the design whose goal is to meet timing and area goals.
- **Delay Optimization:** The process goal is to fix delay violations introduced in the mapping phase. Delay optimization does not fix design rule violations or meet area constraints.

- **Design Rule Fixing:** The process goal is to correct design rule violations by inserting buffers or resizing existing cells. Design Compiler tries to fix these violations without affecting timing and area results, but if necessary, it does violate the optimization constraints.
- **Area Optimization:** The process goal is to meet area constraints after the mapping, delay optimization, and design rule fixing phases are completed.

The figure 4.24 shows the AHB-lite slave RAM module before and after compiling the design. The post compilation figure is after the scan chain insertion that will be explained at the next section (4.5)


Design Vision - TopLevel.1 (slaveuniquify)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
slaveuniquify
Hier.1
Schematic.1
Logical Hierarchy
==> slaveuniquify
ahb_slave_inst
ahb_slave_ram
ahb_slave_mem
Log
History
Cell...
nst/ahb_slave_ram

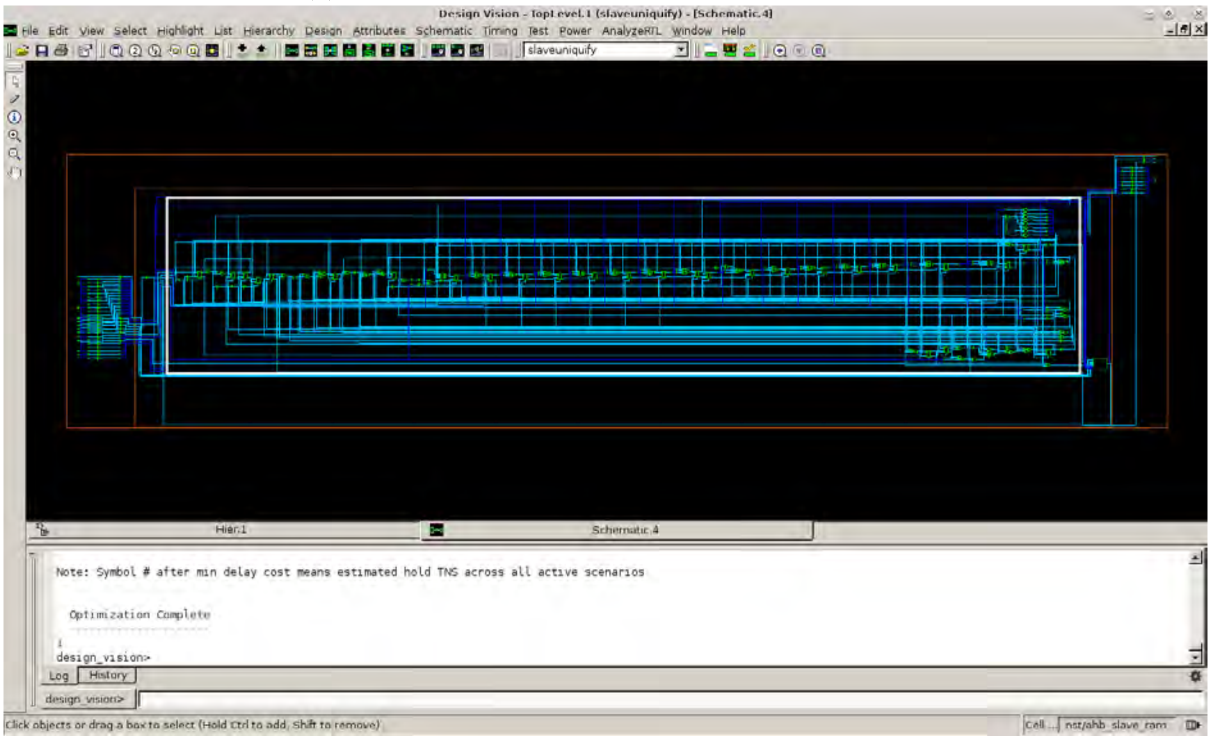
Cells (Hierarchical)

Cell Name	Ref Name	Cell Path	Dont Touch
ahb_slave...	ahb_slave	ahb_slave_inst	undefined

```
0
design_vision> set_max_leakage_power 0
Information: Command 'set_max_leakage_power' is obsolete and is being ignored. (INFO-102)
0
design_vision> set_max_fanout 10 ahb_slave
1
design_vision>
Loading db file '/home1/eda/Synopsys_2015/synopsys/2014-15/RHELx86/SYN_2014.09-SP2/libraries/syn/generic.sdb'
```

```
design_vision>
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
```

(a) AHB-lite slave RAM module before compilation


Design Vision - TopLevel.1 (slaveuniquify) - [Schematic.4]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
slaveuniquify
Hier.1
Schematic.4
Log
History
Cell...
nst/ahb_slave_ram

```
Note: Symbol # after min delay cost means estimated hold TNS across all active scenarios


  Optimization Complete
  ---------------------
1
design_vision>
```

```
design_vision>
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
```

(b) AHB-lite slave RAM module after compilation

Figure 4.24: AHB-lite slave RAM module before vs after compilation

4.5 Scan Chain Insertion, Design For Test

In Section 2.3 we gave an overview of DFT techniques and their necessity in design that are going to fabricate was given. This section will focus in the use of Design Compiler to implement a scan chain with the multiplexed flip-flop style. A figure of scan chain insertion is shown below (4.25):

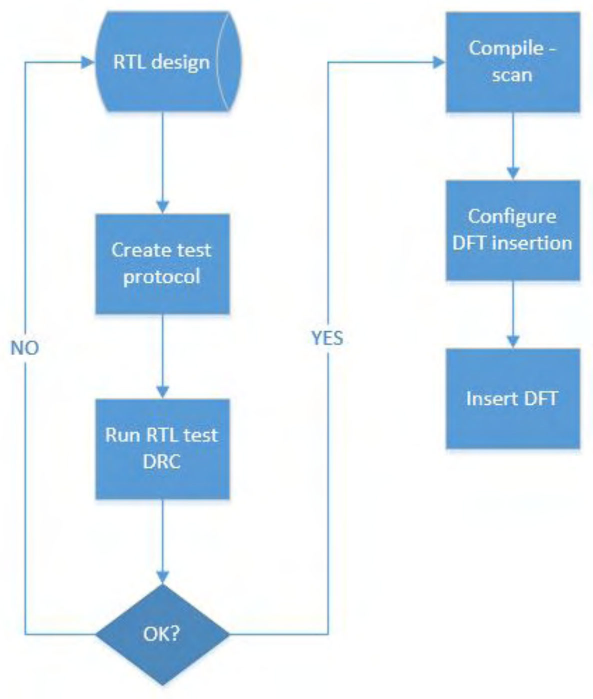


Figure 4.25: DFT insertion flow

4.5.1 Create test protocol

In order to insert scan chains to the design we have to define the test protocol. That means that we have to specify the signals that are going to be used in the scan chain. When specifying a DFT signal there are two options:

- use an existing signal as a DFT signal
- define a new one that Design Compiler will create automatically while performing the DFT insertion

In this thesis, we used the second option, although it was the more difficult one because it becomes difficult to attach an I/O cell to the new signal. Considering this problem, a further modification at the top cell that included pads (4.2.1). That modification included the import of the necessary signals for scan chain insertion at the top level RTL. In the multiplexed flip-flop style, that is used here, the required signals are the following:

- **Clock:** The AHB-lite slave clock used here
- **Reset:** The AHB-lite slave reset used here
- **Scan data input:** new pad created for this signal

- **Scan data output:** new pad created for this signal
- **Scan enable:** new pad created for this signal

The above signals are specified in the .tcl file by the commands:

set_dft_signal -view existing_dft -type ScanClock -port [get_ports PI_clk] -timing 45 55
set_dft_signal -view existing_dft -type reset -port [get_ports PI_reset] -active_state 1
set_dft_signal -view spec -type ScanEnable -port [get_ports PI_Scan_enable] -active_state 1
set_dft_signal -view spec -type ScanDataIn -port [get_ports PI_Scan_In]
set_dft_signal -view spec -type ScanDataOut -port [get_ports PO_Scan_Out]

Finally, the test protocol has to be created:

create_test_protocol

Once the signals and the elements that are going to be used are correctly specified in the corresponding test protocol, we can proceed with the next step.

4.5.2 Preview DFT and Scan Chain Synthesis

After creating the test protocol, we preview the scan chain with *preview_dft* coomand that checks specification for consistency. Afterwards, we want to check the design for any Design Rule Check (DRC) violations. The DRC check can be invoked with:

dft_drc

When the process finishes , the ”Violation Browser” window appears showing the detected violations. If any violations appear we have to eliminate them. Otherwise we proceed to the final step.

4.5.3 Compile, Configure and insert DFT

Once the violations have been corrected (or ignore if possible), the design must be compile including the ”-scan” option among the compile parameters. This option tells Design Compiler to replace the elements being part of the DFT for scannable elements. After that, the last step before inserting the scan chain is to configure the way that it is going to be inserted. There are many options before inserting the scan chain. The most relevant for our design it the three-state buffers and bidirectional ports: By default, DFT Compiler infers logic to bring all the three-state buffers and bidirectional ports into ’Z’ state. This behavior is not necessary in our design, and consequently, has to be deactivated:

set dft configuration -fix disable -fix bidirectional disable

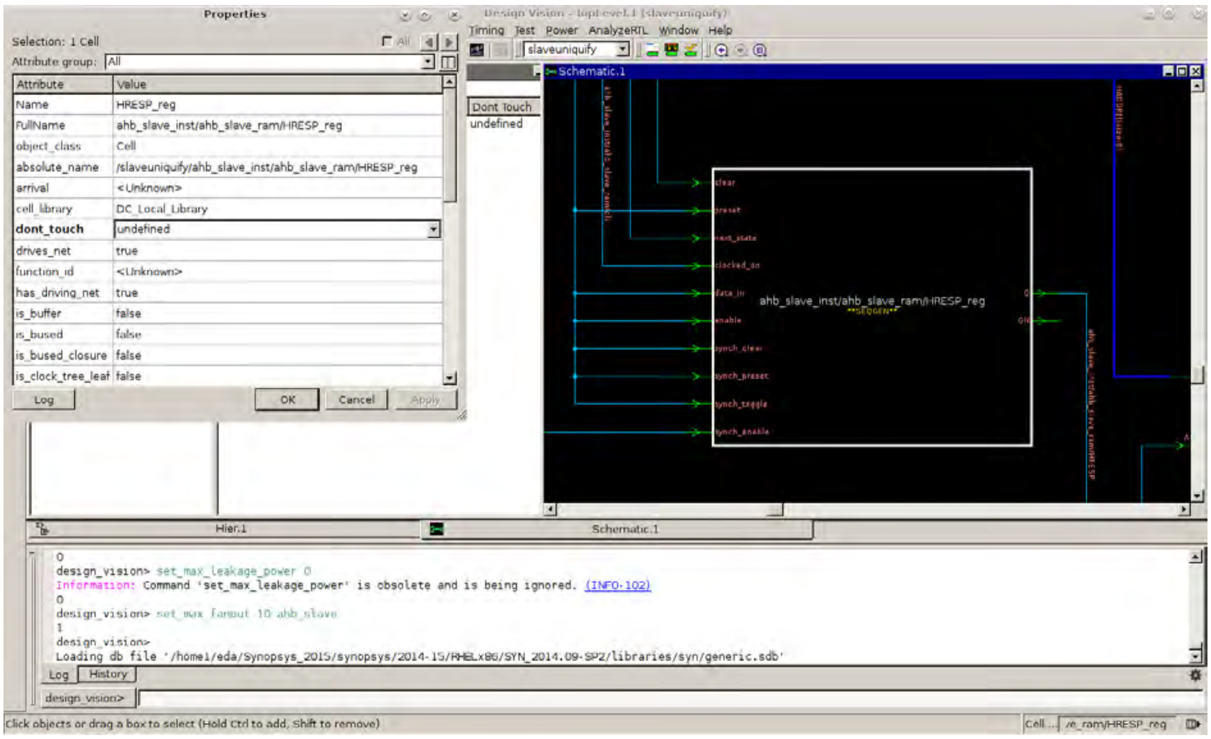
Finally, the insertion of the scan chain is executed with:

insert_dft

A comparison of a register vs a scan register is shown in the figure 4.26

4.6 Design Finishing

After the final reports and the netlist produced from synthesis flow from Design Compiler, we are ready to proceed to the test vectors extraction. **Only** if our design is error free we can proceed further. Otherwise, we must re-run Design Compiler with changing the constraints we set in order to achieve the results we want considering time, power, area.

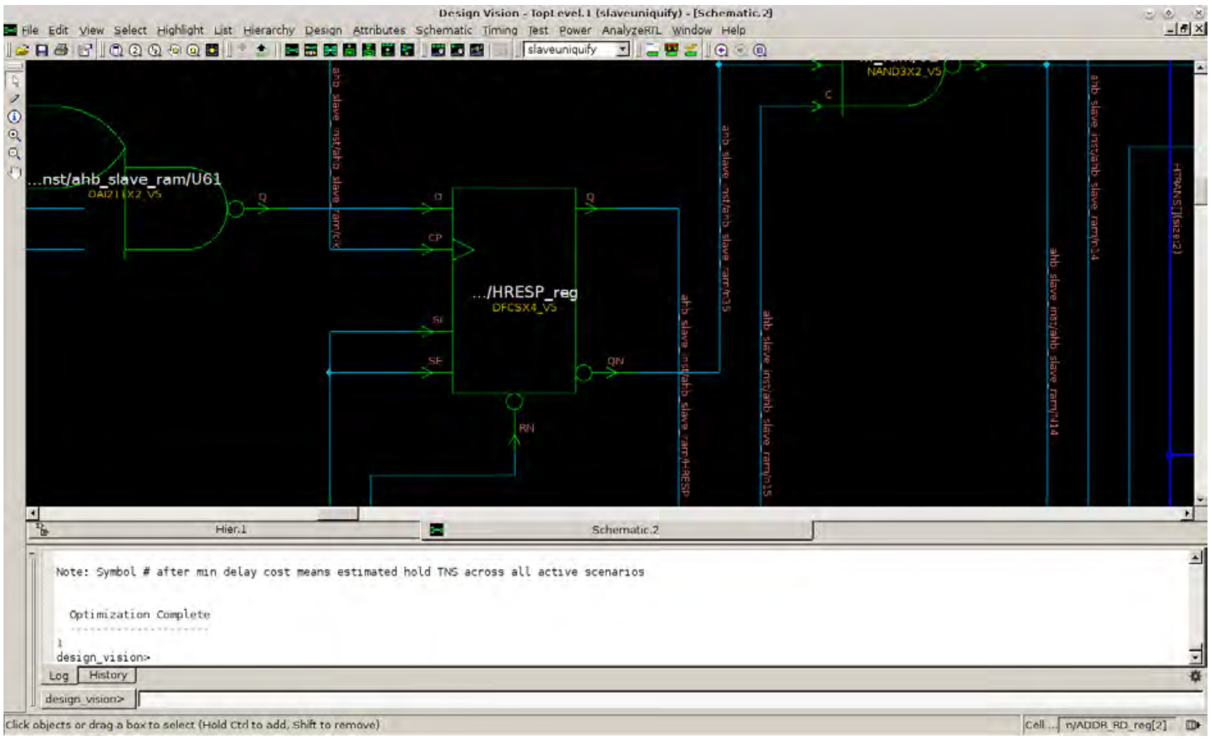


Attribute	Value
Name	HRESP_reg
FullName	ahb_slave_inst/ahb_slave_ram/HRESP_reg
object_class	Cell
absolute_name	/slaveuniquify/ahb_slave_inst/ahb_slave_ram/HRESP_reg
arrival	<Unknown>
cell_library	DC_Local_Library
dont_touch	undefined
drives_net	true
function_id	<Unknown>
has_driving_net	true
is_buffer	false
is_bused	false
is_bused_closure	false
is_clock_tree_leaf	false

```
0
design_vision> set_max_leakage_power 0
Information: Command 'set_max_leakage_power' is obsolete and is being ignored. (INFO-102)
0
design_vision> set_max_fanout 10 ahb_slave
1
design_vision>
Loading db file '/home1/eda/Synopsys_2015/synopsys/2014-15/RHELx86/SYN_2014.09-SP2/libraries/syn/generic.sdb'
```

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

(a) Normal Flip-Flop


Design Vision - TopLevel.1 (slaveuniquify) - [Schematic.2]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
slaveuniquify
NAND3X2_VS
C
...nst/ahb_slave_ram/U61
OAI211X2_VS
Q
D
CP
SI
SE
QN
RN
.../HRESP_reg
DFCSX4_VS
Hier.1
Schematic.2
Log
History
design_vision>
Cell ...
n/ADDR_RD_reg[2]

```
Note: Symbol # after min delay cost means estimated hold TNS across all active scenarios

  Optimization Complete
  ---------------------
1
design_vision>
```

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

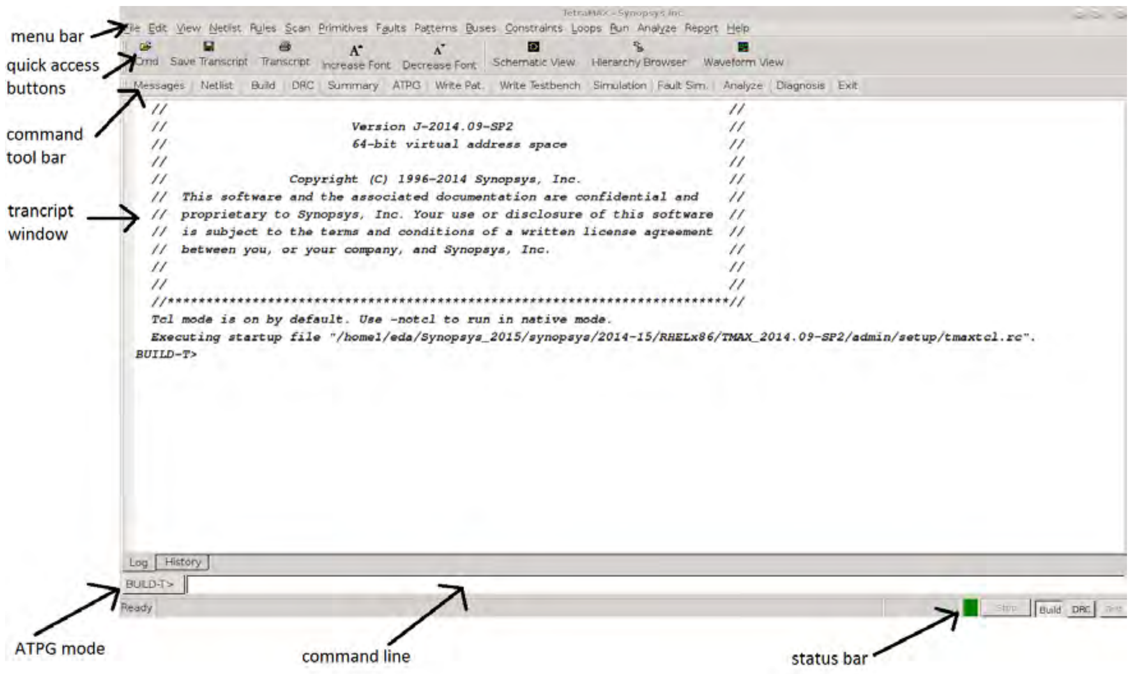
(b) Scan register

Figure 4.26: Register vs Scan Register

4.7 Test vectors extraction

According to [10] the goal of ATPG is to create a set of patterns that achieves a given test coverage, where test coverage is the total percentage of testable faults the patterns set actually defect. The ATPG run itself consist of two main steps: generating patterns and fault simulation to determine which faults the pattern detect. The two most typical methods for pattern generation are random and deterministic. Additionally, the ATPG tool can fault simulate patterns from an external set and place those patterns detecting fault in a test set. TetraMAX is a high-speed, high-capacity Automatic Test Pattern Generation (ATPG) tool. It can generate test patterns that maximize test coverage while using a minimum number of test vectors for a wide variety of design types and design flows. It is well suited for designs of all sizes up to millions of gates. With scan testing, the sequential elements of the device are connected into chains and used as primary inputs and primary outputs for testing purposes. Using ATPG techniques, you can test a much larger number of internal faults than with functional testing alone.

TetraMAX offers three different ATPG modes: Basic-Scan, Fast-Sequential, and Full-Sequential. TetraMAX supports test pattern generation for five types of fault models: stuck-at faults, IDDQ faults, transition delay faults, path delay faults, and bridging faults. It is also compatible with a wide range of design-for-test tools such as DFT Compiler. The design flow using DFT Compiler and TetraMAX ATPG is recommended for maximum ease of use and quality of results. TetraMax ATPG init screen is shown below at Figure 4.27


TetraMAX - Synopsys Inc.
File Edit View Netlist Rules Scan Primitives Faults Patterns Buses Constraints Loops Run Analyze Report Help
Cmd Save Transcript Transcript Increase Font Decrease Font Schematic View Hierarchy Browser Waveform View
Messages Netlist Build DRC Summary ATPG Write Pat. Write Testbench Simulation Fault Sim. Analyze Diagnosis Exit
menu bar
quick access buttons
command tool bar
trancript window
Log History
BUILD-T>
Ready
Stop Build DRC Test
ATPG mode
command line
status bar

```
 //                                                                      //
 //                        Version J-2014.09-SP2                         //
 //                        64-bit virtual address space                  //
 //                                                                      //
 //                 Copyright (C) 1996-2014 Synopsys, Inc.               //
 //  This software and the associated documentation are confidential and //
 //  proprietary to Synopsys, Inc. Your use or disclosure of this software //
 //  is subject to the terms and conditions of a written license agreement //
 //  between you, or your company, and Synopsys, Inc.                    //
 //                                                                      //
 //                                                                      //
 //**********************************************************************//
 Tcl mode is on by default. Use -notcl to run in native mode.
 Executing startup file "/home1/eda/Synopsys_2015/synopsys/2014-15/RHELx86/TMAX_2014.09-SP2/admin/setup/tmaxtcl.rc".
BUILD-T>
```

Figure 4.27: Synopsys TetraMax ATPG Init Screen

Scan inserted netlist, is passed to TetraMAX as inputs. Second step is to build internal database for pattern generation. Specific TetraMAX libraries and netlist are combined and for building module definitions. BUILD process takes place in build mode of TetraMAX. And after completion of successful build process SPF file, which contains scan definitions are provided to check DRC. Several rule violations are check during DRC mode like clock, verilog syntax, scan chain, set/reset etc. and must be cleared before pattern generation. After clean DRC mode it automatically transfer mode to TEST mode. Now at last several efforts as per specification are selected and applied for pattern generation. Selecting fault model and test mode, ATPG process starts. Patterns are generated and stored in different formats like STIL, WGL, Verilog, VHDL, binary and other foundry formats.

Quick estimation of Test and Fault coverage are done after patterns are generated using saved database and results in terms of reports and summaries. Three possible quality measures are defined as follows: *Test coverage = detected fault / detectable faults.* Test coverage gives the most meaningful measure of test pattern quality and is the default coverage reported in the fault summary report. Test coverage is defined as the percentage of detected faults out of detectable faults. *Fault coverage = detected fault / all faults.* Fault coverage is defined as the percentage of detected faults out of all faults. *ATPG effectiveness = ATPG resolvable fault / all faults.* ATPG effectiveness is defined as the percentage of ATPG-resolvable faults out of the total faults. Report summaries are generated in two forms collapsed and uncollapsed fault summaries. Following results are of stuck – at fault model with basic mode and all desired manual efforts. Design reports will be shown at chapter 8.

TetraMAX maintains a list of potential faults in the design and assigns each such fault to a fault class according to its detectability status. Faults classes are organized into categories. A two-character symbol is used as an abbreviated name for both classes and categories. There are five higher-level fault categories containing a total of 11 lower-level fault classes:

1. DT: detected
   - (a) DR: detected robustly
   - (b) DS: detected by simulation
   - (c) DI: detected by implication
2. PT: possibly detected
   - (a) AP: ATPG untestable-possibly detected
   - (b) NP: not analyzed-possibly detected
3. UD: undetectable
   - (a) UU: undetectable unused
   - (b) UT: undetectable tied
   - (c) UB: undetectable blocked
   - (d) UR: undetectable redundant
4. AU: ATPG untestable
   - (a) AN: ATPG untestable-not detected
5. ND: not detected
   - (a) NC: not controlled
   - (b) NO: not observed

TetraMAX can generate patterns in verilog (table/single), verilog (parallel/serial), VHDL, STIL/WGL, Binary and several other formats which will be used after the fabrication process to test the fabricated design for faults.

4.8 Logic Equivalence Checking

Formal Verification verifies the circuit without changing the logic of the circuit. It is the defacto standard used today in the industry. Some of the tools which use formal verification today in the industry are Conformal (from cadence), Formality (from Synopsys). Formal verification is an algorithmic-based approach to logic verification that exhaustively proves functional properties about a design. The algorithms formal verification uses are:

- Binary Decision Diagram (BDD): It is a compact data structure of Boolean logic. It can represent the logic state encoded as a Boolean function.
- Symbolic Finite State Machine (FSM) Traversal

Typically, the following are the types of formal verification:

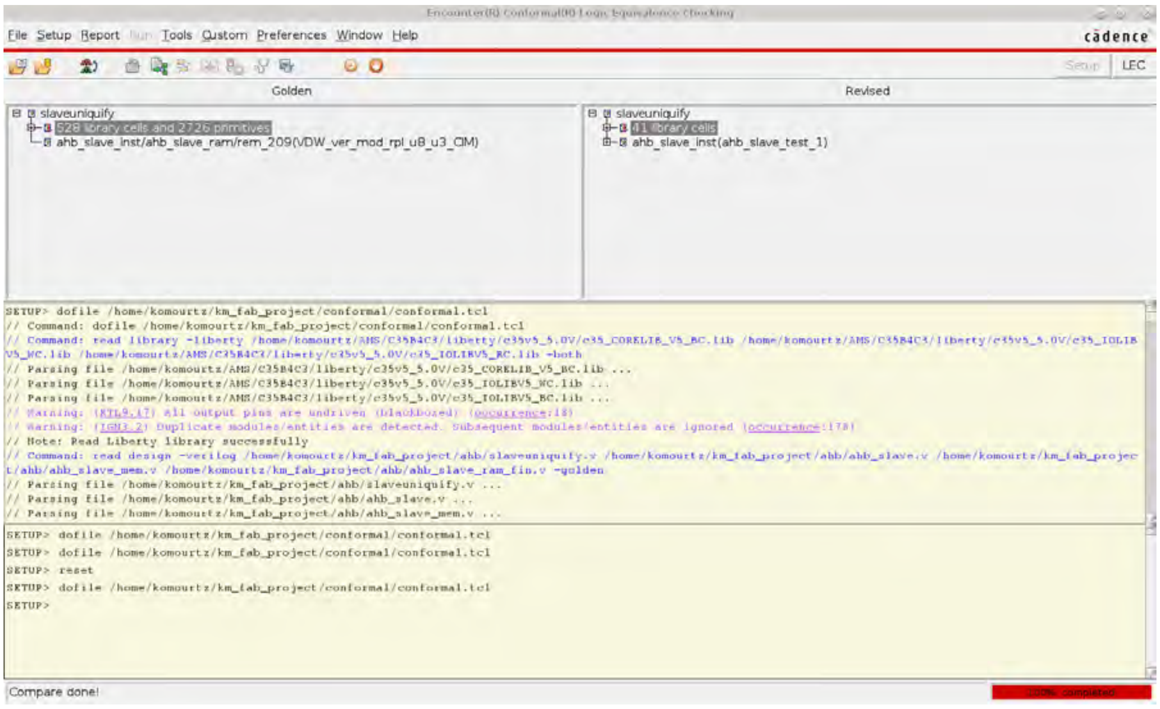
- **Equivalence Checking:** Verifies the functional equivalence of two designs that are at the same or different abstraction levels (e.g., RTL-to-RTL, RTL-to-Gate, or Gate-toGate). It checks combinatorial and sequential elements. (Basically checks if two circuits are equivalent). For sequential elements, it checks if the specific instance name occurs in both the circuits or not.
- **Model Checking:** Verifies that the implementation satisfies the properties of the design. Model checking is used early in the design creation phase to uncover functional bugs.

Here we use the Equivalence checking using Conformal from Cadence. Conformal LEC (Figure) is a powerful equivalence checking tool. It can provide a formal proof that the output from Synthesis matches the original RTL code. It can do all of that without having to run a single simulation. In this section of the tutorial, we will learn how to read in a RTL and a synthesized design and how to prove that they are functionally equivalent. The original RTL netlist is usually referred to as the "golden" design. It serves as the reference for the comparison. The synthesized gate-level netlist is also called the "Revised" design. The following files are required for running the equivalence checking:

- RTL files (golden design) (.v)
- Netlist exported from Design Compiler (revised design) (.v)
- The primitive cells library (.lib)
- Script (.tcl) optional

Since both the designs have been successfully loaded, we can now start the verification process. Conformal has 2 operating modes, the "Setup" and the "LEC" mode. Switch to the LEC mode by Clicking on the "LEC" icon in the upper right hand corner of the window. A table is now printed in the conformal LEC window. It lists the primary inputs (PI) and primary outputs (PO) in both the revised and golden designs. They are equal if the golden and revised designs have the same number of inputs and outputs. After comparing the two designs the equivalence checker reduces the two designs into canonical representations and then checks to see if they are equal.

In our design we had non equivalences at all registers. That happened because we changed the initial RTL code and design by changing the registers into scan registers. A rename command needed in order to solve this problem. After the renaming command the design netlist had no non equivalences and it was ready to import into Cadence Encounter for exporting the final layout.


Encounter(R) Conformal(R) Logic Equivalence Checking
File Setup Report Run Tools Custom Preferences Window Help
cādence
Setup
LEC
Golden
Revised
slaveuniquify
528 library cells and 2726 primitives
ahb_slave_inst/ahb_slave_ram/rem_209(VDW_ver_mod_rpl_u8_u3_CIM)
slaveuniquify
61 library cells
ahb_slave_inst(ahb_slave_test_1)
Compare done!
100% completed

```
SETUP> dofile /home/komourtz/km_fab_project/conformal/conformal.tcl
// Command: dofile /home/komourtz/km_fab_project/conformal/conformal.tcl
// Command: read library -liberty /home/komourtz/AMS/C35B4C3/liberty/c35v5_5.0V/c35_CORELIB_V5_BC.lib /home/komourtz/AMS/C35B4C3/liberty/c35v5_5.0V/c35_IOLIB
V5_WC.lib /home/komourtz/AMS/C35B4C3/liberty/c35v5_5.0V/c35_IOLIBV5_BC.lib -both
// Parsing file /home/komourtz/AMS/C35B4C3/liberty/c35v5_5.0V/c35_CORELIB_V5_BC.lib ...
// Parsing file /home/komourtz/AMS/C35B4C3/liberty/c35v5_5.0V/c35_IOLIBV5_WC.lib ...
// Parsing file /home/komourtz/AMS/C35B4C3/liberty/c35v5_5.0V/c35_IOLIBV5_BC.lib ...
// Warning: (RTL9.17) All output pins are undriven (blackboxed) (occurrence:18)
// Warning: (IGN3.2) Duplicate modules/entities are detected. Subsequent modules/entities are ignored (occurrence:178)
// Note: Read Liberty library successfully
// Command: read design -verilog /home/komourtz/km_fab_project/ahb/slaveuniquify.v /home/komourtz/km_fab_project/ahb/ahb_slave.v /home/komourtz/km_fab_projec
t/ahb/ahb_slave_mem.v /home/komourtz/km_fab_project/ahb/ahb_slave_ram_fin.v -golden
// Parsing file /home/komourtz/km_fab_project/ahb/slaveuniquify.v ...
// Parsing file /home/komourtz/km_fab_project/ahb/ahb_slave.v ...
// Parsing file /home/komourtz/km_fab_project/ahb/ahb_slave_mem.v ...
```

```
SETUP> dofile /home/komourtz/km_fab_project/conformal/conformal.tcl
SETUP> dofile /home/komourtz/km_fab_project/conformal/conformal.tcl
SETUP> reset
SETUP> dofile /home/komourtz/km_fab_project/conformal/conformal.tcl
SETUP>
```

Figure 4.28: Cadence Conformal GUI mode

Chapter 5

Back End Flow

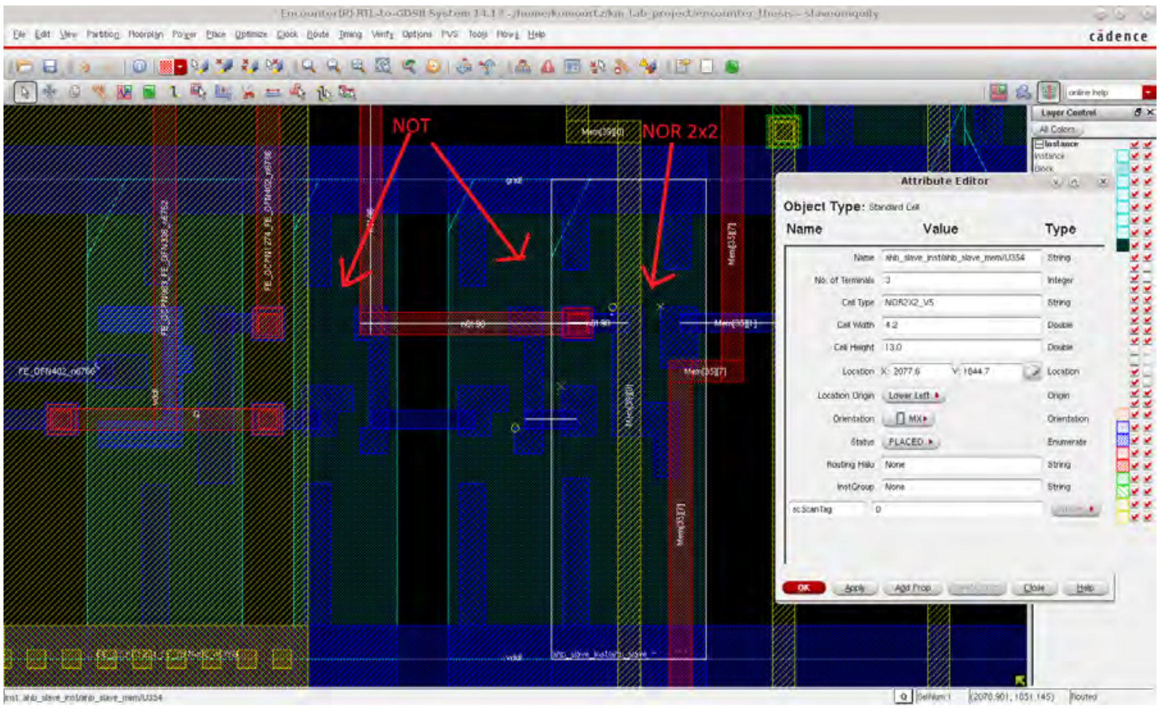
5.1 Initial steps

At the previous chapter (chapter 4) we finished the Front-End flow of our Design and we now have a clean verilog netlist. Proceeding, we have to ensure that some modules should have unique names and they may not. We have to fix that problem by using an Encounter command to make the instances unique:

uniquifyNetlist -top slaveuniquify chip_uniq.v AMS_35b4c3_dft.v

takes the netlist exported from Design Compiler (AMS_35b4c3_dft.v), the top level module name (slaveuniquify) and exports the uniquified netlist (chip_uniq.v).

Physical placement of a design requires the layout and timing information of the standard cells. Constraint driven placement may be utilized with constraints generated during RTL synthesis. Layout and timing information is provided from the technology vendors. The former is provided by the Library Exchange Format (LEF) file. The information in the LEF file is the text version of the Virtuoso standard cell view abstract and includes layer names, layer widths, layer usage, external dimensions, cell pin and port as well as blockage description. An example of how standard cells abstract is and how they connect with each other is shown in figure 5.1, where we see the interconnection of two NOT gates with one NOR gate 2x2 (2-inputs with driving strength 2). Timing information, is provided by the technology libraries (.lib), which are available as worst-corner and best-corner cases to model process variations. The constraints set during synthesis may be transferred to Encounter by a Synopsys Design Constraints (SDC) file. The file is Tool Command Language (TCL) based and specifies the design intent, including the timing, power, and area constraints. With the worst-case and best-case libraries we create the corners we want the tool to place, route and optimize our design. This is the Multi Mode Multi Corner (MMMC) as it called and it combines the .lib, .lef, .sdc, .cap (capacitance tables) to create the analysis views : worst with the worst-case combination of these files for the setup analysis view and fast with the best-case combination of these files for the hold analysis mode. The gate-level netlist of the design which was generated during RTL synthesis needs to be provided in verilog format. A waveform of the SoC Encounter initial gui is shown in the figure 5.2



Object Type: Standard Cell

Name	Value	Type
Name	ahb_slave_inst/ahb_slave_mem/U334	String
No. of Terminals	3	Integer
Cell Type	NOR2X2_V5	String
Cell Width	4.2	Double
Cell Height	13.0	Double
Location	X: 2077.6 Y: 1044.7	Location
Location Origin	Lower Left	Origin
Orientation	MX	Orientation
Status	PLACED	Enumerate
Routing Halo	None	String
InstGroup	None	String
sc:ScanTag	0	

SelNum:1 (2070.901, 1051.145) Routed

Figure 5.1: Standard cell interconnection

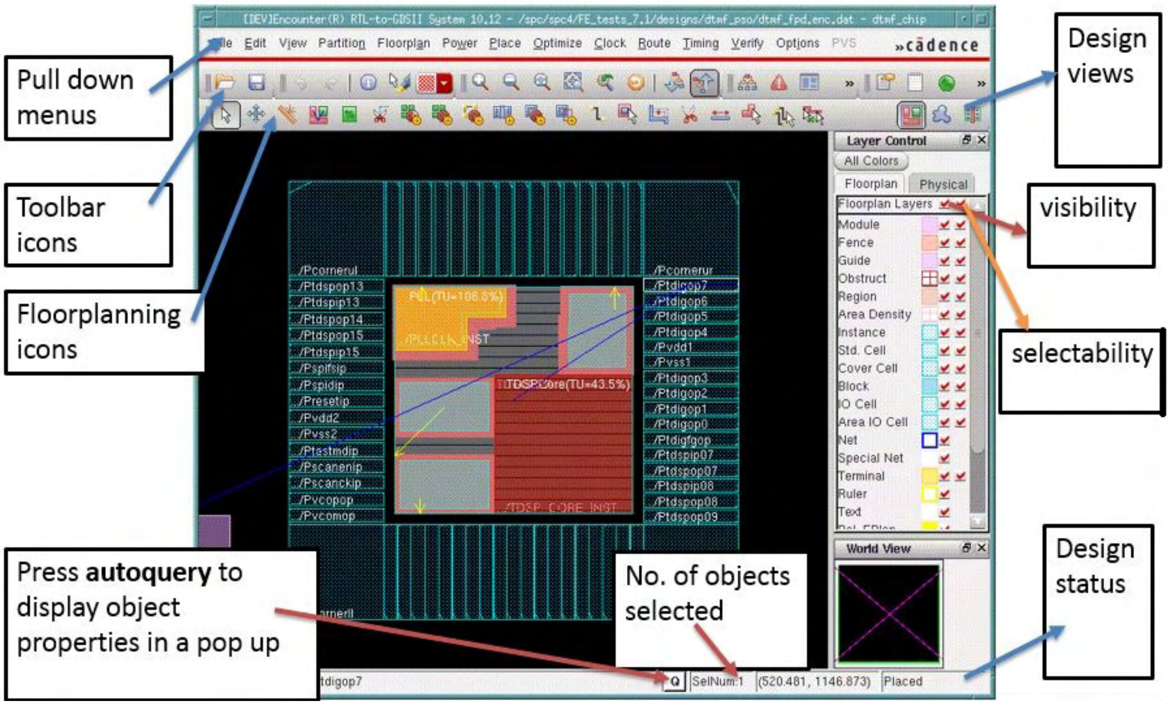


Figure 5.2: SoC EDI initial GUI

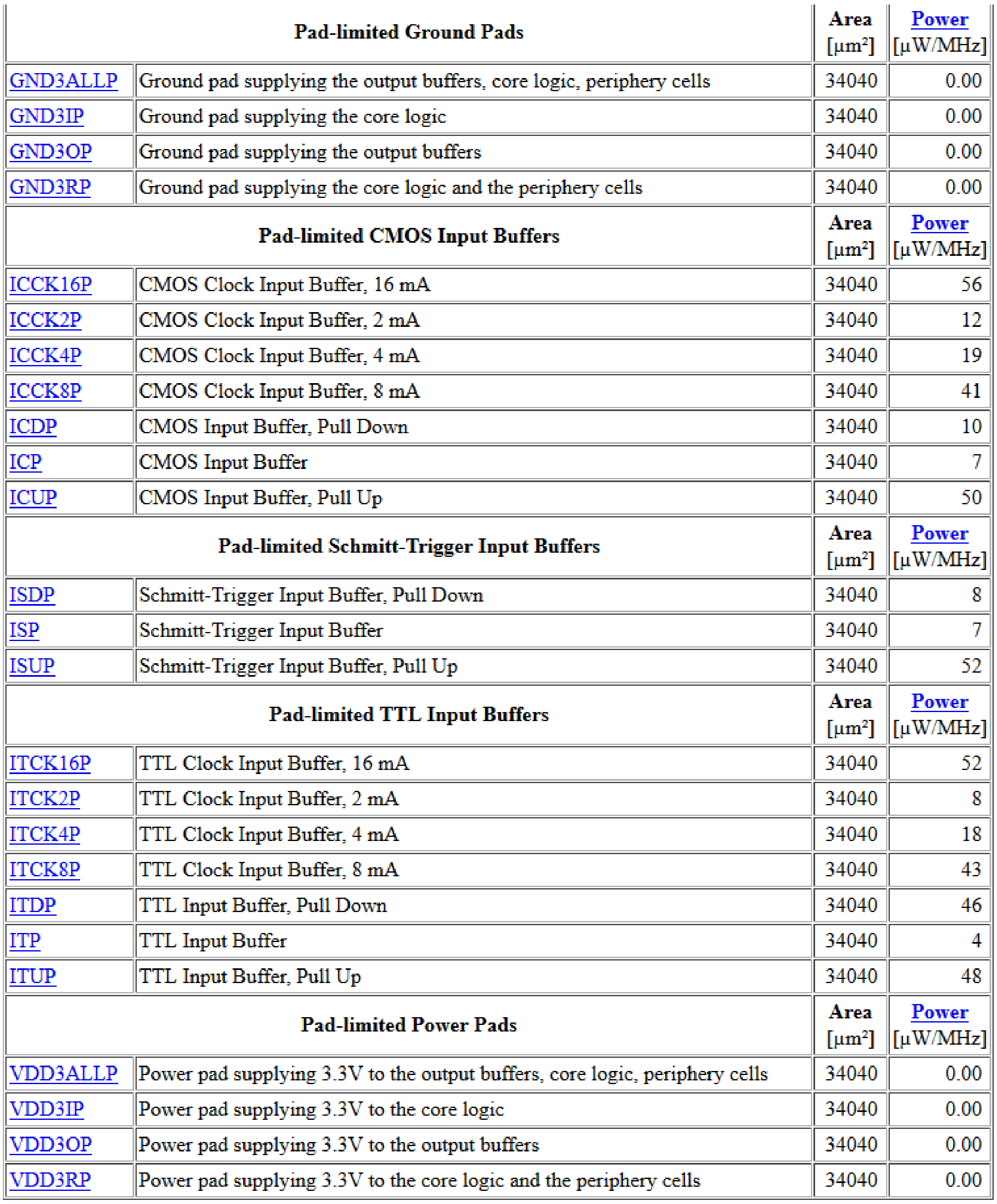
5.1.1 I/O File

The physical location as well as the orientation of the IO's need to be specified in a IO assignment file. Beside the IO's which are already inferred with the RTL model, IO's for core and pad power. So, we have to create a .ioc file to obtain all the pads that transfer signals (basically exported from Design Compiler) and the new power pads that the design needs. We also place the corner pads to the corners of the design. In the .ioc file we name the side of the pad that we want to be placed (E for east, W for west, N for north and S for south) as well as the pad name.

After that we need to invoke the tool (initial screen shown in figure 5.2) and we have to import our design. We had to choose between many IO pad cells, so we had to review all of them and choose the right ones for power pads, signal pads as well as clock and reset pads. Figure 5.3 shows an example of some of these pad cells with 3.3 Voltage and figure 5.4 shows an example of these pad cells with a voltage of 5.

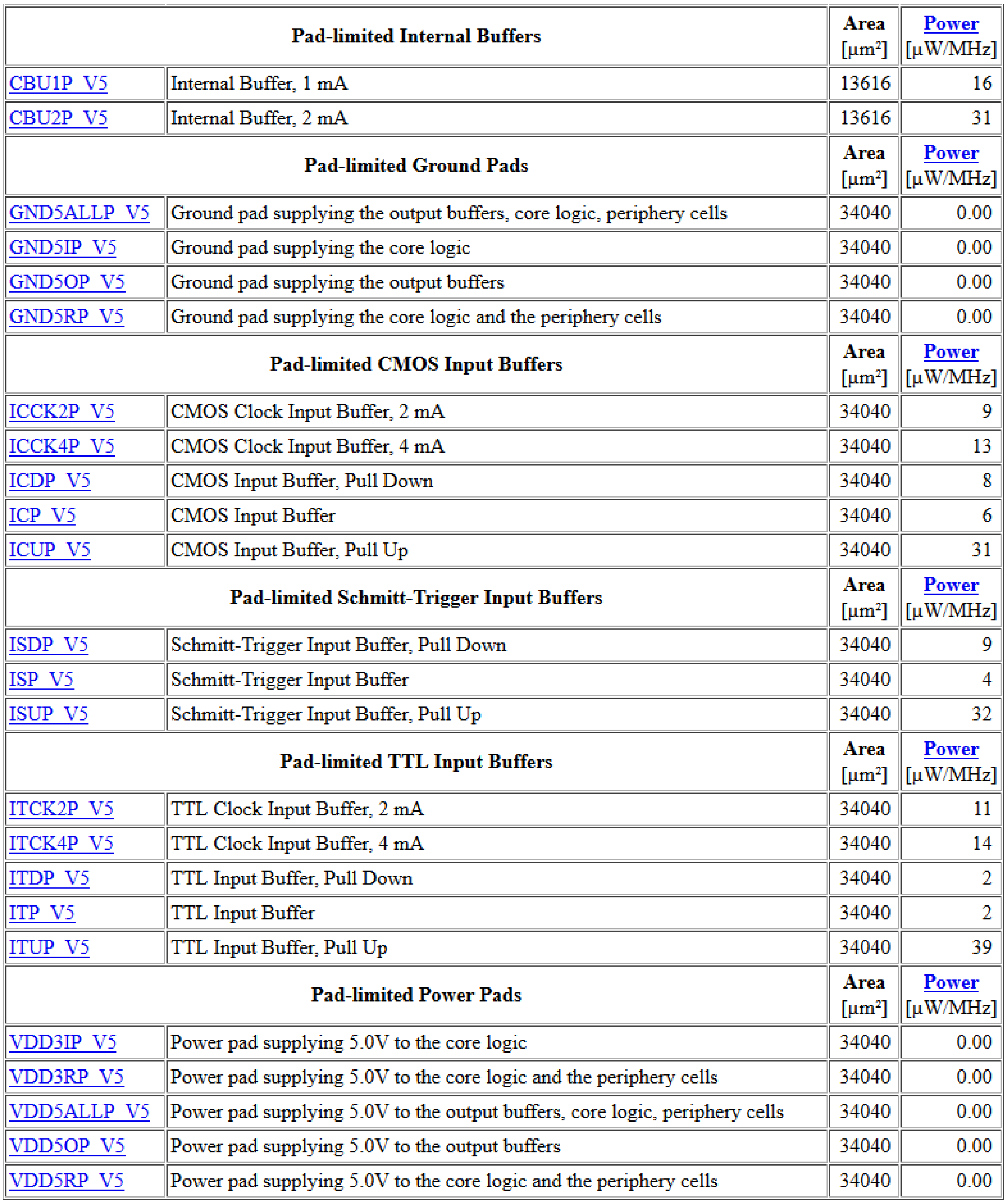
The figure 5.5 shows us a vertical cut of a wafer and how the metals and the transistors are represented.

The figure 5.6 shows us the power buses for the pad limited c35 IO cells. The LV IO-cells have two pairs of power buses: The ESD-Rails and the power supply for the periphery-ring. The ESD-Rails have names which end with an ”o” (e.g. vdd3o! and gnd3o!). The power supply for the periphery ring has to use the letter ”r” (e.g. vdd3r1! or gnd3r!). The power bus vdd3r! is split into two physical buses vdd3r1! and vdd3r2!, these two buses are connected together in the VDD pads. On 3 bus cells or on HV cells additional vsub! buses can be found on the cells.



Pad-limited Ground Pads		Area [µm²]	Power [µW/MHz]
GND3ALLP	Ground pad supplying the output buffers, core logic, periphery cells	34040	0.00
GND3IP	Ground pad supplying the core logic	34040	0.00
GND3OP	Ground pad supplying the output buffers	34040	0.00
GND3RP	Ground pad supplying the core logic and the periphery cells	34040	0.00
Pad-limited CMOS Input Buffers		**Area** [µm²]	**Power** [µW/MHz]
ICCK16P	CMOS Clock Input Buffer, 16 mA	34040	56
ICCK2P	CMOS Clock Input Buffer, 2 mA	34040	12
ICCK4P	CMOS Clock Input Buffer, 4 mA	34040	19
ICCK8P	CMOS Clock Input Buffer, 8 mA	34040	41
ICDP	CMOS Input Buffer, Pull Down	34040	10
ICP	CMOS Input Buffer	34040	7
ICUP	CMOS Input Buffer, Pull Up	34040	50
Pad-limited Schmitt-Trigger Input Buffers		**Area** [µm²]	**Power** [µW/MHz]
ISDP	Schmitt-Trigger Input Buffer, Pull Down	34040	8
ISP	Schmitt-Trigger Input Buffer	34040	7
ISUP	Schmitt-Trigger Input Buffer, Pull Up	34040	52
Pad-limited TTL Input Buffers		**Area** [µm²]	**Power** [µW/MHz]
ITCK16P	TTL Clock Input Buffer, 16 mA	34040	52
ITCK2P	TTL Clock Input Buffer, 2 mA	34040	8
ITCK4P	TTL Clock Input Buffer, 4 mA	34040	18
ITCK8P	TTL Clock Input Buffer, 8 mA	34040	43
ITDP	TTL Input Buffer, Pull Down	34040	46
ITP	TTL Input Buffer	34040	4
ITUP	TTL Input Buffer, Pull Up	34040	48
Pad-limited Power Pads		**Area** [µm²]	**Power** [µW/MHz]
VDD3ALLP	Power pad supplying 3.3V to the output buffers, core logic, periphery cells	34040	0.00
VDD3IP	Power pad supplying 3.3V to the core logic	34040	0.00
VDD3OP	Power pad supplying 3.3V to the output buffers	34040	0.00
VDD3RP	Power pad supplying 3.3V to the core logic and the periphery cells	34040	0.00

Figure 5.3: c35 IOLIB Cells 3.3V



Pad-limited Internal Buffers		Area [µm²]	Power [µW/MHz]
CBU1P_V5	Internal Buffer, 1 mA	13616	16
CBU2P_V5	Internal Buffer, 2 mA	13616	31
Pad-limited Ground Pads		**Area** [µm²]	**Power** [µW/MHz]
GND5ALLP_V5	Ground pad supplying the output buffers, core logic, periphery cells	34040	0.00
GND5IP_V5	Ground pad supplying the core logic	34040	0.00
GND5OP_V5	Ground pad supplying the output buffers	34040	0.00
GND5RP_V5	Ground pad supplying the core logic and the periphery cells	34040	0.00
Pad-limited CMOS Input Buffers		**Area** [µm²]	**Power** [µW/MHz]
ICCK2P_V5	CMOS Clock Input Buffer, 2 mA	34040	9
ICCK4P_V5	CMOS Clock Input Buffer, 4 mA	34040	13
ICDP_V5	CMOS Input Buffer, Pull Down	34040	8
ICP_V5	CMOS Input Buffer	34040	6
ICUP_V5	CMOS Input Buffer, Pull Up	34040	31
Pad-limited Schmitt-Trigger Input Buffers		**Area** [µm²]	**Power** [µW/MHz]
ISDP_V5	Schmitt-Trigger Input Buffer, Pull Down	34040	9
ISP_V5	Schmitt-Trigger Input Buffer	34040	4
ISUP_V5	Schmitt-Trigger Input Buffer, Pull Up	34040	32
Pad-limited TTL Input Buffers		**Area** [µm²]	**Power** [µW/MHz]
ITCK2P_V5	TTL Clock Input Buffer, 2 mA	34040	11
ITCK4P_V5	TTL Clock Input Buffer, 4 mA	34040	14
ITDP_V5	TTL Input Buffer, Pull Down	34040	2
ITP_V5	TTL Input Buffer	34040	2
ITUP_V5	TTL Input Buffer, Pull Up	34040	39
Pad-limited Power Pads		**Area** [µm²]	**Power** [µW/MHz]
VDD3IP_V5	Power pad supplying 5.0V to the core logic	34040	0.00
VDD3RP_V5	Power pad supplying 5.0V to the core logic and the periphery cells	34040	0.00
VDD5ALLP_V5	Power pad supplying 5.0V to the output buffers, core logic, periphery cells	34040	0.00
VDD5OP_V5	Power pad supplying 5.0V to the output buffers	34040	0.00
VDD5RP_V5	Power pad supplying 5.0V to the core logic and the periphery cells	34040	0.00

Figure 5.4: c35 IOLIBV5 Cells 5.0V

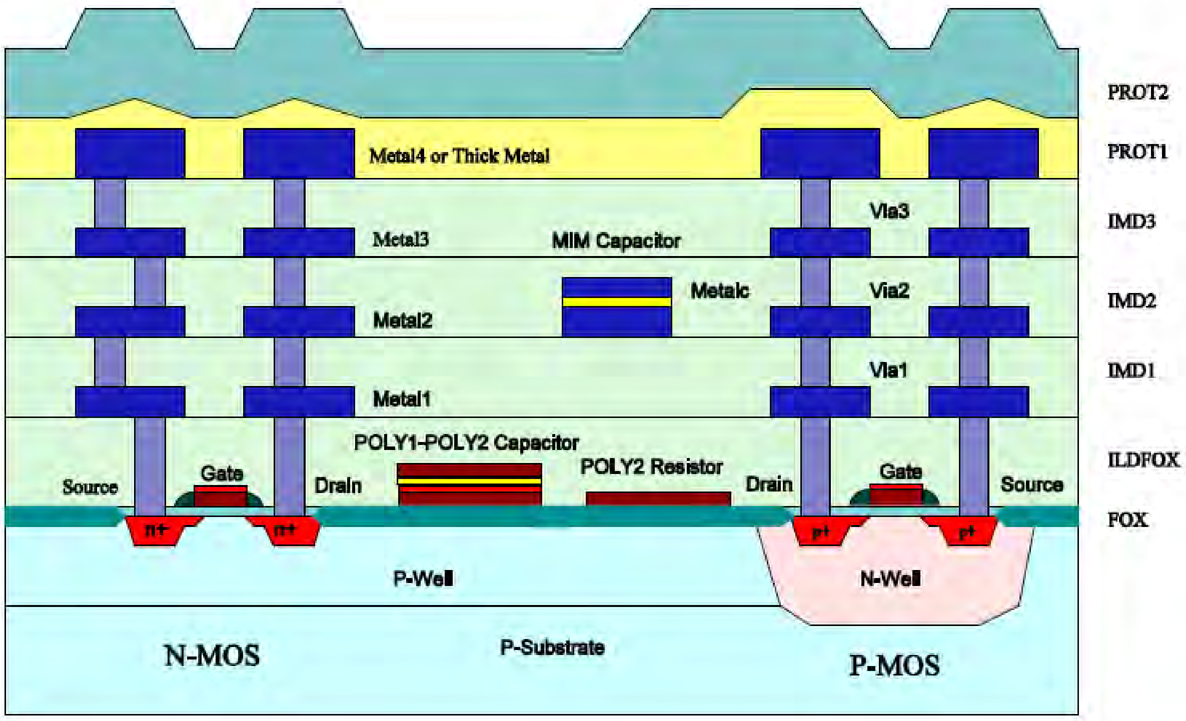


Figure 5.5: Wafer Cross

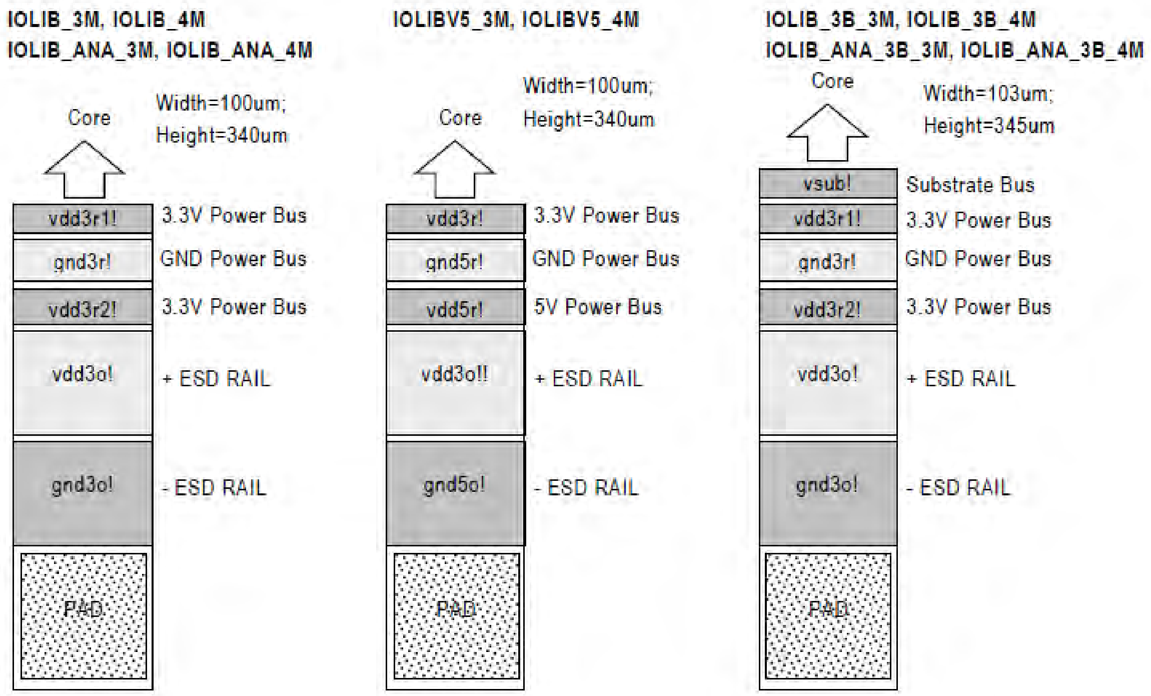


Figure 5.6: Power Buses for the pad limited C35 IO cells

5.1.2 Design Import

Before importing the design we have to ensure that we have selected the technology we work on (in our case is 350nm). So we use the command:

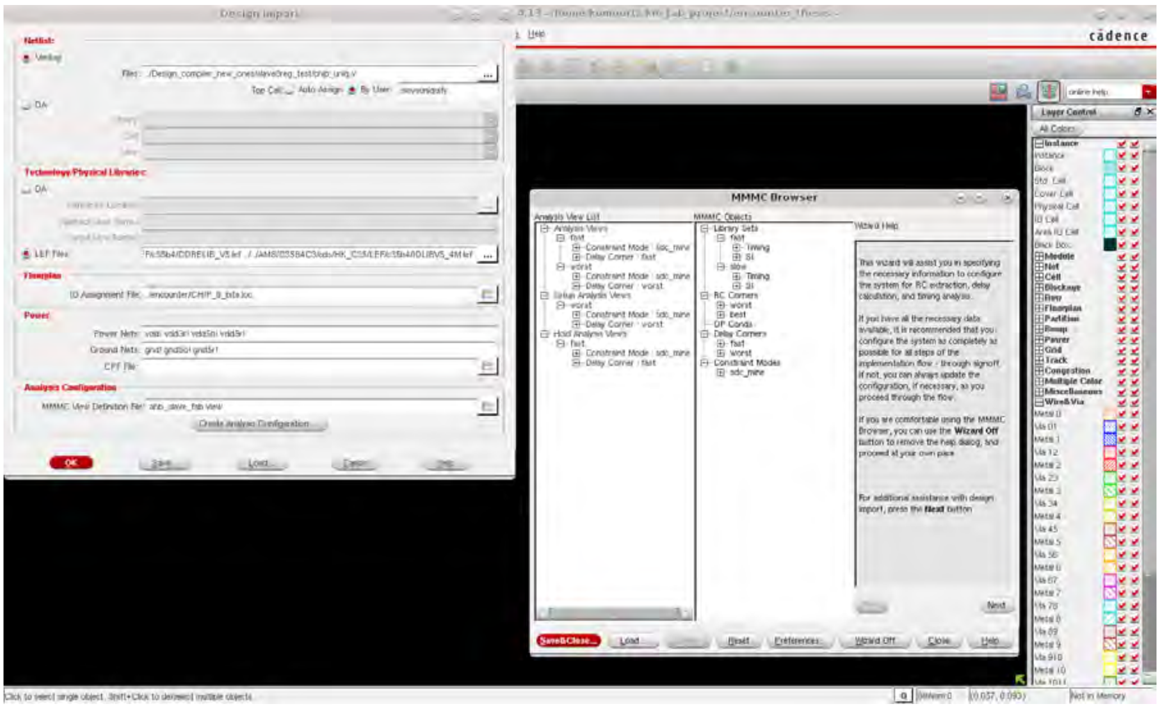
setDesignMode -process 250 -flowEffort high

(Because the newest version of SoC Ecnounter runs with technology up to 250nm). Furthermore, we have to specify the scan chain in order the tool not try to reorder or "change" the scan chain. To achieve that we use the command:

specifyScanChain scan1 -start PAD_scan_in/Y -stop PAD_scan_out/A scanTrace

where we specify the imput pin and the output pin of the scan chain and we tell the tool to locate this scan chain.

The technology and design files need to be read by Encounter to generate a database. The technology files that accommodate physical and timing information are accessed by selecting files with the extension **.lef** and **.lib** The required .lib files model worst-case (wc) and best-case (bc) process corners by max-timing and min-timing libraries, respectively. This is valid for the standard cells and IO's, as well as for any hard macro model. The design files consist of the gate-level netlist, timing constraint file, and pad arrangement, having the endings **.v .sdc .ioc**. We also have to specify the VDD and GND (or the way they called at the .lef file from all pads and standard cells) for Power and Ground nets, respectively. A figure of Design import is shown below (Figure 5.7)



Design Import

Netlist:
Files: ./Design_compiler_new_ones/slave0reg_test/chip_uniq.v
Top Cell: By User: [illegible]

LEF Files: [illegible]COREL!B_V5.lef ../../AMS/[illegible]

IO Assignment File: ../encounter/CHIP_8_[illegible].ioc

Power Nets: vdd! vdd3r! vdd5ri! vdd3r!
Ground Nets: gnd! gnd5o! gnd3r!

MMMC View Definition File: ahb_slave_[illegible].view

Wizard Help

This wizard will assist you in specifying the necessary information to configure the system for RC extraction, delay calculation, and timing analysis.

If you have all the necessary data available, it is recommended that you configure the system as completely as possible for all steps of the implementation flow - through signoff. If not, you can always update the configuration, if necessary, as you proceed through the flow.

If you are comfortable using the MMMC Browser, you can use the **Wizard Off** button to remove the help dialog, and proceed at your own pace.

For additional assistance with design import, press the **Next** button.

Figure 5.7: Design Import

In this design we use the c35_CORELIB and c35_IOLIB as library sets. This libraries as we saw at section 4.4.2.1 its cells has a smaller dynamic power consumption but bigger leakage power comparing with the 5 Volts lib that has small leakage and high dynamic power consumption.

After importing the design we can see at the .log file that the results are extracted that we have:

- Read 248 cells in library 'c35_CORELIB_WC' which is the library for the standard cells of our design.
- Read 181 cells in library 'c35_IOLIB_WC' which is the library for the IO pads of our design.

5.2 Floorplan

After importing our design we have to specify the dimensions of the SoC core, arrangement of the core rows, distance between SoC core and IO's, physical location of any hard macros and distance between these blocks and the core rows. The floorplan can be either core- or pad limited dependent on the size of the design and the number of pads. The space between the core and pad frame need to be specified to make space for the power supplies. In order to improve timing the gates should be placed as dense as possible on the core rows. This density is indicated by the core utilization. By specifying the Core to IO Boundary on each side, i.e., left, top, right, bottom, a gap between the core and IOs will be introduced, and the Core Utilization is updated. It is recommended to increase both Core to IO Boundary and the value for the desired core utilization until Core Utilization reaches a value around 0.8. Thereby, 20% of the core row space will be reserved for buffers in the clock tree (to be created later) and signal routing. The settings under the Advanced tab specify arrangement of the core rows and should remain unchanged. The IO's need to be placed in a certain distance to avoid a design rule violation. Furthermore, it is necessary to specify the global net connections, e.g., VCC and GND, need to be specified. The floorplan created with dimensions of: 2600 um x 2600 um, that means a 6.76mm^2 at die area. That means that the final area of the SoC will be at 6.76mm^2 .

A figure after floorplaning our design is shown below (fig 5.8). Also, a schematic of the floorplan dimensions is shown at figure 5.9

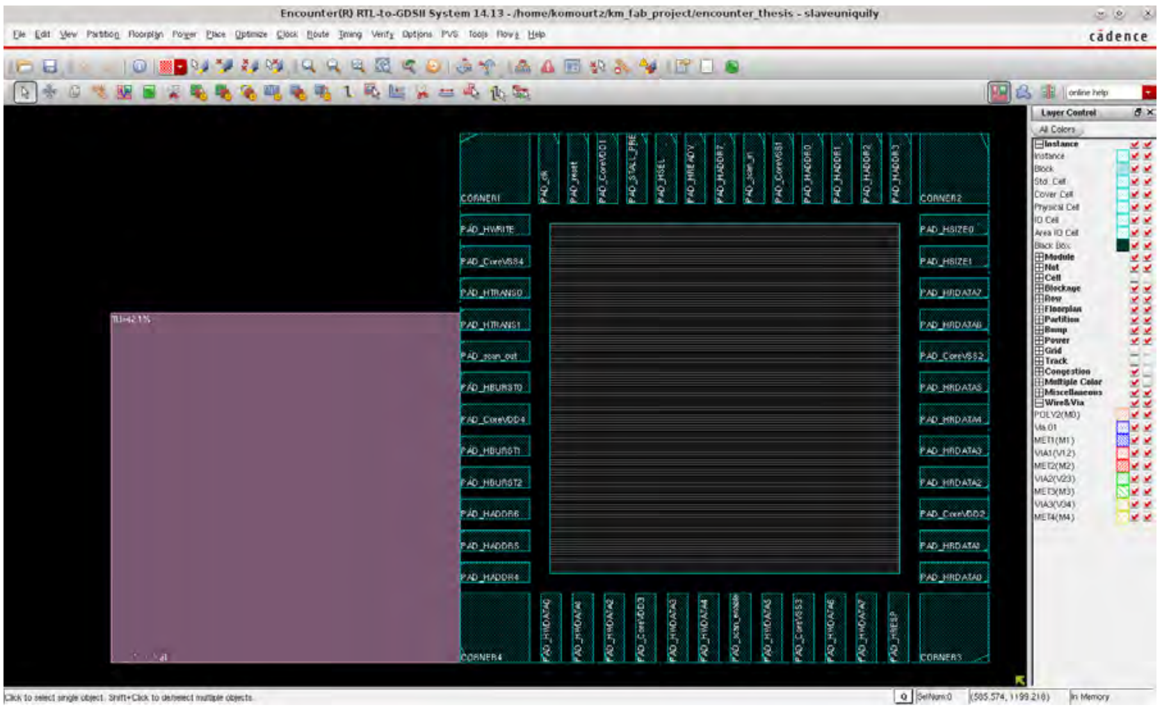


Figure 5.8: Floorplan

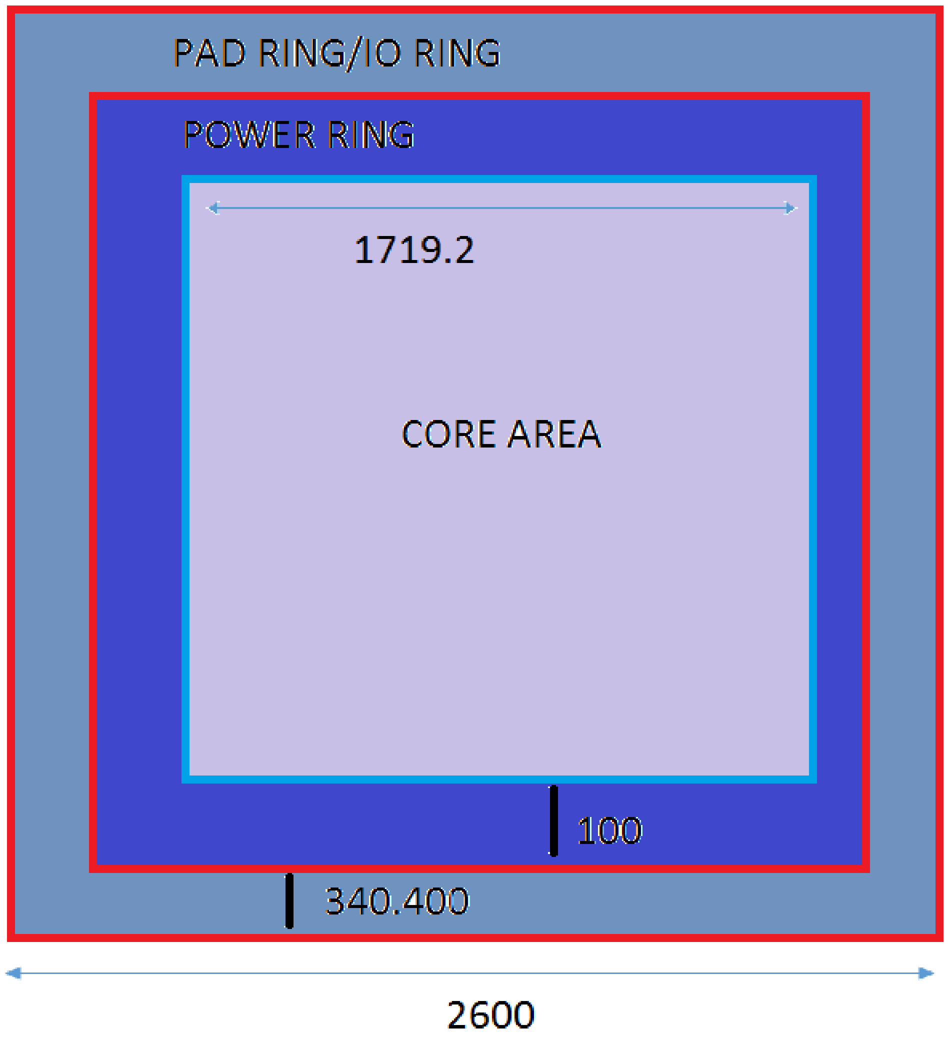


Figure 5.9: Floorplan dimensions

5.3 Power Plan

The gates on the core rows and the blocks need to be connected to the IO supply pads. This connection is done by core and block rings that need to be setup during power planning. In this section we specify the physical location and size of the power rings. Furthermore, it is recommended to route power stripes above the core rows to assure a sufficient current propagation. For the TopBottom and LeftRight we select the top routing metals, Metal 3 Horizontal and Metal 4 Vertical respectively.

We configure the width as 20um and spacing as 10um. For the offset we select center in channel so the power ring center at the margin we left for that purpose between pads and core area of the design. For the stripes we select the Metal 4 Vertical routing layer. We configure the width as 10um and spacing as 10um. Finally, we specify the spacing from the left side of the core area that the first stripe be placed at 100um and the spacing between the stripes (set-to-set distance) to 150um. In the figure 5.10 we can see the power plan of the design after the above procedure.

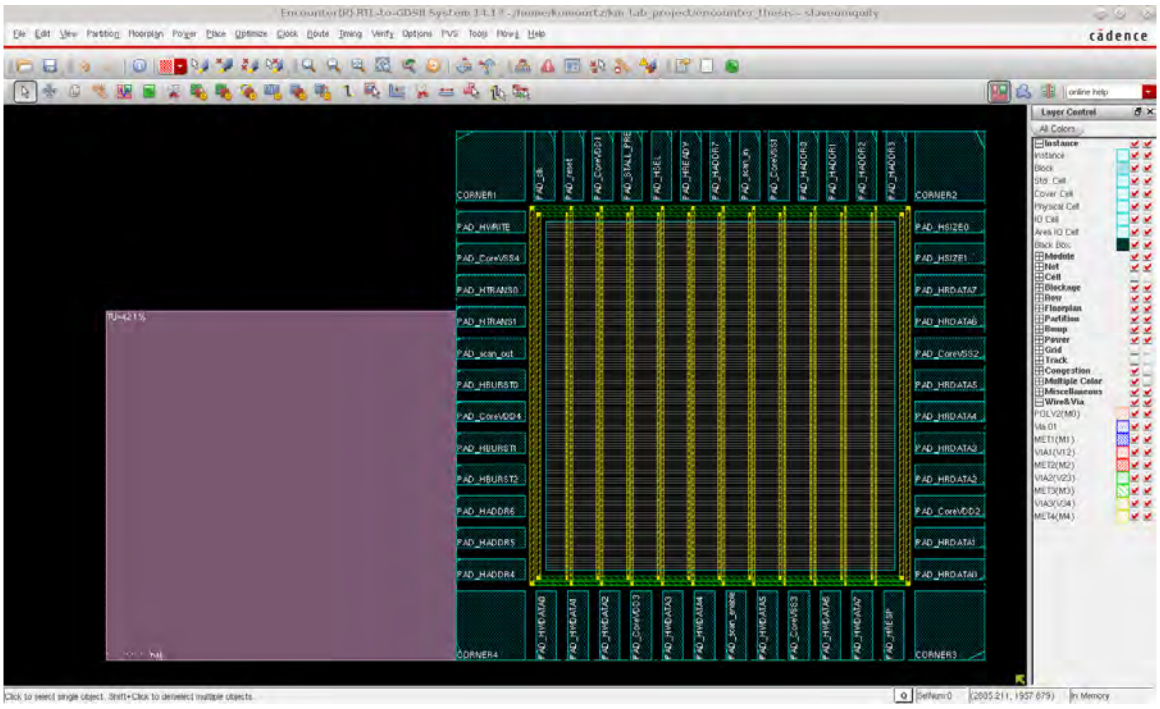


Figure 5.10: Power Plan

After power planing we have the below results:

- 4301 new pwr-pin connections were made to global net 'vdd!'.
- 4301 new gnd-pin connections were made to global net 'gnd!'.
- 53 new gnd-pin connections were made to global net 'gnd5o!'.
- 53 new gnd-pin connections were made to global net 'gnd5r!'.
- 53 new pwr-pin connections were made to global net 'vdd3r!'.
- 53 new pwr-pin connections were made to global net 'vdd5o!'.

- 53 new pwr-pin connections were made to global net 'vdd5r!'.

The vdd! and gnd! are the power and ground definitions for the core cells while vdd3o! vdd3r2! vdd3r1! vdd! are power and gnd3r! gnd3o! gnd! are the ground nets for IO pads.

3 Types of power pads are available in the digital IOLIBs (the cell names might have different extensions in the different libraries):

- VDD3I = core supply
- VDD3R = periphery logic supply (vdd3r! bus) and core supply
- VDD3O = ESD Rail supply (vdd3o! bus)
- VDD3ALL = ESD + peri-logic + core supply

In our design we had to make sure that all power bus rings in the periphery are supplied. This could either be done by using a single VDD3ALL pad or by using a combination of VDD3R, VDD3I, and VDD3O pads (Splitting the supply rails reduces noise on the nets). As the VDD3R cell could also be used to supply the core cells, it would also be possible to use only VDD3R and VDD3O cell to get a correct power supply. In this case VDD3R pad has to be connected to the core supply nets. Bellow is a representation of the power pads which maps to the IO library we use (figure 5.11).

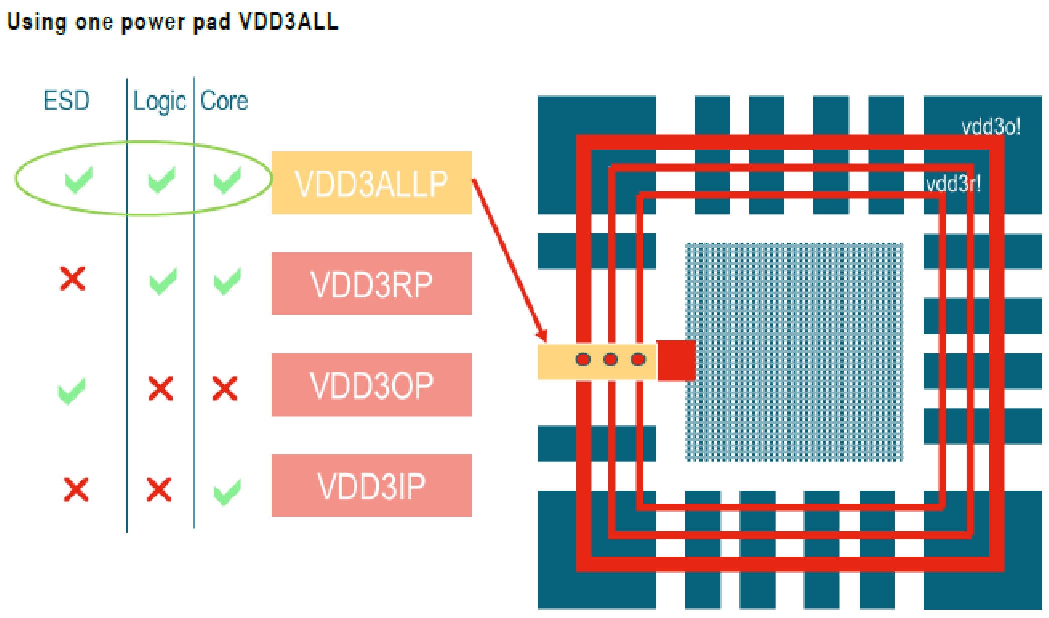


Figure 5.11: Power Pads

Similar to the power pads also 3 types of ground pads are available:

- GND3I: core supply
- GND3R: periphery logic supply (gnd3r! bus) and core supply
- GND3O: ESD Rail supply (gnd3o! bus)
- GND3ALL: core + periphery + ESD Rail supply

As for the power pads you also need to make sure that all of your ground buses in the periphery ring nets are connected to one of the ground pads.

5.4 Placement and Routing

At the next step we have to place and route our design. The standard cells have to be placed at the core area of our design. With this goal, it is necessary to select a timing-driven placement algorithm to improve the placement of instances on timing critical paths. In this optimization all the buffer, inverter and delay cells are deleted in order to calculate the real capacitance and resistance value associated to every net. With these values, SOC Encounter can estimate the optimum driving capabilities for every cell to fulfill timing requirements, some cells can be "upsize" or "downsize" depending on the requirements. Sometimes, SOC Encounter place two cells too close to each other, yielding in a "spacing violation". These violations can be fixed performing geometry verification with: then using the "Violation browser" to find the exact location and, finally, spacing the cell manually.

Before doing the design routing we need to fill the margins between IO pads with filler pads. To do that we have to move the signal pads that already are placed at the IO area of our design close to each other without any gap between them. Afterwards, we run the following command to fill the gaps that arising with the filler pads:

addIoFiller -cell PERI_SPACER_100_P_V5 -prefix IOFILLER

And we repeat the command with each pad from the ones with the bigger dimensions to the ones with the smaller dimensions (we find them by reading the .lef file that the foundry provides us) in order to fill all the remaining gaps. The purpose of doing this procedure is to have a continuous power distribution with no gaps that could cause power shutdown of the circuit. Filler pads are dummy pads that only have the three power and two ground pins mentioned at section 5.3 (They do not transfer any signal) We totally place:

Pad dimension	**100**	**50**	**20**	**10**	**5**	**2**	**1**	**0.1**
Top side	6	0	0	1	1	2	0	2
Left	7	0	0	1	1	2	0	2
Bottom	7	0	0	1	1	2	0	2
Right	7	0	0	1	1	2	0	2

Table 5.1: Global Signals

The n (0.1, 1, 2, ... 100) indicates the vertical size that each filler pad has. The results after placement are:

stdCell: stdCell: 4016 single + 0 double + 0 multi Total standard cell length = 79.0160 (mm), area = 1.0272 (mm^2)

Average module density = 0.349

Density for the design = 0.349 = stdcell_area 56440 sites (1027208 um^2) / alloc_area 161505 sites (2939387 um^2).

Pin Density = 0.412 = total number of pins 23267 / total Instance area 56440.

textitInitial total scan wire length: 89162.415

Final total scan wire length: 69323.389

Improvement: 19839.026 percent 22.25

textitTotal length: 8.640e+05um, number of vias: 49246

M1(H) length: 0.000e+00um, number of vias: 23149

M2(V) length: 3.225e+05um, number of vias: 22916

M3(H) length: 4.070e+05um, number of vias: 3181

M4(V) length: 1.345e+05um

A figure of the placed and routed design with the insertion of filler pads is shown at the figure 5.12

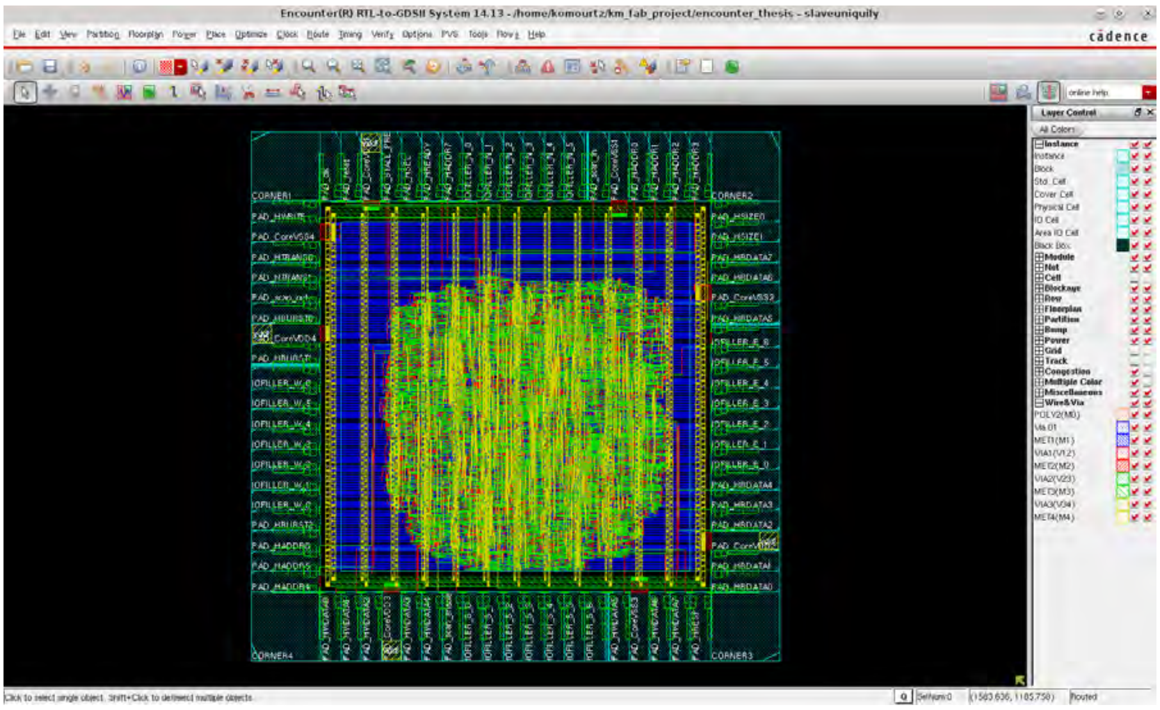

Encounter(R) RTL-to-GDSII System 14.13 - /home/komourtz/km_fab_project/encounter_thesis - slaveuniquify
File Edit View Partition Floorplan Power Place Optimize Clock Route Timing Verify Options PVS Tools Flow Help
cadence
Layer Control
CORNER1
CORNER2
CORNER3
CORNER4
Click to select single object. Shift+Click to de/select multiple objects.
Routed

Figure 5.12: Design after Placement and Routing

5.4.1 Pre-CTS optimization

After the design is placed we observe that there are some timing and area problems. We run the first design optimization (Pre-CTS), so the tool optimize the area, timing and some DRC problems of our design. Optimization is moving cells and nets in order to achieve the best timing and area results. Figure 5.13 shows the design after the optimization we perform before Clock Tree Synthesis (CTS).

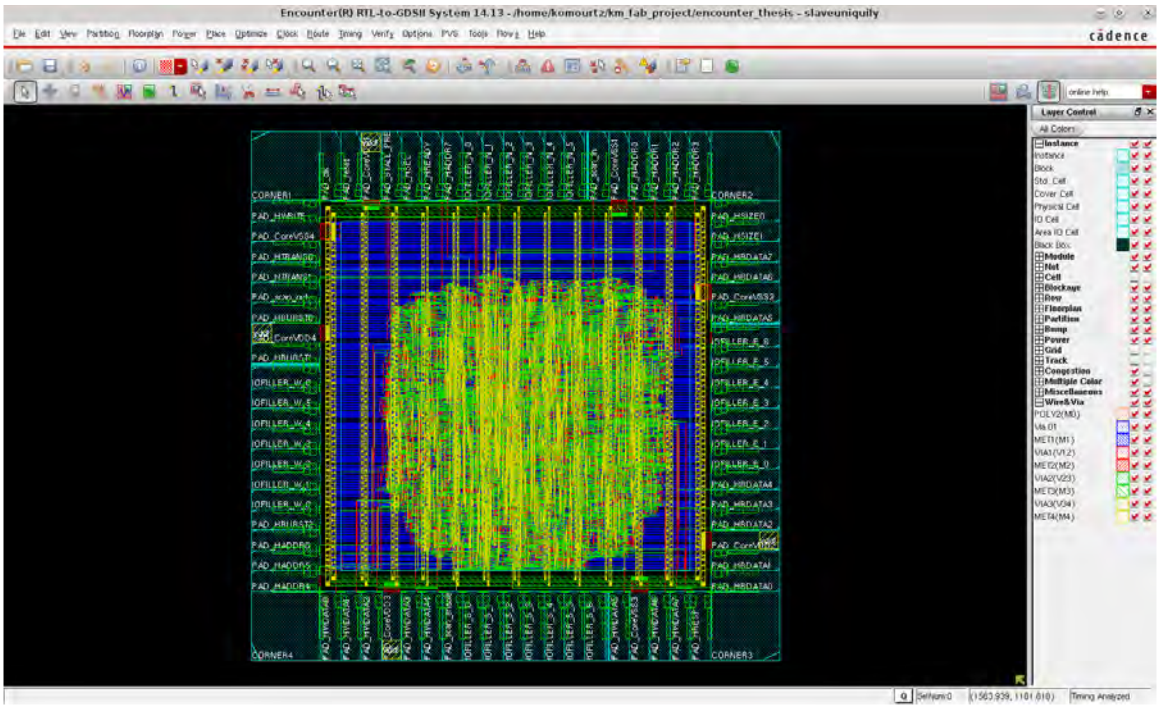


Figure 5.13: Design after Placement and Routing Optimization (Pre CTS)

5.5 Clock Tree Synthesis

The physical location of the registers is known after cell placement and thus the clock tree can be synthesized. SoC Encounter generates a clock tree by mapping the requirements in the clock specification file (.cts) and constraint file (.scf) to the physical facts. The clock tree is assembled by appropriate sized clock buffers that will be accommodated in the core row gaps. The clock should be able to reach all the points of the design at the same time, otherwise, data from the past clock cycle can be used in the current cycle. The requirements for the reset are not so strict because it is "only" necessary that reaches all the logic within a clock cycle. In order to performed this task with SOC Encounter, a "clock tree specification file" has to be provided containing the synthesis information for every clock and reset in the design. In our design we create this file from all the available buffers from the library files. A figure of the design after Clock Tree Synthesis (CTS) is shown at the figure 5.14, a more clear display is shown at figure 5.15. In figure 5.16 we can see the min/max paths of the Clock Tree Synthesis (CTS) and the final figure (5.17) shows us the phase delay of the Clock Tree Synthesis (CTS).

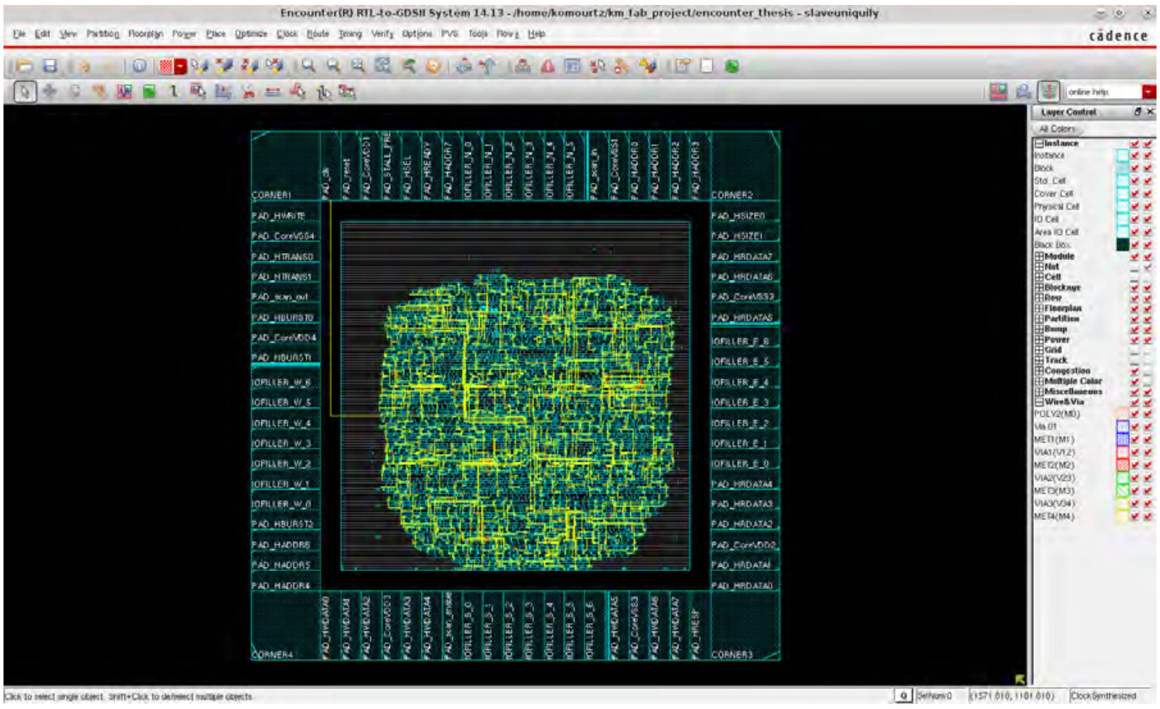

Encounter(R) RTL-to-GDSII System 14.13 - /home/komourtz/km_fab_project/encounter_thesis - slaveuniquify
cādence
Layer Control
CORNER1
CORNER2
CORNER3
CORNER4
ClockSynthesized

Figure 5.14: Clock Tree Synthesis no.1

According to [3]
Setup Time: is the minimum amount of time the data signal should be held steady before the clock event so that the data is reliably sampled by the clock. This applies to synchronous input signals to the flip-flop.
Hold Time: is the minimum amount of time the data signal should be held steady after the clock event so that the data are reliably sampled. This applies to synchronous input signals to the flip-flop.

Flip-flops are subject to a problem called **metastability**, which can happen when two inputs, such as data and clock or clock and reset, are changing at about the same time. When the order is not clear, within appropriate timing constraints, the result is that the output may behave unpredictably, taking many times longer than normal to settle to one state or the other, or even oscillating several times before settling.

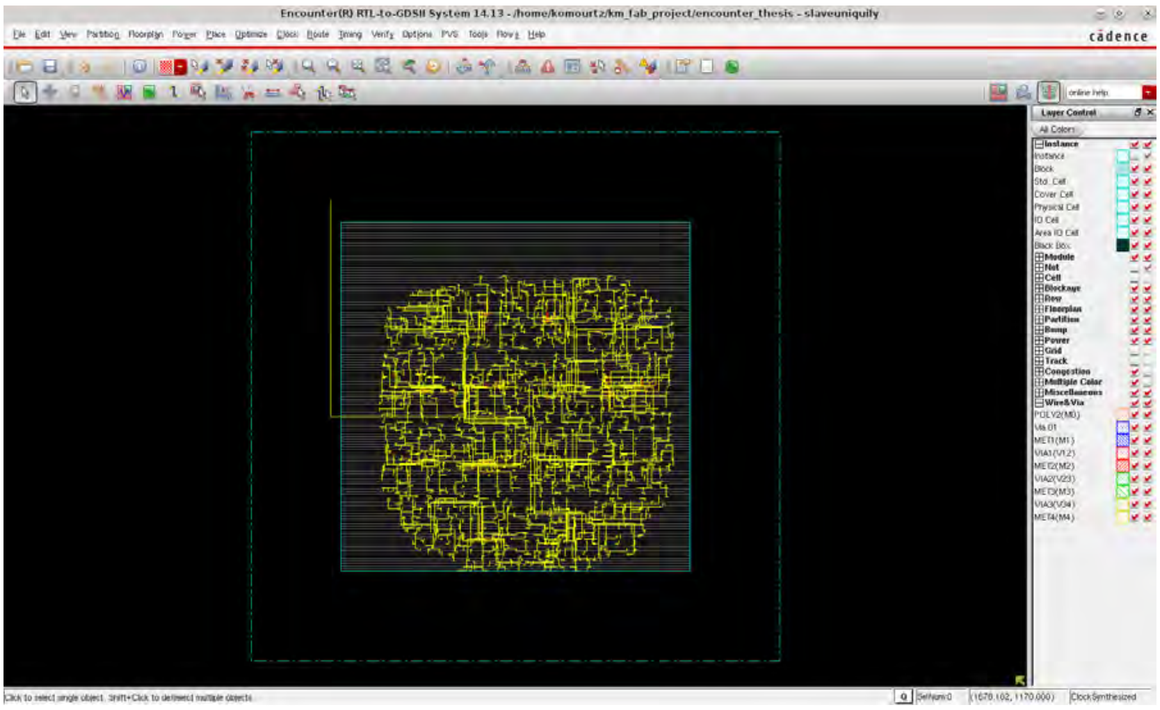

Encounter(R) RTL-to-GDSII System 14.13 - /home/komourtz/km_fab_project/encounter_thesis - slaveuniquify
cādence
Layer Control
ClockSynthesized

Figure 5.15: Clock Tree Synthesis no.2

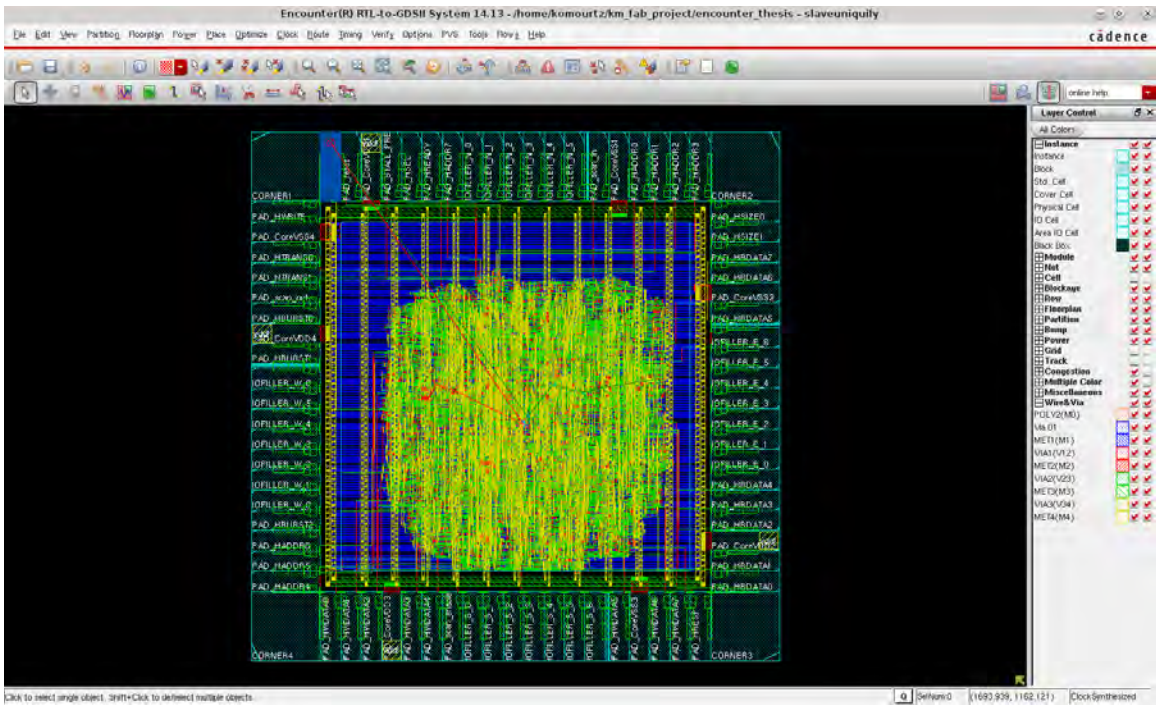

Encounter(R) RTL-to-GDSII System 14.13 - /home/komourtz/km_fab_project/encounter_thesis - slaveuniquify
cādence
Layer Control
CORNER1
CORNER2
CORNER3
CORNER4
ClockSynthesized

Figure 5.16: Clock Tree Synthesis display min max paths

As soon as the clock tree has been synthesized, a hold timing analysis has to be done. SoC Encounter needs to use the delay cells to fixed situations, which are quite common throughout the design. Like before, the design needs to be optimized using:

optDesign -postCTS -hold

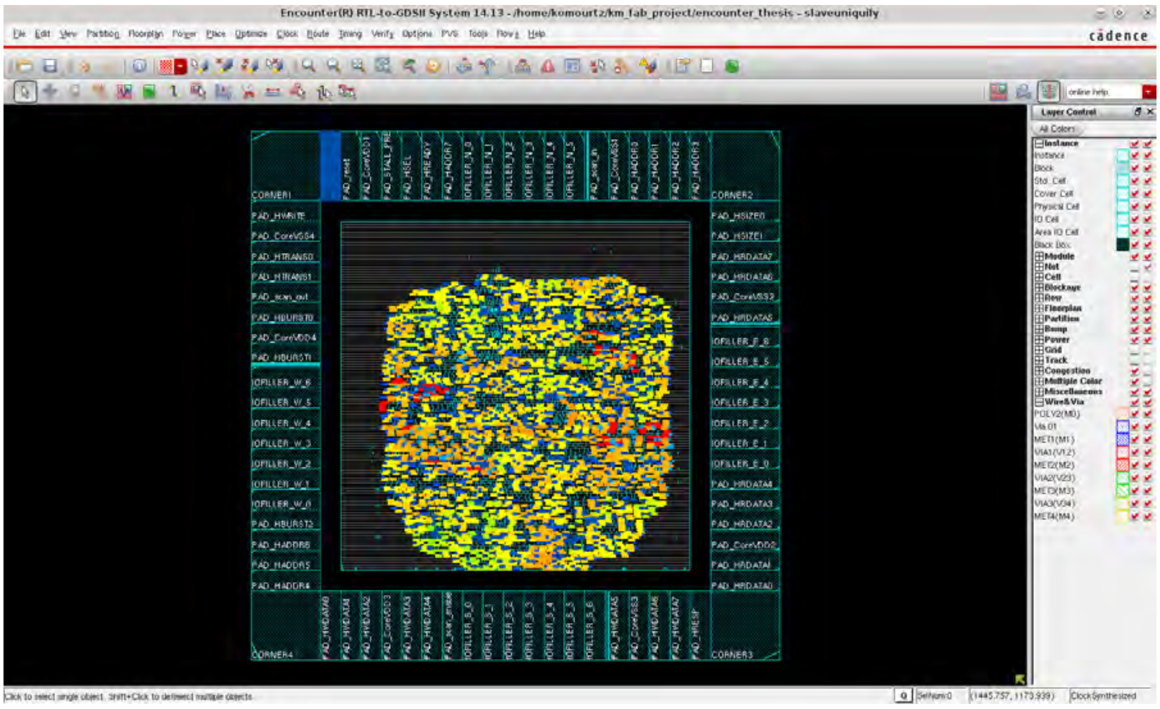

Encounter(R) RTL-to-GDSII System 14.13 - /home/komourtz/km_fab_project/encounter_thesis - slaveuniquily
File Edit View Partition Floorplan Power Place Optimize Clock Route Timing Verify Options PVS Tools Flows Help
cadence
Layer Control
CORNER1
CORNER2
CORNER3
CORNER4
Click to select single object. Shift+Click to de/select multiple objects.
ClockSynthesized

Figure 5.17: Clock Tree Synthesis phase delay

5.6 Power Analysis

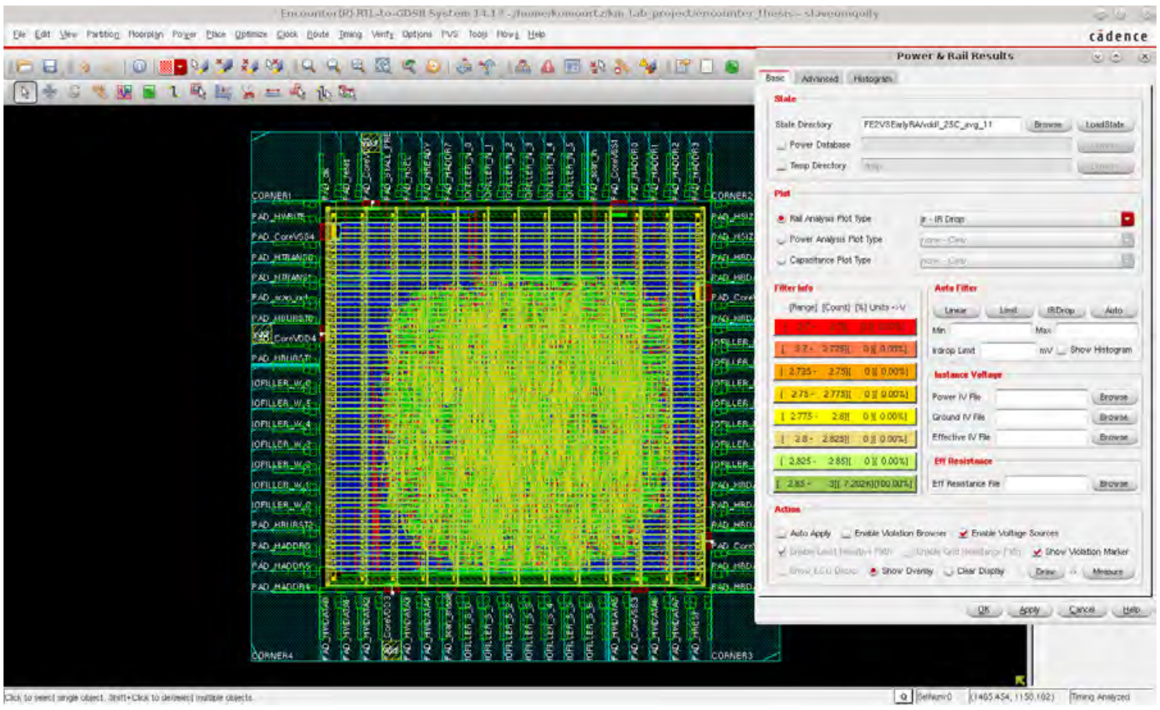
Power analysis supposes that the power plan of our design was correct and evaluate the correctness of power supply to all the cells of our design and the IR drop violations.This task is not critical in this design due to the over dimensioning of the power supply. Assuming:

- Clock frequency of 83.3 MHz (12 ns as our clock is)
- A toggle probability of 0.5. That means that the registers will toggle the half amount of the clock given.
- Worst case condition supply voltage is 4.5 Volts according to our design library.

The power of a circuit is consists of:

- Dynamic Power
- Leakage power

Internal power P_int is the power dissipated inside a cell for the charging and discharging of internal capacitances and due to crossover currents.
Switching power P_ext is the power dissipated inside a cell for charging and discharging the load capacitance connected to the cell's output. That external load consists of the input capacitances of all cells being driven plus the parasitic capacitances of the wires (aka interconnect).
The total power dissipation P_tot related to a cell can now be expressed as P_tot = P_stat + P_dyn
A detailed figure of Static Power analysis and the IR drop applied on the circuit is shown at the figure 5.18 below. While in figure 5.19 we can see the power distribution with auto adjustment.



State
State Directory: FE2VSEarlyRA/vdd!_25C_avg_11
Power Database
Temp Directory

Plot
[✓ Rail Analysis Plot Type]: ir - IR Drop
Power Analysis Plot Type: none - Clear
Capacitance Plot Type: none - Clear

Filter Info

[Range] [Count] [%] Units +/-V
🟥 [2.7 - 2.7]][0][0.00%]
🟧 [2.7 - 2.725]][0][0.00%]
🟧 [2.725 - 2.75]][0][0.00%]
🟨 [2.75 - 2.775]][0][0.00%]
🟨 [2.775 - 2.8]][0][0.00%]
🟨 [2.8 - 2.825]][0][0.00%]
🟩 [2.825 - 2.85]][0][0.00%]
🟩 [2.85 - 3]][7.202K][100.00%]

Auto Filter
Min
Max
Irdrop Limit mV
Show Histogram

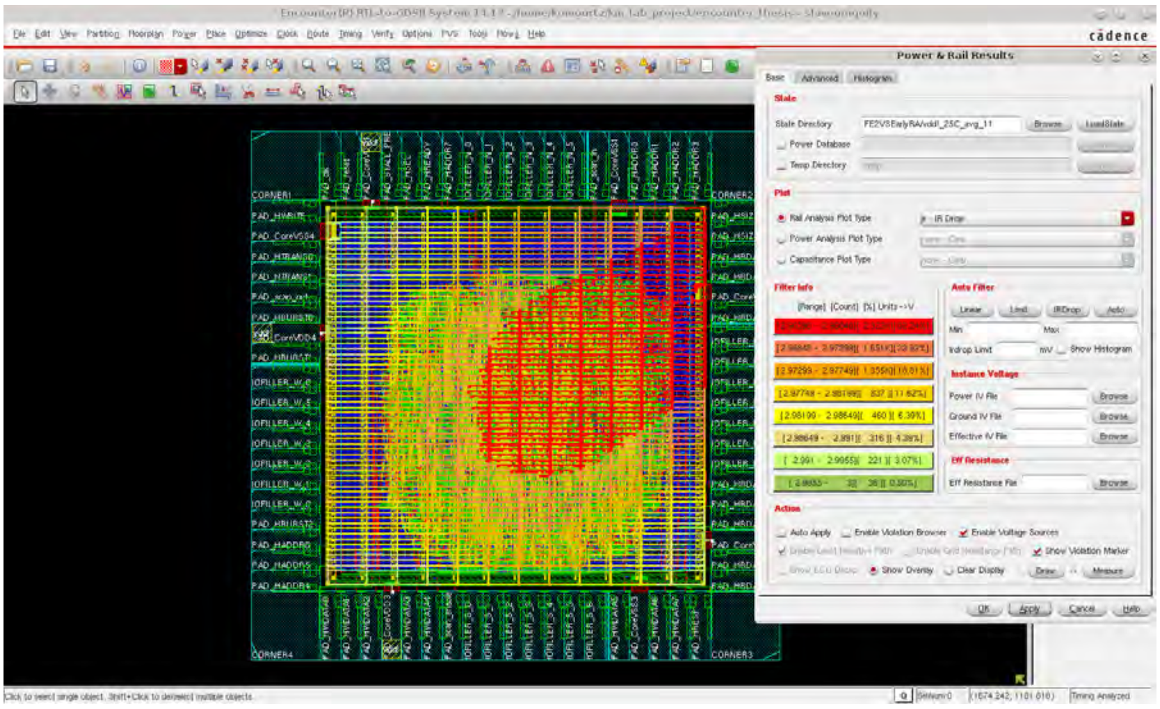
Instance Voltage
Power IV File
Ground IV File
Effective IV File

Eff Resistance
Eff Resistance File

Action
Auto Apply
Enable Violation Browser
[✓ Enable Voltage Sources]
[✓ Show Violation Marker]
[✓ Show Overlay] Clear Display

Click to select single object. Shift+Click to de/select multiple objects.
(1465.454, 1150.102) Timing Analyzed

Figure 5.18: Static Power Analysis IR drop



State
State Directory: FE2VSEarlyRA/vddl_25C_avg_11
Power Database
Temp Directory

Plot
● Rail Analysis Plot Type: IR Drop
Power Analysis Plot Type
Capacitance Plot Type

Filter info
[Range] [Count] [%] Units->V
[2.96290 - 2.96848][2.323K][32.24%]
[2.96848 - 2.97299][1.651K][22.92%]
[2.97299 - 2.97749][1.359K][18.81%]
[2.97749 - 2.98199][837][11.62%]
[2.98199 - 2.98649][460][6.39%]
[2.98649 - 2.991][316][4.39%]
[2.991 - 2.9955][221][3.07%]
[2.9955 - 3][36][0.50%]

Auto Filter
Min Max
Irdrop Limit mV Show Histogram

Instance Voltage
Power IV File
Ground IV File
Effective IV File

Eff Resistance
Eff Resistance File

Action
Auto Apply
Enable Violation Browser
✓ Enable Voltage Sources
✓ Show Violation Marker
● Show Overlay
Clear Display

Figure 5.19: Auto Power Analysis IR drop

5.7 Verification

Before exporting the final reports and files, the circuit has to pass some verification tests from Encounter. Design Rule Check (DRC) process has to be run in order to find possible layout rule violations. The command is:

verifyGeommetry

We also perform some other final verifications such as verify the connectivity of all the cells, pins, pads with the command:

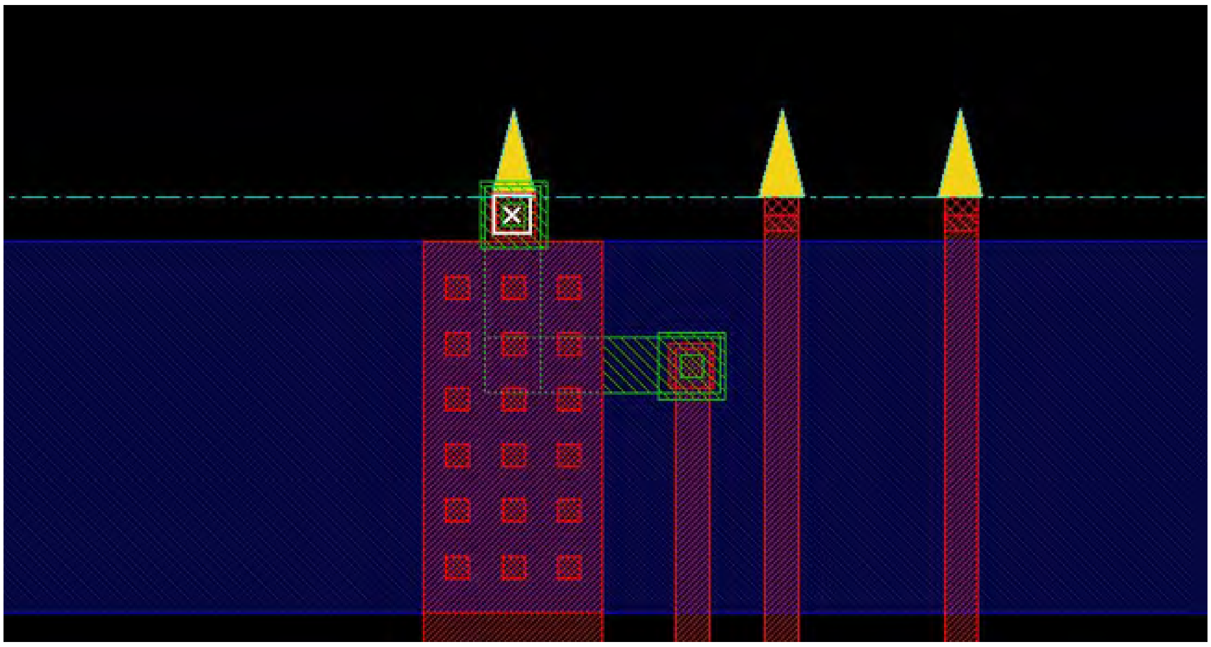
verifyConnectivity

Finally we run

verify_drc

in addition to the verifyGeommetry command to ensure that there are no DRC violations. If there are no violations, the GDSII file can be extracted. Also the SPEF, SDF and verilog netlist files are extracted. There might be some problems such as wire DRC violations. We then have to move the wires on our own with the move tool Encounter has. A figure of a wire error before and after fixing it with wire editing tools is shown in figure 5.20

We are now ready to proceed to the final step of our design finishing.


(a) Wire violation

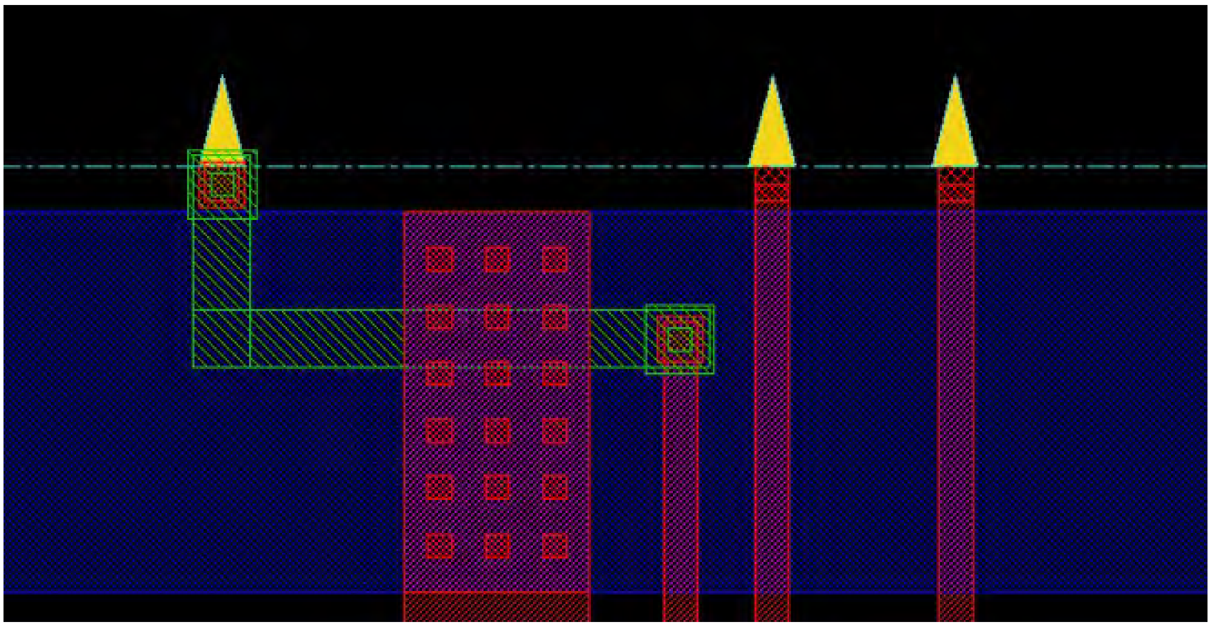

(b) Violation corrected

Figure 5.20: Wire edit example

5.8 Design Finishing

The gaps on the core and IO rows need to be filled with dummy cells referred to as core and IO filler, respectively. Core filler cells ensure the continuity of power/ground rails and N+/P+ wells in the row. Figure 5.21 shows a closer look of how the design looks like with the filler cells. We add all the available filler cells from the list from the window that shows up that our library has available. The number in the name specifies the width of the filler cell. The gaps on the pad frame need to be filled with IO fillers which connect the pad-row power supply. The IO frame is filled by execution of the command below:

addFiller -cell ENDCAPL ENDCAPR FILLCAPX16 FILLCAPX2 FILLCAPX32 FILLCAPX4 FILLCAPX8 FILLCELLX16 FILLCELLX1 FILLCELLX2 FILLCELLX32 FILLCELLX4 FILLCELLX8 -prefix FILLER

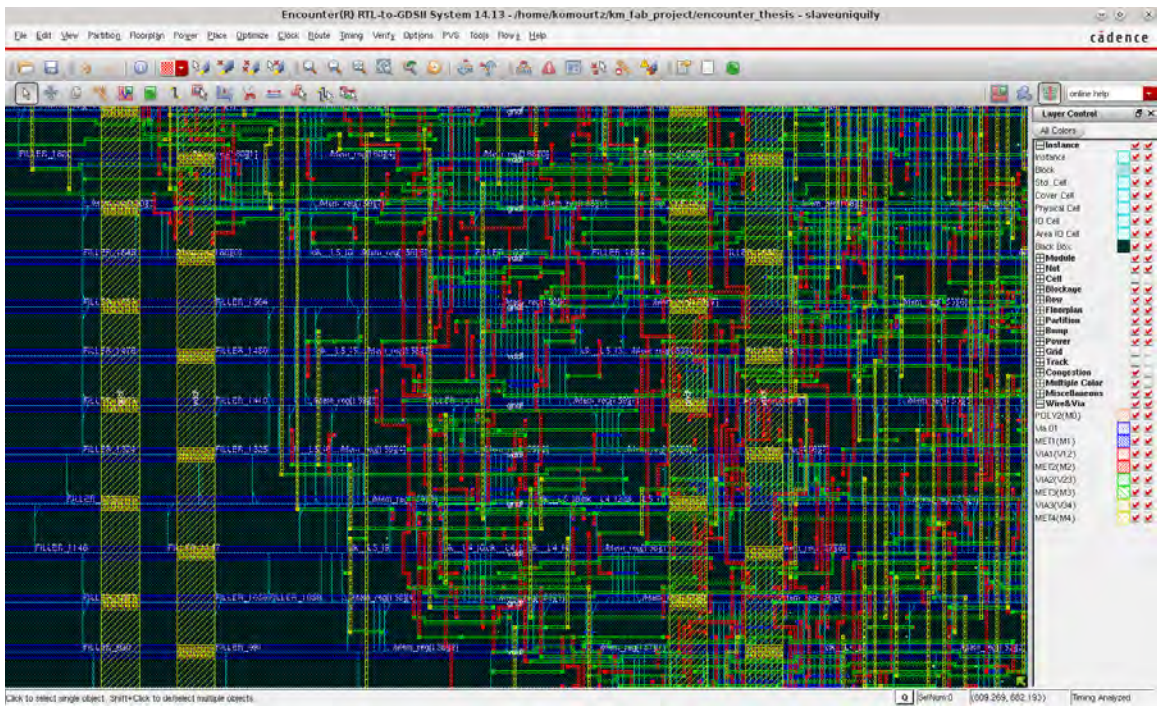

Encounter(R) RTL-to-GDSII System 14.13 - /home/komourtz/km_fab_project/encounter_thesis - slaveuniquify
File Edit View Partition Floorplan Power Place Optimize Clock Route Timing Verify Options PVS Tools Flows Help
cadence
Layer Control
Instance
Module
Net
Cell
Blockage
Row
Floorplan
Partition
Bump
Power
Grid
Track
Congestion
Multiple Color
Miscellaneous
Wire&Via
Click to select single object. Shift+Click to de/select multiple objects.
Timing Analyzed

Figure 5.21: Design Finishing filler cells

Finally for extracting the GDSII

streamOut GDSII/slaveuniquify.gds -mapFile ../../AMS/C35B4C3/cds/HK_C35/LEF/c35b4/qrclay.map -libName AMS_C35B4C3 -units 1000 -mode ALL

The GDSII file we just exported is the file that we are going to send to the foundry for the final checks (DRC, antenna process violations) they perform. If the design passes all the tests they do afterwards, then the fabrication process is ready to begin. The final result of the design with no violations is shown below on figure 5.22

And a detailed final correct routed is shown at the image 5.23

All the commands which were executed during a run of SoC Encounter are dumped into a encounter. cmd file. This file needs to be cleaned from all excessive commands that were executed on the way. Open encounter.cmd in text editor and remove such commands. Save the

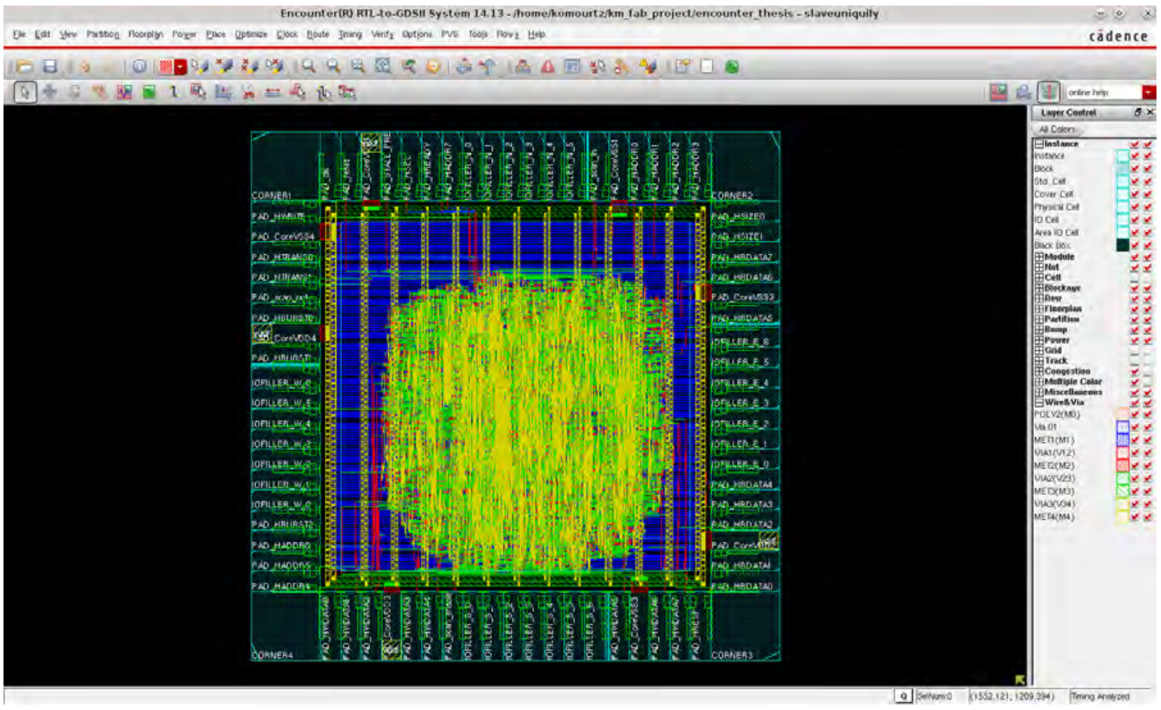

Encounter(R) RTL-to-GDSII System 14.13 - /home/komourtz/km_fab_project/encounter_thesis - slaveuniquify
cadence
Layer Control
CORNER1
CORNER2
CORNER3
CORNER4

Figure 5.22: Design Finishing filler cells

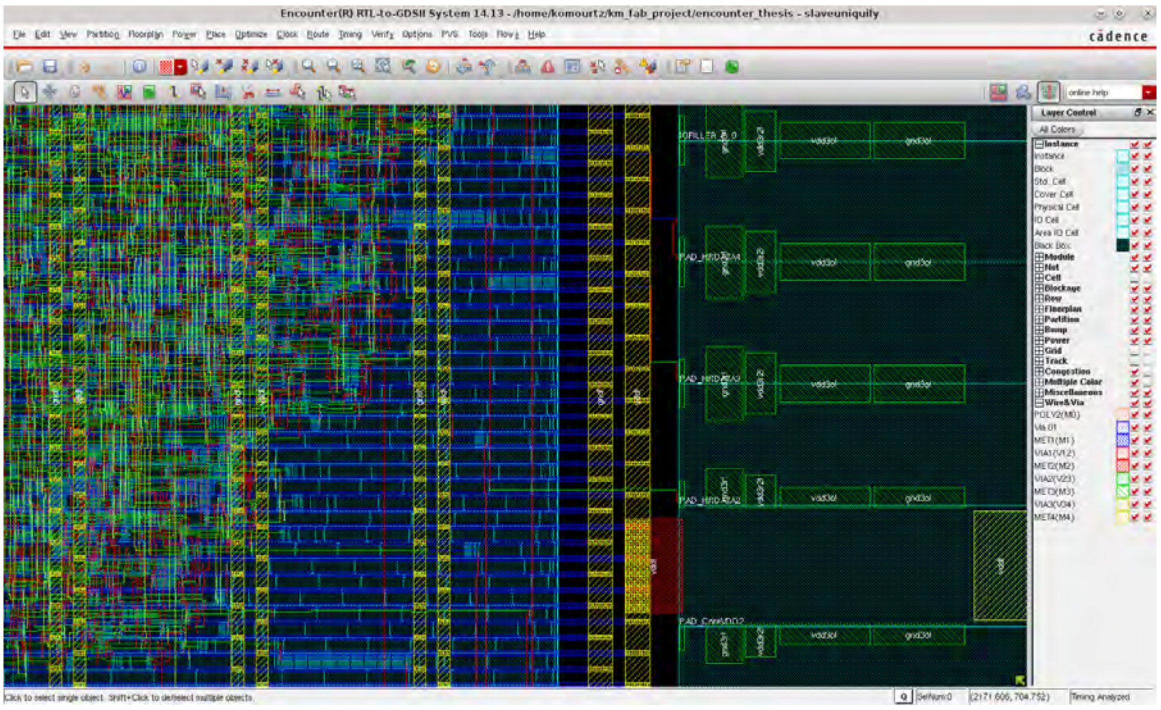
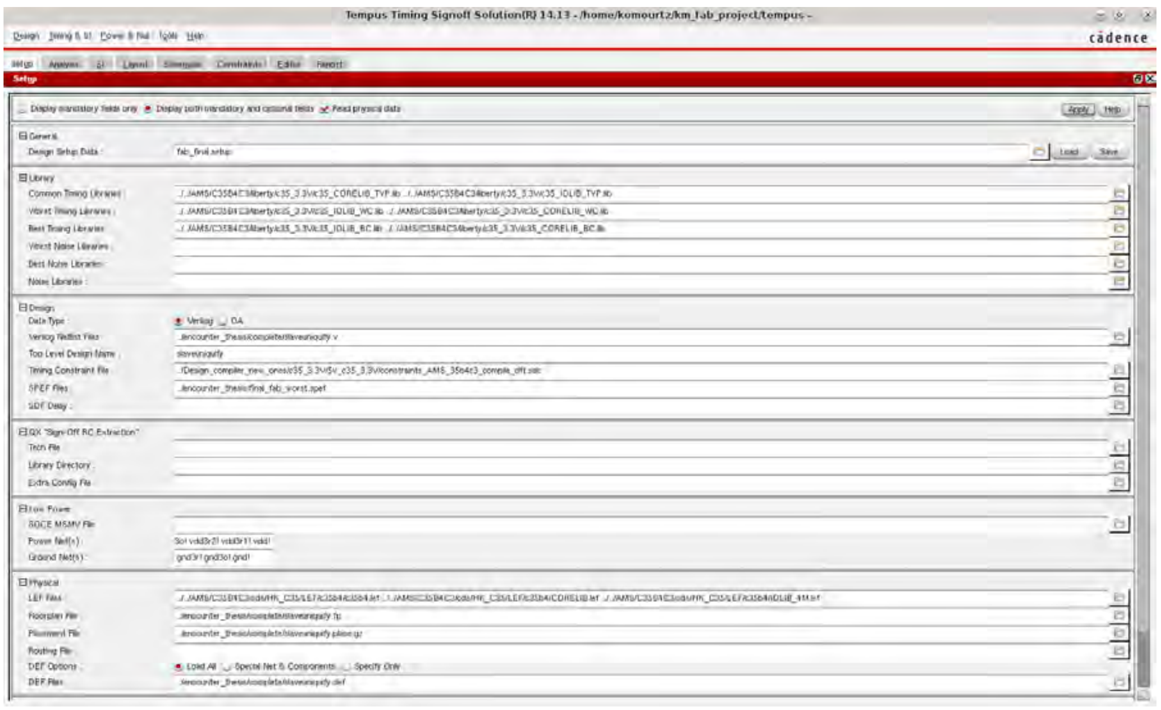

Encounter(R) RTL-to-GDSII System 14.13 - /home/komourtz/km_fab_project/encounter_thesis - slaveuniquify
cadence
Layer Control
Click to select single object. Shift+Click to de/select multiple objects.

Figure 5.23: Routed Signals Closer look

cmd file with a different name and source it in the encounter shell. The entire place and route process should be automatically executed, and result in the same layout as done manually.

5.9 Static Timing Analysis

After exporting the GDSII file from Encounter we have to run some final timing tests to ensure that the timing of our design is the same that we want to and we do not have any setup or hold violations. The EDA tool used to perform this action is Tempus from Cadence. An initial screen of the tool is shown in the figure 5.24


Tempus Timing Signoff Solution(R) 14.13 - /home/komourtz/km_fab_project/tempus -
cādence
Design Timing & SI Power & Rail Tools Help
Setup Analysis SI Layout Schematic Constraints Editor Report
Setup
Apply Help
Load Save

Display mandatory fields only [✓ Display both mandatory and optional fields] [✓ Read physical data]

General

Design Setup Data :	fab_final.setup

Library

Common Timing Libraries :	../AMS/C35B4C3/liberty/c35_3.3V/c35_CORELIB_TYP.lib ../AMS/C35B4C3/liberty/c35_3.3V/c35_IOLIB_TYP.lib
Worst Timing Libraries :	../AMS/C35B4C3/liberty/c35_3.3V/c35_IOLIB_WC.lib ../AMS/C35B4C3/liberty/c35_3.3V/c35_CORELIB_WC.lib
Best Timing Libraries :	../AMS/C35B4C3/liberty/c35_3.3V/c35_IOLIB_BC.lib ../AMS/C35B4C3/liberty/c35_3.3V/c35_CORELIB_BC.lib
Worst Noise Libraries :	
Best Noise Libraries :	
Noise Libraries :	

Design

Data Type :	[✓ Verilog] OA
Verilog Netlist Files :	../encounter_thesis/complete/slaveuniquify.v
Top Level Design Name :	slaveuniquify
Timing Constraint File :	../Design_compiler_new_ones/c35_3.3V/5V_c35_3.3V/constraints_AMS_35b4c3_compile_dft.sdc
SPEF Files :	../encounter_thesis/final_fab_worst.spef
SDF Delay :	

QX "Sign-Off RC Extraction"

Tech File :	
Library Directory :	
Extra Config File :	

Low Power

SOCE MSMV File :	
Power Net(s) :	3o1 vdd3r2! vdd3r1! vdd!
Ground Net(s) :	gnd3r1 gnd3o1 gnd!

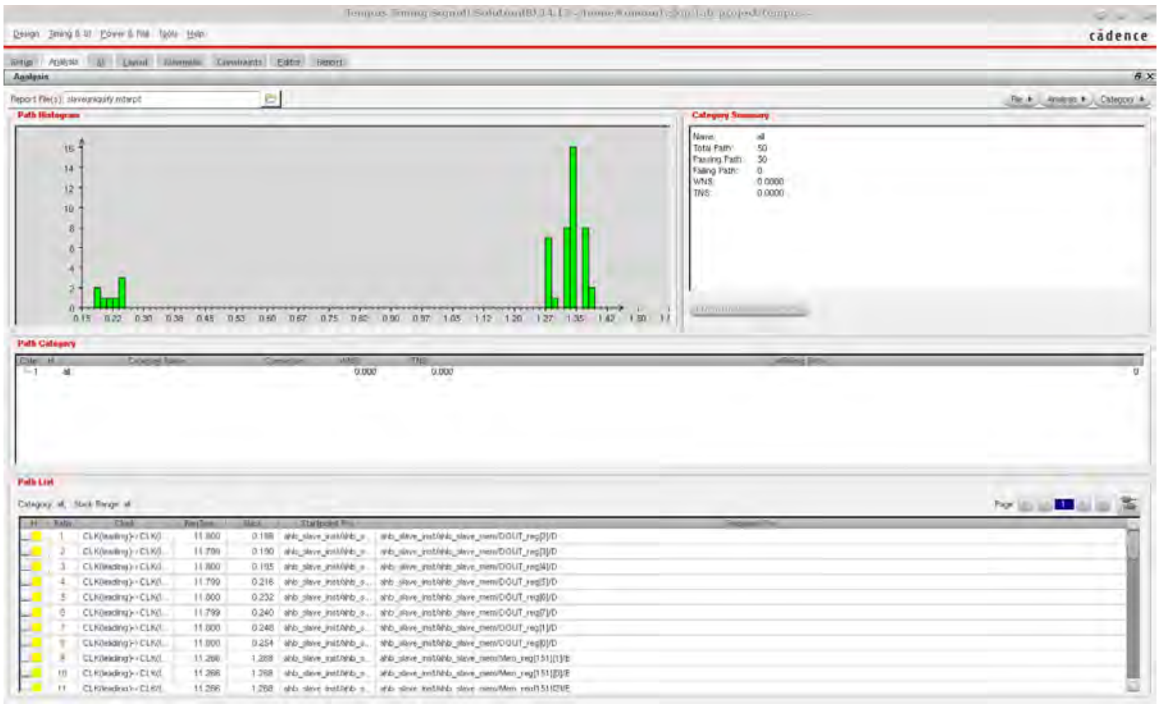
Physical

LEF Files :	../AMS/C35B4C3/cds/HK_C35/LEF/c35b4/c35b4.lef ../AMS/C35B4C3/cds/HK_C35/LEF/c35b4/CORELIB.lef ../AMS/C35B4C3/cds/HK_C35/LEF/c35b4/IOLIB_4M.lef
Floorplan File :	../encounter_thesis/complete/slaveuniquify.fp
Placement File :	../encounter_thesis/complete/slaveuniquify.place.gz
Routing File :	
DEF Options :	[✓ Load All] Special Net & Components Specify Only
DEF Files :	../encounter_thesis/complete/slaveuniquify.def

Figure 5.24: Cadence Tempus initial GUI with necessary files

As we can see at the figure 5.24, we use the .spef file, the libraries, .lef, the verilog netlist, the .sdc file from Design Compiler, the floorplan file, the .def file and the placement file exported from SoC Encounter flow. With these files we can have a schematic view of our design and proceed to the final Static Timing Analysis (STA) of our design. The most important files to do that are the verilog netlist from SoC Encounter flow, the .sdc exported from Design Compiler flow which give us all the constraints we set in our design and most importantly the clock definition. Also, very important is the .spef file. Standard Parasitic Exchange Format (SPEF) is an IEEE standard for representing parasitic data of wires in a chip in American Standard Code for Information Interchange (ASCII) format. Resistance, capacitance and inductance of wires in a chip are known as parasitic data. But SPEF does not include inductances. SPEF is used for delay calculation and ensuring signal integrity of a chip which eventually determines its speed of operation.

In figures 5.25 we can see the tempus setup time histogram and in figure 5.26 we can see the tempus hold time histogram. As we can realize from the images follows, setup and hold violations were solved after SoC Encounter flow and our design is ready for the final signoff verification test we explain to the next section(5.10)



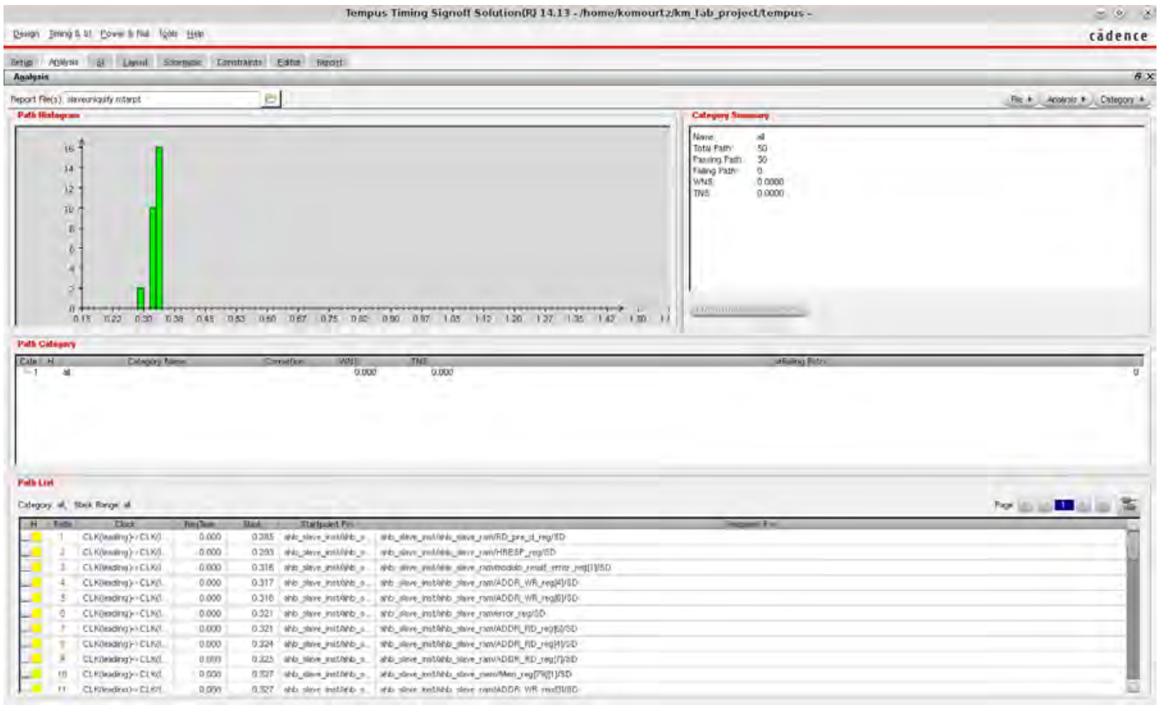
Name: all
Total Path: 50
Passing Path: 50
Failing Path: 0
WNS: 0.0000
TNS: 0.0000

Cate	H	Category Name	Correction	WNS	TNS	Failing Path
1	all			0.000	0.000	0

Category: all, Slack Range: all

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
🟨	1	CLK(leading)->CLK(l...	11.800	0.188	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/DOUT_reg[2]/D
🟨	2	CLK(leading)->CLK(l...	11.799	0.190	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/DOUT_reg[3]/D
🟨	3	CLK(leading)->CLK(l...	11.800	0.195	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/DOUT_reg[4]/D
🟨	4	CLK(leading)->CLK(l...	11.799	0.216	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/DOUT_reg[5]/D
🟨	5	CLK(leading)->CLK(l...	11.800	0.232	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/DOUT_reg[6]/D
🟨	6	CLK(leading)->CLK(l...	11.799	0.240	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/DOUT_reg[7]/D
🟨	7	CLK(leading)->CLK(l...	11.800	0.248	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/DOUT_reg[1]/D
🟨	8	CLK(leading)->CLK(l...	11.800	0.254	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/DOUT_reg[0]/D
🟨	9	CLK(leading)->CLK(l...	11.266	1.268	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/Mem_reg[151][1]/E
🟨	10	CLK(leading)->CLK(l...	11.266	1.268	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/Mem_reg[151][0]/E
🟨	11	CLK(leading)->CLK(l...	11.266	1.268	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/Mem_reg[151][2]/E

Figure 5.25: Cadence Tempus setup time histogram



Name: all
Total Path: 50
Passing Path: 50
Failing Path: 0
WNS: 0.0000
TNS: 0.0000

Cate	H	Category Name	Correction	WNS	TNS	Failing Path
1	all			0.000	0.000	0

Category: all, Slack Range: all

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
🟨	1	CLK(leading)->CLK(l...	0.000	0.285	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/RD_pre_d_reg/SD
🟨	2	CLK(leading)->CLK(l...	0.000	0.293	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/HRESP_reg/SD
🟨	3	CLK(leading)->CLK(l...	0.000	0.316	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/modulo_result_error_reg[1]/SD
🟨	4	CLK(leading)->CLK(l...	0.000	0.317	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[4]/SD
🟨	5	CLK(leading)->CLK(l...	0.000	0.318	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[6]/SD
🟨	6	CLK(leading)->CLK(l...	0.000	0.321	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/error_reg/SD
🟨	7	CLK(leading)->CLK(l...	0.000	0.321	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[5]/SD
🟨	8	CLK(leading)->CLK(l...	0.000	0.324	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[4]/SD
🟨	9	CLK(leading)->CLK(l...	0.000	0.325	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[7]/SD
🟨	10	CLK(leading)->CLK(l...	0.000	0.327	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_mem/Mem_reg[79][1]/SD
🟨	11	CLK(leading)->CLK(l...	0.000	0.327	ahb_slave_inst/ahb_s...	ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[3]/SD

Figure 5.26: Cadence Tempus hold time histogram

5.10 Signoff Formal Verification

Last but not least a final signoff verification is required in order to ensure the logic equivalence of our circuit that is now ready to send for fabrication. We verified that the netlist exported from Cadence SoC Encounter is the same in terms of logic as the netlist exported from Design Compiler. Again we had to rename some instances that were different in SoC Encounter, but the logic of the circuit was the same. We can also run Cadence Conformal from SoC Encounter gui and check the design through there. For our design, because we have only the academic licenses for the EDA tools, the integrated Conformal at SoC Encounter did not work, so we had to run it separately.

Chapter 6

Results

The results of the design is summarized at the table that follows. The table 6.1 contains the final frequency, area and power of the design that will operate after its fabrication process.

Instance	Frequency (MHz)	Area (mmˆ2)	Power (mW)
Slaveuniquify (top level)	80.64 12.4ns	6.76 (die size area)	280.83 (total power)

Table 6.1: Final Results

And the power results which consists of Dynamic Power (Switching power, Internal power) and Leakage power are shown in the table 6.2

Power(mW)	Internal	Switching	Leakage
Slaveuniquify (top level)	213.51326128 (76.0291%)	67.28797235 (23.9603%)	0.02967489 (0.0106%)

Table 6.2: Power Results

The total gate area is at 54.6000 um^2. Analytically the area per module is shown in table 6.3

Module	Gates	Cells	Area
slaveuniquify	22742	5756	1241731.4 umˆ2
ahb_slave	19442	4497	1061533.2 umˆ2
ahb_slave_ram	347	159	18964.4 umˆ2
ahb_slave_mem	19036	4296	1039402.0 umˆ2

Table 6.3: Area Results

The test coverage considering scan chain insertion was at 100% with full scan and multiplexed flip-flops style. The final GDSII file was exported and send to the foundry for fabrication (which will take approximately 4 months including the packaging and the final testing of the design). For more detailed results and reports from Electronic Design Automation (EDA) tools , you can refer to chapter 8. Finally, the required minimum area for fabrication was at $7mm^2$, so the reason we choose $6.76mm^2$ for the final die size area is to satisfy the minimum fabrication area requirements from the foundry.

Finally, we can see the run time of each EDA tool used at the table 6.4 . As we notice, the run time varies per tool. The reason of this variety on run times is depending on which corner we choose each time to run and which script with which constraints. More constraints means more run time because the tool tries to satisfy these constraints. Furthermore the minimum time achieved with the c35v5_5.0V design lib with c35_CORELIB_V5_WC.db, c35_IOLIBV5_WC.db as worst case design libraries set while the biggest run time achieved with c35_1.8V design lib with c35_CORELIBD_WC.db, c35_IOLIB_WC.db as worst case design libraries set. The v5 notation means that the cells included to the respective design libraries have a 5 Volts operating voltage while where we do not have any v5 notation the voltage is less than 5 (from 1.8 to 3.3).

The final table is making clear what made us choose the right library set combination possible. The only constraint we had is to make a design as fast as we could, so we set a clock period at 50ns to all designs and we took the results following at tables **??**, **??**, **??**.

EDA tools	Design Compiler	TetraMAX ATPG	SoC Encounter
runtime (minutes)	3 - 120	4 - 6	60-240
Memory Usage (GB) of 32GB available	1	1	1.5
# of Cores used	1 (local)	1 (local)	8 (local)

Table 6.4: Run time Results

Process	1.8v	1.8v	2.2v	2.2v
Library set (Core-IO libs)	CORELIB - IOLIB	CORELIBD - IOLIB	CORELIB - IOLIB	CORELIBD - IOLIB
Worst negative slack (ns)	23.347620	20.687208	31.461990	30.307341
Total Power (mW)	5.218016	4.538731	7.836362	6.828680
Internal Power (mW)	3.190864	2.756248	4.749336	4.153425
Switching Power (mW)	2.013087	1.777478	3.070842	2.669174
Leakage Power (pW)	1.406694e+07	4.991078e+06	1.618433e+07	6.087151e+06
Area (um^2)	2628676.707783	2560673.007113	2619785.905987	2549806.806873

Table 6.5: Library process comparison

Process	2.7v	2.7v	3.3v	3.3v
Library set (Core-IO libs)	CORELIB - IOLIB	CORELIBD - IOLIB	CORELIB - IOLIB	CORELIBD - IOLIB
Worst negative slack (ns)	34.920925	33.997688	0.404160	36.811172
Total Power (mW)	11.743589	10.414916	71.594215	15.733788
Internal Power (mW)	7.303819	6.362256	44.437351	9.656704
Switching Power (mW)	4.420347	4.045175	27.132803	6.067789
Leakage Power (pW)	1.943141e+07	7.528124e+06	2.404943e+07	9.328473e+06
Area (um^2)	2604102.503986	2546070.286953	2600429.902935	2544221.926521

Table 6.6: Library process comparison

Process	2v	3v	4v	5v
Library set (Core-IO libs)	CORELIB_V5 - IOLIBV5	CORELIB_V5 - IOLIBV5	CORELIB_V5 - IOLIBV5	CORELIB_V5 - IOLIBV5
Worst negative slack (ns)	-7.356525	14.098717	22.341166	24.976328
Total Power (mW)	9.867247	17.576820	29.046421	45.671612
Internal Power (mW)	6.408111	11.318076	18.348286	29.017313
Switching Power (mW)	3.454204	6.252011	10.689300	16.643463
Leakage Power (pW)	4.938449e+06	6.735742e+06	8.845647e+06	1.083181e+07
Area (um^2)	3032106.079094	2878373.882740	2860239.482403	2838949.682816

Table 6.7: Library process comparison

Chapter 7

Conclusion and future work

7.1 Conclusion

If we consider the years passed from the invention of transistor to nowadays, we can see that technology made huge steps throughout these years. For almost half century we moved from big transistors that could be visible in bare eyes to transistors that we need the most advanced microscopes to see their form. Furthermore, from "hand-made" designs we moved to designs created with the aid of Electronic Design Automation (EDA) tools. These tools helped designers to build big projects, such as processors and fit billion of transistors in a small area. Also, by decreasing the transistors dimensions, we managed to make computations even more faster and less power consuming. Nowadays, we talk for System on Chip (SoC), which are chips containing billions of transistors and made for many purposes. As we can see at the table 6.2 of chapter 6 we realize that the total amount of leakage power the circuit consumes is way more smaller that the total dynamic power of the circuit. As we mentioned before that happens because of the technology we use, where if we have bigger technology our concerns are on the Dynamic power whilst we have smaller designs e.g. 90nm and below leakage power becomes more important that the Dynamic power. The main goal of this thesis that was to complete a VLSI design flow of AMBA ARM AHB-lite slave circuit at 350nm with AMS C35B4C3 technology library with the latest versions of EDA tools which legally provided for academic use from Europractice agreement my department has, was achieved. Now the chip will enter the fabrication process with the foundry in order to approximately 4 months from now we have a full functional and fabricated AMBA ARM AHB-lite slave circuit. The process we followed in order to achieve that outcome was:

- Take the specifications from [7] of the AHB-lite slave circuit in order to design the block diagrams and the signals the circuit would have according to these specifications.

- Afterwards, we started to write the RTL code which part of the RTL code that described the circuit found at opencores website. At this step the interaction and the feedback with the simulation EDA tools was necessary. After using three EDA tools (VCSMX, Simvision, Questasim) from the three biggest vendors (Synopsys, Cadence, Mentor Graphics) to compare the results and ensure that the waveforms produced was identical, the initial RTL with exhaustive testbench, written from the AHB-lite slave specifications of ARM, tested at functional simulation mode and if something went wrong we had to go back and correct some sections on the RTL code.

- When we took a waveform as we designed it, we used Design Compiler to complete the logic synthesis of the circuit and scan chain insertion. After many trials that the design has to pass in order to satisfy the timing mostly requirements, we write out the verilog netlist produced from Synopsys Design Compiler as well as .sdc file that was the constraints file we set during the logic synthesis and other files that gave us the opportunity to double-check our results and reach to a conclusion that we had a netlist that satisfied our requirements.

- The synthesized netlist produced from Design Compiler, imported to Synopsys TetraMAX ATPG which is a tool for Automatic Test Pattern Generation (ATPG). There, we got results for the test coverage of scan chains of the circuit itself as well as the test vectors that we will use after fabrication process to test our design for stuck-at faults.

- Before going to the physical implementation of our design, we had to do a final check with Cadence Conformal, for the logic equivalence checking of the circuit.

- Now we were ready to proceed to the layout flow of our design with Cadence SoC Encounter EDA tool were we indented to export the final GDSII file that will be send for fabrication.

First of all we had to floorplan the design, i.e. define the area that the design would have in order the pads, power ring and standard cells would be placed. Then, we proceed with the power plan where we defined the power and ground nets of our design and made the power ring, and stripes in order to have a proper power distribution to our circuit. Afterwards, we made the placement and the first routing of the signals in our design. In this step we got the first power, density (percentage of area the design had of the core area that was available) and timing results. After the standard cells placed we had to create the clock tree of the design in order to eliminate all timing violations of our design. The next step was to analyze the power of our design in order to examine if there were any hot spots of areas that needed a lot of power consumption and take the necessary measures to eliminate these violations. Before exporting the GDSII file was necessary to do the final verification tests regarding DRC and connectivity mostly. After the design passed all the verification steps we produced the final GDSII file to be send to the foundry for fabrication as well as the final verilog netlist, parasitic extraction files, placed design files, etc.

- Before our design was send to the foundry we had to do some final Static Timing Analysis (STA) tests to ensure that the timing of our design was correct.
- Last but not least we had to ensure that our design after layout is correct in terms of Logic. For that purpose a final signoff logic equivalence checking with Cadence Conformal needed to performed.

7.2 Future Work

In this final section, we have to mention the future work can be done on this thesis. First of all, for academic funding reasons the final design will be fabricated at 350nm with a voltage of 5 Volts. For future System on Chip (SoC) fabrications at the department can be used a smaller design process, such as 45nm or 28nm. Despite the area reduction we will observe, we will realize the meaning of leakage power problems on this smaller processes designs. According to [11], Power consumption (7.1) can be divided into two aspects:

- **Dynamic power** the power that is consumed by a device when it is actively switching from one state to another. Dynamic power consists of switching power, consumed while charging and discharging the loads on a device, and internal power (also referred to as short circuit power), consumed internal to the device while it is changing state.
- **Leakage power** the power consumed by a device not related to state changes (also referred to as static power). Leakage power is actually consumed when a device is both static and switching, but generally the main concern with leakage power is when the device is in its inactive state, as all the power consumed in this state is considered "wasted" power.

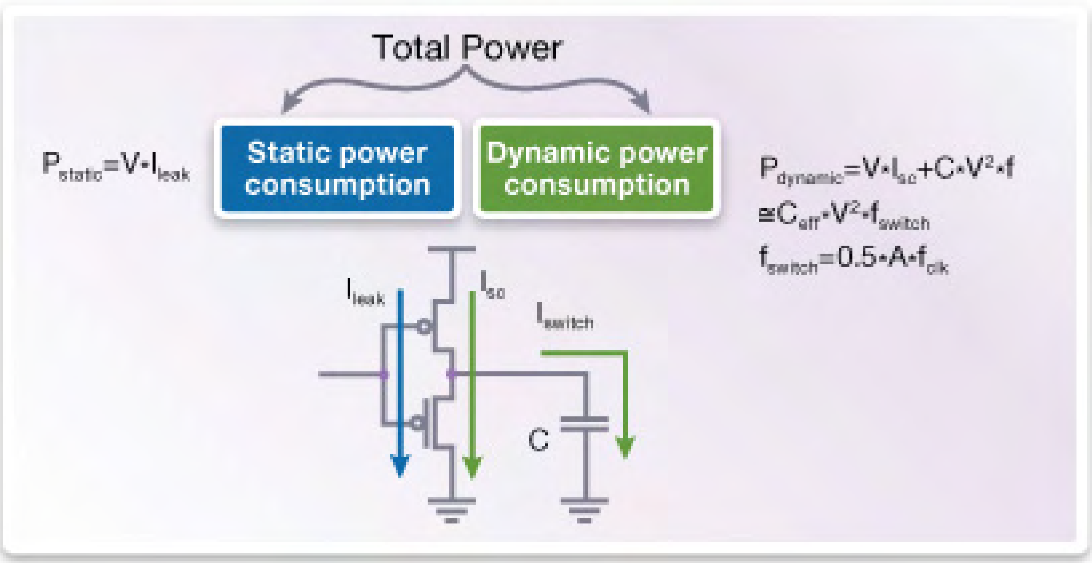
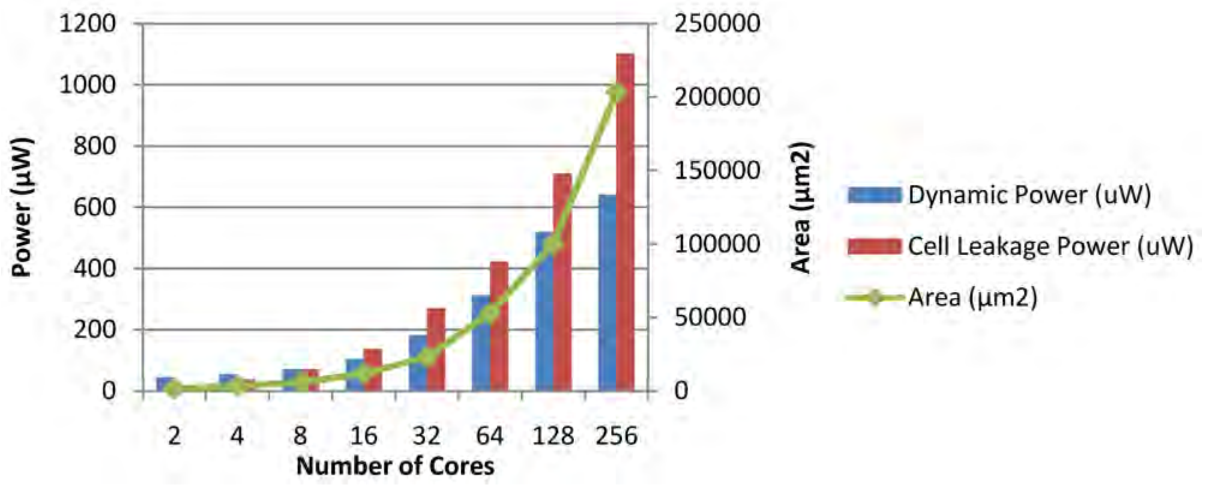
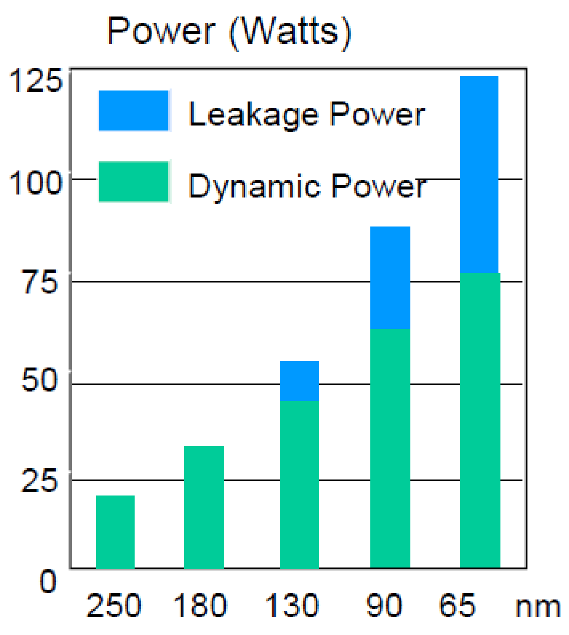

Total Power
Static power consumption
Dynamic power consumption
$P_{static}=V \cdot I_{leak}$
$P_{dynamic}=V \cdot I_{sc}+C \cdot V^2 \cdot f$
$\cong C_{eff} \cdot V^2 \cdot f_{switch}$
$f_{switch}=0.5 \cdot A \cdot f_{clk}$
I_{leak}
I_{sc}
I_{switch}
C

Figure 7.1: Total circuit power

As we can see at figure 7.2, leakage power becomes the main concern as the transistors getting smaller. Furthermore we have more cores in our designs nowadays, so leakage power is a big problem there also.



(a) Power vs number of cores

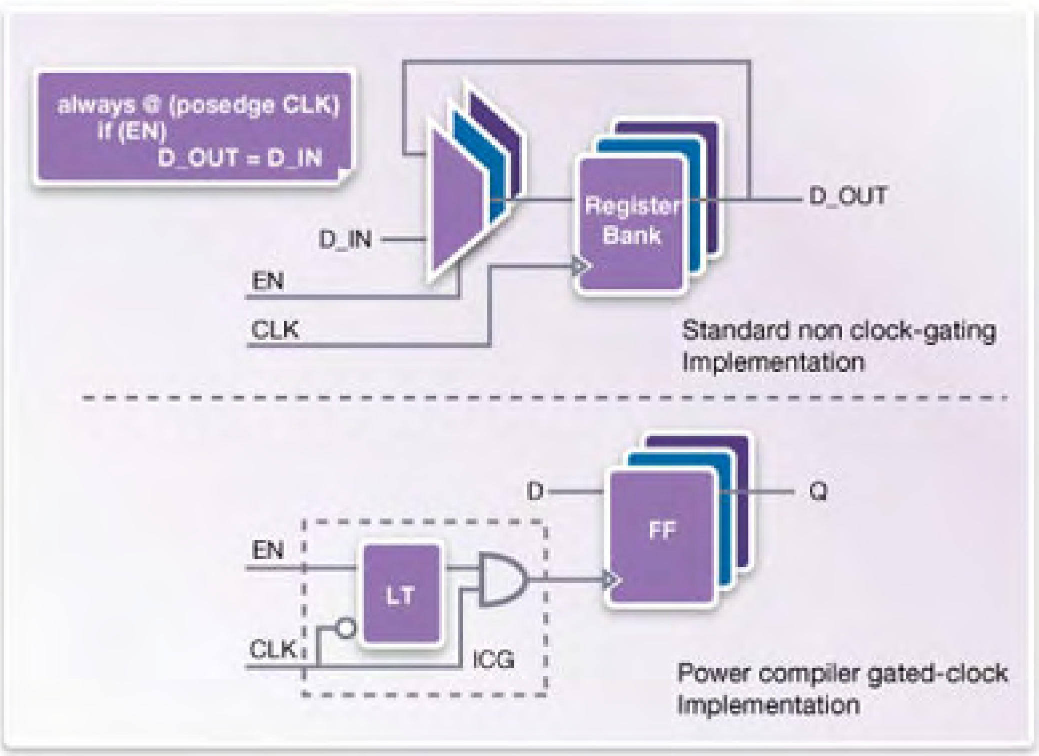


(b) Power vs technology scaling

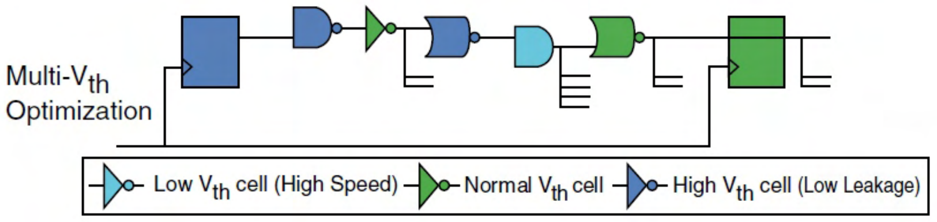
Figure 7.2: Power distribution with technology scaling

Various techniques have been developed to reduce both dynamic and leakage power. The two most common traditional, mainstream techniques are:

- **Clock gating** the disconnecting of the clock from a device it drives when the data going into the device is not changing. This technique is used to minimize dynamic power (figure 7.3a).
- **Multi-Vth** optimization – the replacement of faster Low-Vth cells, which consume more leakage power, with slower High-Vth cells, which consume less leakage power. Since the High-Vth cells are slower, this swapping only occurs on timing paths that have positive slack and thus can be allowed to slow down (figure 7.3b).



(a) Clock gating (source: Synopsys)


Multi-V_{th} Optimization
Low V_{th} cell (High Speed)
Normal V_{th} cell
High V_{th} cell (Low Leakage)

(b) Multi-Vth

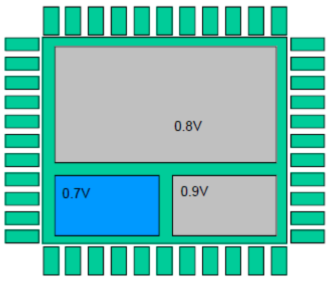
Figure 7.3: Synopsys VCSMX vs Cadence Simvision initial waveform

As technologies have shrunk, leakage power consumption has grown exponentially, thus requiring more aggressive power reduction techniques to be used. Similarly, clock frequency increases have caused dynamic power consumption of the devices to outstrip the capacity of the power networks that supply them, and this becomes especially acute when high power consumption occurs in very small geometries, as this is a power density issue as well as a power consumption issue.

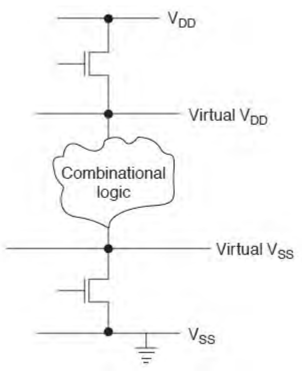
Several advanced low power techniques have been developed to address these needs. The most commonly adopted techniques today are:

- **Multi-voltage (MV)** the operation of different areas of a design at different voltage levels. Only specific areas that require a higher voltage to meet performance targets are connected to the higher voltage supplies. Other portions of the design operate at a lower voltage, allowing for significant power savings. Multi-voltage is generally a technique used to reduce dynamic power, but the lower voltage values also cause leakage power to be reduced (figure 7.4a).

- **Power gating** the complete shut off of supply nets to different areas of a design when they are not needed (also known as MTCMOS or power shutdown). Since the power has been completely removed from these shutdown areas, the power for these areas is reduced essentially to zero. This technique is used to reduce leakage power (figure 7.4b).



(a) Multy supply voltage



(b) Power Gating

Figure 7.4: Advanced low-power techniques

It is very common to see multi-voltage and power gating used together on the same design, whereby different regions operate at different voltages, and one or more of those regions can also be shutdown.

Finally, if we move a step beyond and do a research on EDA tools comparison. This comparison will consist of EDA tools of Synopsys and Cadence. More specifically the comparison will be as follows:

- **Logic Synthesis:** Synopsys Design Compiler vs Cadence RTL Compiler
- **Logic Equivalence Checking:** Synopsys Formality vs Cadence Conformal
- **Automatic Test Pattern Generation:** Synopsys TetraMAX ATPG vs Cadence Encounter Test
- **Physical Design:** Synopsys IC Compiler vs Cadence SoC Encounter
- **Static Timing Analysis:** Synopsys Primetime vs Cadence Tempus

Chapter 8

Appendix A

8.1 Design Compiler Results

Area report (top level)

```
Report : area
Design : slaveuniquify
Version: J-2014.09-SP2
Date   : Mon Jul  6 00:11:49 2015

Librarys Used:

    c35_CORELIB_WC File: /home/komourtz/AMS/C35B4C3/synopsys/c35_3.3V/c35_CORELIB_WC.db
    c35_IOLIB_WC File: /home/komourtz/AMS/C35B4C3/synopsys/c35_3.3V/c35_IOLIB_WC.db

Number of ports:                          41
Number of nets:                           82
Number of cells:                          42
Number of combinational cells:             0
Number of sequential cells:                0
Number of macros/black boxes:             41
Number of buf/inv:                         0
Number of references:                      4

Combinational area:            171826.199623
Buf/Inv area:                   18054.399719
Noncombinational area:         871197.612701
Macro/Black Box area:         1395640.000000
Net Interconnect area:         161766.090611

Total cell area:              2438663.812325
Total area:                   2600429.902935
1
```

Violation report

```
Report : constraint
        -all_violators
Design : slaveuniquify
Version: J-2014.09-SP2
Date   : Mon Jul  6 00:11:47 2015


   max_area

                          Required        Actual
   Design                   Area           Area           Slack
   -----------------------------------------------------------------
   slaveuniquify           0.000000       2600430.000000 -2600430.000000
                                                                   VIOLATED


1
```

Setup timing report

```
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : slaveuniquify
Version: J-2014.09-SP2
Date   : Mon Jul  6 00:11:49 2015

Operating Conditions: WORST-MIL   Library: c35_CORELIB_WC
Wire Load Model Mode: enclosed

  Startpoint: ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[1]
              rising edge-triggered flip-flop clocked by CLK
  Endpoint: ahb_slave_inst/ahb_slave_mem/DOUT_reg[1]
            rising edge-triggered flip-flop clocked by CLK
  Path Group: CLK
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  slaveuniquify      30k                   c35_CORELIB_WC
  ahb_slave_test_1   10k                   c35_CORELIB_WC
  ahb_slave_mem_test_1
                     10k                   c35_CORELIB_WC

  Point                                                     Incr       Path
  --------------------------------------------------------------------------
  clock CLK rise edge                                 0.000000   0.000000
  clock network delay ideal                           0.000000   0.000000
  ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[1]/C DFSC1
                                                        0.000000   0.000000 r
  ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[1]/Q DFSC1
                                                        1.918338   1.918338 f
  ahb_slave_inst/ahb_slave_ram/ADDR_RD[1] ahb_slave_ram_test_1
                                                        0.000000   1.918338 f
  ahb_slave_inst/ahb_slave_mem/ADDR_RD[1] ahb_slave_mem_test_1
                                                        0.000000   1.918338 f
  ahb_slave_inst/ahb_slave_mem/U992/Q NOR21         1.204737   3.123075 r
  ahb_slave_inst/ahb_slave_mem/U1387/Q NAND22       0.168019   3.291094 f
  ahb_slave_inst/ahb_slave_mem/U1386/Q INV3         1.733972   5.025066 r
  ahb_slave_inst/ahb_slave_mem/U27/Q BUF2           2.337953   7.363019 r
  ahb_slave_inst/ahb_slave_mem/U196/Q AOI221        0.579772   7.942791 f
  ahb_slave_inst/ahb_slave_mem/U195/Q NAND41        1.057071   8.999863 r
  ahb_slave_inst/ahb_slave_mem/U539/Q OAI212        0.236692   9.236555 f
  ahb_slave_inst/ahb_slave_mem/U1536/Q NAND41       0.842835   10.079391 r
  ahb_slave_inst/ahb_slave_mem/U1688/Q AOI221       0.454668   10.534059 f
  ahb_slave_inst/ahb_slave_mem/U1686/Q NAND22       0.463758   10.997816 r
  ahb_slave_inst/ahb_slave_mem/DOUT_reg[1]/D DFSEC1 0.000134   10.997951 r
  data arrival time                                              10.997951

  clock CLK rise edge                                 12.000000  12.000000
  clock network delay ideal                           0.000000   12.000000
  ahb_slave_inst/ahb_slave_mem/DOUT_reg[1]/C DFSEC1 0.000000   12.000000 r
  library setup time                                  -0.597889  11.402111
  data required time                                             11.402111
  ---------------------------------------------------------------------------
  data required time                                                11.402111
  data arrival time                                                -10.997951
  ---------------------------------------------------------------------------
  slack MET                                                       0.404160


1
```

Power report

```
Information: Updating design information... UID-85
Information: Propagating switching activity low effort zero delay simulation. PWR-6
Warning: Design has unannotated primary inputs. PWR-414
Warning: Design has unannotated sequential cell outputs. PWR-415

Report : power
        -analysis_effort low
Design : slaveuniquify
Version: J-2014.09-SP2
Date   : Mon Jul  6 00:11:47 2015


Librarys Used:

    c35_CORELIB_WC File: /home/komourtz/AMS/C35B4C3/synopsys/c35_3.3V/c35_CORELIB_WC.db
    c35_IOLIB_WC File: /home/komourtz/AMS/C35B4C3/synopsys/c35_3.3V/c35_IOLIB_WC.db


Operating Conditions: WORST-MIL   Library: c35_CORELIB_WC
Wire Load Model Mode: enclosed

Design        Wire Load Model            Library
------------------------------------------------
slaveuniquify          30k               c35_CORELIB_WC
ahb_slave_test_1       10k               c35_CORELIB_WC
ahb_slave_ram_test_1   10k               c35_CORELIB_WC
ahb_slave_mem_test_1   10k               c35_CORELIB_WC


Global Operating Voltage = 3
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    derived from V,C,T units
    Leakage Power Units = 1pW


  Cell Internal Power  = 44.437294 mW   62%
  Net Switching Power  = 27.132820 mW   38%
                         ---------
Total Dynamic Power    = 71.570114 mW  100%

Cell Leakage Power     =  24.0494 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power      %      Attrs
--------------------------------------------------------------------------------------------------
io_pad           5.515468         22.154610        3.092414e+06          27.673168
                                                                                  38.65%
memory           0.000000          0.000000           0.000000          0.000000    0.00%
black_box        0.000000          0.000000           0.000000          0.000000    0.00%
clock_network    0.000000          0.000000           0.000000          0.000000    0.00%
register        38.062965          1.714385        1.735926e+07          39.794746
                                                                                  55.58%
```

```
sequential       0.000000          0.000000          0.000000          0.000000      0.00%
combinational    0.858918          3.263807        3.597753e+06          4.126307
                                                                                     5.76%
--------------------------------------------------------------------------------------------------
Total           44.437351 mW      27.132803 mW     2.404943e+07 pW      71.594215 mW
1
```

QoR report

```
Report : qor
Design : slaveuniquify
Version: J-2014.09-SP2
Date   : Mon Jul  6 00:11:57 2015


  Timing Path Group 'CLK'
  -----------------------------------
  Levels of Logic:          10.000000
  Critical Path Length:     10.997951
  Critical Path Slack:       0.404160
  Critical Path Clk Period: 12.000000
  Total Negative Slack:      0.000000
  No. of Violating Paths:    0.000000
  Worst Hold Violation:      0.000000
  Total Hold Violation:      0.000000
  No. of Hold Violations:    0.000000
  -----------------------------------


  Cell Count
  -----------------------------------
  Hierarchical Cell Count:          3
  Hierarchical Port Count:        155
  Leaf Cell Count:               4342
  Buf/Inv Cell Count:             346
  Buf Cell Count:                 285
  Inv Cell Count:                  61
  CT Buf/Inv Cell Count:            0
  Combinational Cell Count:      2259
  Sequential Cell Count:         2083
  Macro Count:                      0
  -----------------------------------


  Area
  -----------------------------------
  Combinational Area:   171826.199623
  Noncombinational Area:
                        871197.612701
  Buf/Inv Area:          18054.399719
  Total Buffer Area:     15815.799629
  Total Inverter Area:    2238.600090
  Macro/Black Box Area:
                       1395640.000000
  Net Area:             161766.090611
  -----------------------------------
  Cell Area:           2438663.812325
  Design Area:         2600429.902935


  Design Rules
  -----------------------------------
  Total Number of Nets:          6447
  Nets With Violations:             0
  Max Trans Violations:             0
  Max Cap Violations:               0
  -----------------------------------


  Hostname: noyce

  Compile CPU Statistics
  -----------------------------------------
  Resource Sharing:                0.004318
  Logic Optimization:              0.366875
  Mapping Optimization:            24.889622
  -----------------------------------------
  Overall Compile Time:            28.064869
  Overall Compile Wall Clock Time: 67.053444

  --------------------------------------------------------------------

  Design  WNS: 0.000000  TNS: 0.000000  Number of Violating Paths: 0


  Design Hold  WNS: 0.000000  TNS: 0.000000  Number of Violating Paths: 0

  --------------------------------------------------------------------


1
```

Scan configuration report

```
Report : Scan configuration
Design : slaveuniquify
Version: J-2014.09-SP2
Date   : Mon Jul  6 00:11:47 2015

========================================
TEST MODE: all_dft
VIEW     : Specification
========================================
Chain count:                          1
Scan Style:                           Multiplexed flip-flop
Maximum scan chain length:            Undefined
Exact scan chain length:              Undefined
Physical Partitioning:                Horizontal
Replace:                              True
Preserve multibit segments:           False
Clock mixing:                         No mix
Internal clocks:                      none
Add lockup:                           True
Lockup type:                          latch
Insert terminal lockup:               False
Create dedicated scan out ports:      False
Shared scan in:                       0
Bidirectional mode:                   No bidirectional type
Internal Clock Mixing:                False
Test Clocks by System Clocks:         False
Hierarchical Isolation:               False
Multiple Scan Enable:                 Disable
Pipeline Scan Enable:                 Disable
Voltage Mixing:                       False
Identify Shift Register:              False
Power Domain Mixing:                  False
Reuse MV Isolation Cells:             True
Multi LSSD:                           Disable

1
```

Scan path report

```
Report : Scan path
Design : slaveuniquify
Version: J-2014.09-SP2
Date   : Mon Jul  6 00:11:49 2015

========================================
TEST MODE: Internal_scan
VIEW     : Existing DFT
========================================

========================================
AS SPECIFIED BY USER
========================================


========================================
AS BUILT BY insert_dft
========================================

Scan_path    Len   ScanDataIn  ScanDataOut ScanEnable  MasterClock SlaveClock
-----------  ----- ----------- ----------- ----------- ----------- -----------
I 1          2083  PI_Scan_In  PO_Scan_Out PI_Scan_enable PI_clk   -

1
```

Hold timing results

```
Report : timing
        -path full
        -delay min
        -nworst 10000
        -input_pins
        -max_paths 10000
Design : slaveuniquify
Version: J-2014.09-SP2
Date   : Mon Jul  6 00:11:54 2015

Operating Conditions: WORST-MIL   Library: c35_CORELIB_WC
Wire Load Model Mode: enclosed

  Startpoint: ahb_slave_inst/ahb_slave_ram/data_phase_reg
              rising edge-triggered flip-flop clocked by CLK
  Endpoint: ahb_slave_inst/ahb_slave_ram/error_reg
            rising edge-triggered flip-flop clocked by CLK
  Path Group: CLK
  Path Type: min

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  slaveuniquify      30k                   c35_CORELIB_WC
  ahb_slave_ram_test_1
                     10k                   c35_CORELIB_WC

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock CLK rise edge                                 0.000000   0.000000
  clock network delay ideal                           0.000000   0.000000
  ahb_slave_inst/ahb_slave_ram/data_phase_reg/C DFSC1
                                                        0.000000   0.000000 r
  ahb_slave_inst/ahb_slave_ram/data_phase_reg/Q DFSC1
                                                        1.220685   1.220685 r
  ahb_slave_inst/ahb_slave_ram/error_reg/SD DFSC1   0.000134   1.220819 r
  data arrival time                                              1.220819

  clock CLK rise edge                                 0.000000   0.000000
  clock network delay ideal                           0.000000   0.000000
  ahb_slave_inst/ahb_slave_ram/error_reg/C DFSC1    0.000000   0.000000 r
  library hold time                                     0.000000   0.000000
  data required time                                             0.000000
  --------------------------------------------------------------------------
  data required time                                             0.000000
  data arrival time                                              -1.220819
  --------------------------------------------------------------------------
  slack MET                                                    1.220819
```

Coverage estimation (part of)

```
In mode: Internal_scan...
  Design has scan chains in this mode
  Design is scan routed
  Post-DFT DRC enabled

Information: Starting test design rule checking. TEST-222
  Loading test protocol
  ...basic checks...
  ...basic sequential cell checks...
  ...checking vector rules...
  ...checking clock rules...
  ...checking scan chain rules...
  ...checking scan compression rules...
  ...checking X-state rules...
  ...checking tristate rules...
  ...extracting scan details...

-----------------------------------------------------------------
  DRC Report

  Total violations: 0

-----------------------------------------------------------------


Test Design rule checking did not find violations

-----------------------------------------------------------------
  Sequential Cell Report

  0 out of 2083 sequential cells have violations

-----------------------------------------------------------------

SEQUENTIAL CELLS WITHOUT VIOLATIONS
               ahb_slave_inst/ahb_slave_ram/counter_reg[0]
         ahb_slave_inst/ahb_slave_ram/HRESP_reg
         ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[7]
         ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[6]
         ahb_slave_inst/ahb_slave_ram/STALL_reg
         ahb_slave_inst/ahb_slave_ram/error_reg
         ahb_slave_inst/ahb_slave_ram/modulo_result_error_reg[1]
         ahb_slave_inst/ahb_slave_ram/modulo_result_error_reg[0]
         ahb_slave_inst/ahb_slave_ram/WR_pre_d_reg
         ahb_slave_inst/ahb_slave_ram/RD_pre_d_reg
         ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[7]
         ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[6]
         ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[5]
         ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[4]
         ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[3]
         ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[2]
         ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[1]
         ahb_slave_inst/ahb_slave_ram/ADDR_WR_reg[0]
         ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[5]
         ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[4]
         ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[3]
```

```
ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[1]
ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[0]
ahb_slave_inst/ahb_slave_ram/counter_reg[2]
ahb_slave_inst/ahb_slave_ram/counter_reg[1]
ahb_slave_inst/ahb_slave_ram/data_phase_reg
ahb_slave_inst/ahb_slave_ram/ADDR_RD_reg[2]
ahb_slave_inst/ahb_slave_mem/Mem_reg[255][7]
ahb_slave_inst/ahb_slave_mem/Mem_reg[255][6]
ahb_slave_inst/ahb_slave_mem/Mem_reg[255][5]
ahb_slave_inst/ahb_slave_mem/Mem_reg[255][4]
ahb_slave_inst/ahb_slave_mem/Mem_reg[255][3]
ahb_slave_inst/ahb_slave_mem/Mem_reg[255][2]
ahb_slave_inst/ahb_slave_mem/Mem_reg[255][1]
ahb_slave_inst/ahb_slave_mem/Mem_reg[255][0]
ahb_slave_inst/ahb_slave_mem/Mem_reg[254][7]
ahb_slave_inst/ahb_slave_mem/Mem_reg[254][6]
ahb_slave_inst/ahb_slave_mem/Mem_reg[254][5]
ahb_slave_inst/ahb_slave_mem/Mem_reg[254][4]
ahb_slave_inst/ahb_slave_mem/Mem_reg[254][3]
ahb_slave_inst/ahb_slave_mem/Mem_reg[254][2]
ahb_slave_inst/ahb_slave_mem/Mem_reg[254][1]
ahb_slave_inst/ahb_slave_mem/Mem_reg[254][0]
ahb_slave_inst/ahb_slave_mem/Mem_reg[253][7]
ahb_slave_inst/ahb_slave_mem/Mem_reg[253][6]
ahb_slave_inst/ahb_slave_mem/Mem_reg[253][5]
ahb_slave_inst/ahb_slave_mem/Mem_reg[253][4]
ahb_slave_inst/ahb_slave_mem/Mem_reg[253][3]
ahb_slave_inst/ahb_slave_mem/Mem_reg[253][2]
ahb_slave_inst/ahb_slave_mem/Mem_reg[253][1]
ahb_slave_inst/ahb_slave_mem/Mem_reg[253][0]
ahb_slave_inst/ahb_slave_mem/Mem_reg[252][7]
ahb_slave_inst/ahb_slave_mem/Mem_reg[252][6]
ahb_slave_inst/ahb_slave_mem/Mem_reg[252][5]
ahb_slave_inst/ahb_slave_mem/Mem_reg[252][4]
ahb_slave_inst/ahb_slave_mem/Mem_reg[252][3]
ahb_slave_inst/ahb_slave_mem/Mem_reg[252][2]
ahb_slave_inst/ahb_slave_mem/Mem_reg[252][1]
ahb_slave_inst/ahb_slave_mem/Mem_reg[252][0]
ahb_slave_inst/ahb_slave_mem/Mem_reg[251][7]
ahb_slave_inst/ahb_slave_mem/Mem_reg[251][6]
ahb_slave_inst/ahb_slave_mem/Mem_reg[251][5]
ahb_slave_inst/ahb_slave_mem/Mem_reg[251][4]
ahb_slave_inst/ahb_slave_mem/Mem_reg[251][3]
ahb_slave_inst/ahb_slave_mem/Mem_reg[251][2]
ahb_slave_inst/ahb_slave_mem/Mem_reg[251][1]
ahb_slave_inst/ahb_slave_mem/Mem_reg[251][0]
ahb_slave_inst/ahb_slave_mem/Mem_reg[250][7]
ahb_slave_inst/ahb_slave_mem/Mem_reg[250][6]
ahb_slave_inst/ahb_slave_mem/Mem_reg[250][5]
ahb_slave_inst/ahb_slave_mem/Mem_reg[250][4]
ahb_slave_inst/ahb_slave_mem/Mem_reg[250][3]
ahb_slave_inst/ahb_slave_mem/Mem_reg[250][2]
ahb_slave_inst/ahb_slave_mem/Mem_reg[250][1]
ahb_slave_inst/ahb_slave_mem/Mem_reg[250][0]
ahb_slave_inst/ahb_slave_mem/Mem_reg[249][7]
ahb_slave_inst/ahb_slave_mem/Mem_reg[249][6]
ahb_slave_inst/ahb_slave_mem/Mem_reg[249][5]
ahb_slave_inst/ahb_slave_mem/Mem_reg[249][4]
ahb_slave_inst/ahb_slave_mem/Mem_reg[249][3]
ahb_slave_inst/ahb_slave_mem/Mem_reg[249][2]
```

```
         ahb_slave_inst/ahb_slave_mem/Mem_reg[249][1]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[249][0]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[248][7]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[248][6]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[248][5]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[248][4]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[248][3]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[248][2]
                 ..............
                 ..............
                 ..............
                 ahb_slave_inst/ahb_slave_mem/Mem_reg[1][7]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[1][6]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[1][5]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[1][4]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[1][3]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[1][2]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[1][1]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[1][0]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[0][7]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[0][6]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[0][5]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[0][4]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[0][3]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[0][2]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[0][1]
         ahb_slave_inst/ahb_slave_mem/Mem_reg[0][0]
         ahb_slave_inst/ahb_slave_mem/DOUT_reg[7]
         ahb_slave_inst/ahb_slave_mem/DOUT_reg[6]
         ahb_slave_inst/ahb_slave_mem/DOUT_reg[5]
         ahb_slave_inst/ahb_slave_mem/DOUT_reg[4]
         ahb_slave_inst/ahb_slave_mem/DOUT_reg[3]
         ahb_slave_inst/ahb_slave_mem/DOUT_reg[2]
         ahb_slave_inst/ahb_slave_mem/DOUT_reg[1]
         ahb_slave_inst/ahb_slave_mem/DOUT_reg[0]

....Inferring feed-through connections....
Information: Test design rule checking completed. TEST-123
  Running test coverage estimation...
 47664 faults were added to fault list.
 ATPG performed for stuck fault model using internal pattern source.
 ----------------------------------------------------------
 #patterns     #faults     #ATPG faults  test      process
 stored     detect/active  red/au/abort  coverage  CPU time
 ---------  -------------  ------------  --------  --------
 Begin deterministic ATPG: #uncollapsed_faults=32804, abort_limit=10...
 0           10178  22626         0/0/0    52.52%      0.02
 0            3250  19376         0/0/0    59.34%      0.03
 0            2085  17291         0/0/0    63.72%      0.05
 0            1171  16120         0/0/0    66.17%      0.05
 0            1168  14952         0/0/0    68.62%      0.06
 0             928  14024         0/0/0    70.57%      0.06
 0             807  13217         0/0/0    72.26%      0.07
 0             799  12418         0/0/0    73.94%      0.07
 0             772  11646         0/0/0    75.56%      0.08
 0             627  11019         0/0/0    76.88%      0.08
 0             701  10318         0/0/0    78.35%      0.09
 0             600   9718         0/0/0    79.61%      0.09
 0             516   9202         0/0/0    80.69%      0.10
 0             604   8598         0/0/0    81.96%      0.10
```

```
0             531   8067         0/0/0    83.07%       0.11
0             415   7652         0/0/0    83.94%       0.11
0             416   7236         0/0/0    84.82%       0.12
0             437   6799         0/0/0    85.73%       0.12
0             377   6422         0/0/0    86.52%       0.13
0             418   6004         0/0/0    87.40%       0.13
0             412   5592         0/0/0    88.27%       0.13
0             431   5161         0/0/0    89.17%       0.14
0             334   4827         0/0/0    89.87%       0.14
0             361   4466         0/0/0    90.63%       0.15
0             323   4143         0/0/0    91.31%       0.15
0             311   3832         0/0/0    91.96%       0.15
0             256   3576         0/0/0    92.50%       0.16
0             289   3287         0/0/0    93.10%       0.16
0             235   3052         0/0/0    93.60%       0.17
0             193   2859         0/0/0    94.00%       0.17
0             251   2608         0/0/0    94.53%       0.17
0             222   2386         0/0/0    94.99%       0.18
0             168   2218         0/0/0    95.35%       0.18
0             170   2048         0/0/0    95.70%       0.19
0             212   1836         0/0/0    96.15%       0.19
0             137   1699         0/0/0    96.43%       0.19
0             139   1560         0/0/0    96.73%       0.20
0             147   1413         0/0/0    97.03%       0.20
0             114   1299         0/0/0    97.27%       0.20
0             112   1187         0/0/0    97.51%       0.21
0             123   1064         0/0/0    97.77%       0.21
0             136    928         0/0/0    98.05%       0.21
0             132    796         0/0/0    98.33%       0.22
0             124    672         0/0/0    98.59%       0.22
0             112    560         0/0/0    98.82%       0.23
0              81    479         0/0/0    98.99%       0.23
0              70    409         0/0/0    99.14%       0.23
0              77    332         0/0/0    99.30%       0.24
0              74    258         0/0/0    99.46%       0.24
0              59    199         0/0/0    99.58%       0.24
0              74    125         0/0/0    99.74%       0.25
0              65     60         0/0/0    99.87%       0.25
0              32     28         0/0/0    99.94%       0.25
0              28      0         0/0/0   100.00%       0.25

           Pattern Summary Report
-----------------------------------------------
#internal patterns                            0
-----------------------------------------------


    Uncollapsed Stuck Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT      47654
Possibly detected                PT          0
Undetectable                     UD         10
ATPG untestable                  AU          0
Not detected                     ND          0
-----------------------------------------------
total faults                             47664
test coverage                          100.00%
-----------------------------------------------
```

```
  Information: The test coverage above may be inferior
               than the real test coverage with customized
               protocol and test simulation library.
1
```

8.2 TetraMAX ATPG Results

TetraMAX ATPG summary

```
    Uncollapsed Stuck Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT      66900
  detected_by_simulation         DS     52014
  detected_by_implication        DI     14886
Possibly detected                PT          0
Undetectable                     UD         10
  undetectable-unused            UU        10
ATPG untestable                  AU          0
Not detected                     ND          0
-----------------------------------------------
total faults                             66910
test coverage                          100.00%
fault coverage                          99.99%
-----------------------------------------------
           Pattern Summary Report
-----------------------------------------------
#internal patterns                        1003
    #full_sequential patterns             1003
-----------------------------------------------
```

Undetected faults

```
sa0   UU   PI_HBURST[2]
sa1   UU   PI_HBURST[2]
sa0   UU   PI_HBURST[1]
sa1   UU   PI_HBURST[1]
sa0   UU   PI_HBURST[0]
sa1   UU   PI_HBURST[0]
sa0   UU   PI_HSIZE[1]
sa1   UU   PI_HSIZE[1]
sa0   UU   PI_HSIZE[0]
sa1   UU   PI_HSIZE[0]
```

Faults example

```
sa0   DI   PAD_scan_in/Y   ITP    1: 13440/0/0, SCOAP=1/1/2 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U42/B   AOI221    2: 13444/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U40/D   AOI221    3: 13473/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U49/B   AOI221    4: 13488/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[236][0]/Q   DFSE1    5: 13492/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U50/D   AOI221    6: 13496/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1512/D   AOI221    7: 13501/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U47/B   AOI221    8: 13519/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U48/B   AOI221    9: 13524/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U56/D   AOI221    10: 13530/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[222][0]/Q   DFSE1    11: 13531/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[220][0]/Q   DFSE1    12: 13536/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[219][0]/Q   DFSE1    13: 13539/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[213][0]/Q   DFSE1    14: 13555/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U53/B   AOI221    15: 13558/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[211][0]/Q   DFSE1    16: 13560/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U63/B   AOI221    17: 13576/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1518/D   AOI221    18: 13589/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U59/D   AOI221    19: 13595/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[198][0]/Q   DFSE1    20: 13596/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[183][0]/Q   DFSE1    21: 13640/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U39/D   AOI221    22: 13468/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1087/B   AOI221    23: 13648/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1089/D   AOI221    24: 13651/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U55/B   AOI221    25: 13568/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[175][0]/Q   DFSE1    26: 13663/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[170][0]/Q   DFSE1    27: 13675/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U123/B   AOI221    28: 13692/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U134/D   AOI221    29: 13713/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U135/B   AOI221    30: 13720/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U133/D   AOI221    31: 13723/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[255][0]/Q   DFSE1    32: 13441/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1082/D   AOI221    33: 13729/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U130/D   AOI221    34: 13739/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U122/B   AOI221    35: 13697/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U143/D   AOI221    36: 13762/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[138][0]/Q   DFSE1    37: 13763/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[137][0]/Q   DFSE1    38: 13766/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U141/B   AOI221    39: 13769/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U62/D   AOI221    40: 13610/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U60/D   AOI221    41: 13600/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1084/D   AOI221    42: 13773/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U139/D   AOI221    43: 13778/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[131][0]/Q   DFSE1    44: 13782/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U138/B   AOI221    45: 13785/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1077/B   AOI221    46: 13801/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1078/D   AOI221    47: 13804/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[119][0]/Q   DFSE1    48: 13819/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1072/B   AOI221    49: 13822/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U131/B   AOI221    50: 13736/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1073/D   AOI221    51: 13825/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[116][0]/Q   DFSE1    52: 13826/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1074/D   AOI221    53: 13830/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1058/D   AOI221    54: 13843/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U99/D   AOI221    55: 13853/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[106][0]/Q   DFSE1    56: 13854/1/0, SCOAP=1/1/35 0/0/0/0
```

```
sa0   DS   ahb_slave_inst/ahb_slave_mem/U98/B   AOI221    57: 13860/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1057/D   AOI221    58: 13864/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[254][0]/Q   DFSE1    59: 13443/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[100][0]/Q   DFSE1    60: 13870/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U95/D   AOI221    61: 13874/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[98][0]/Q   DFSE1    62: 13875/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U108/D   AOI221    63: 13892/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U107/D   AOI221    64: 13897/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[166][0]/Q   DFSE1    65: 13686/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[91][0]/Q   DFSE1    66: 13896/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1060/B   AOI221    67: 13889/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[90][0]/Q   DFSE1    68: 13898/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U106/D   AOI221    69: 13902/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[88][0]/Q   DFSE1    70: 13903/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[96][0]/Q   DFSE1    71: 13880/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1059/D   AOI221    72: 13908/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U103/B   AOI221    73: 13920/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U102/D   AOI221    74: 13923/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U96/D   AOI221    75: 13869/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1091/B   AOI221    76: 13622/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[152][0]/Q   DFSE1    77: 13724/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[184][0]/Q   DFSE1    78: 13636/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1092/B   AOI221    79: 13627/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[241][0]/Q   DFSE1    80: 13477/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1070/D   AOI221    81: 13931/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[79][0]/Q   DFSE1    82: 13930/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[78][0]/Q   DFSE1    83: 13932/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U115/D   AOI221    84: 13936/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U41/B   AOI221    85: 13480/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[76][0]/Q   DFSE1    86: 13937/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[75][0]/Q   DFSE1    87: 13940/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U114/D   AOI221    88: 13946/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1066/D   AOI221    89: 13952/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U142/D   AOI221    90: 13757/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[165][0]/Q   DFSE1    91: 13689/1/0, SCOAP=1/1/35 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U112/D   AOI221    92: 13957/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U146/D   AOI221    93: 13635/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U1085/B   AOI221    94: 13754/0/1, SCOAP=1/5/31 0/0/0/0
sa0   DS   ahb_slave_inst/ahb_slave_mem/U111/D   AOI221    95: 13962/0/1, SCOAP=1/5/31 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[186][0]/Q   DFSE1    96: 13631/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[234][0]/Q   DFSE1    97: 13497/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[67][0]/Q   DFSE1    98: 13961/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[66][0]/Q   DFSE1    99: 13963/1/0, SCOAP=1/1/35 0/0/0/0
sa1   DS   ahb_slave_inst/ahb_slave_mem/\Mem_reg[240][0]/Q   DFSE1    100: 13479/1/0, SCOAP=1/1/35 0/0/0/0
```

Pattern example

```
Pattern 0 full_sequential
Time 0: period = 100
Time 0: load 1 =
    0010111010 0101001111 1000110001 0001110100 0101011111 1100101111 1100101111
    0101101011 0101010100 0111100100 0101100001 0010110111 0101110001 1101101001
    0101110100 0111000100 0101100101 0111010011 1001000110 0011011000 1000010010
    1101010101 0100100010 0101101001 0101011110 1100011101 0011001110 0101001001
    0111100100 1110001101 1011100001 1110101000 1011000110 1101010111 0110001010
    0011110001 0010010100 0110100110 0100000101 0001001100 1010111010 1011100111
    0001110110 0100100001 0001110100 1011101111 1110100111 0100101111 1000101010
    1111001011 0101001010 1100001101 0010110000 1101101110 1100111100 0010001001
    0001011110 1101100101 0111110110 0001110011 0111100110 0110000110 1101001100
    0110000011 0101010110 1011111101 0100100110 1001001001 1101100000 0001010010
    1101000110 0001000001 1010001010 0011000111 1100010010 1001010111 1011010011
    1010110010 0001110001 0011001101 1001001001 1101100110 0000101000 0001000101
    0100101111 0010010111 0110101000 0001111101 0101010100 1001110011 0010100111
    0101001100 0010011000 0110101111 0110011100 1010111001 0000011000 0111010100
    0011010101 0001011101 1111101101 0011000101 0001010100 0110011011 0011011010
    1110100010 1110011100 1001011111 0000100111 1101110110 1101000101 1101010011
    0001111110 1010010010 1011001000 0100110101 1101110111 0110110101 0001001110
    1100110110 0011010010 0101010110 1111011000 1101101101 1001101100 1100100010
    0010110110 1000001011 0111101011 1001101011 1011011100 1000010011 0101100001
    0010010010 1000110101 0010010000 1110111110 1101111100 1011010001 1001000110
    1101100101 0101011100 1110010100 0100010001 0100011100 1100100111 0100101111
    0100101001 0111100100 1111101100 0010110100 0000011110 1011111111 0000010101
    0101001101 1001000111 0101010000 0000111101 1101000110 0111010111 0011100011
    0100001001 0101110100 1001010100 0100000101 0000100000 1010010010 0001111011
    0000100110 0111000110 1000100011 1000100101 0000100001 0111110001 0001110011
    1011010101 0101010100 0100101111 0011000010 0101010000 0010011101 0000001101
    1111101100 0010110101 0111111010 1110101010 1011011111 0100110010 1101110111
    1000010011 0000101110 1010110101 0101011000 0011000010 0001101111 0001100001
    0011010110 0101000101 1110010000 0001101010 1011100010 0101110101 0100111111
    1100010010 1001011101 1011100001 1100100100 0110001010 010
Time 100: force_all_pis = 1000111010 1001111011 1100001001
Time 140: measure_all_pos =1000000001 0
Time 145: pulse clocks  PI_clk 2
Time 200: unload 1 =
    1110001100 1100000010 1010101110 1110001011 1010100000 0011010000 0011010000
    1010010100 1010101011 1000011011 1010011110 1101001000 1010001110 0010010110
    1010001011 1000111011 1010011010 1000101100 0110111001 1100100111 0111101101
    0010101010 1011011101 1010010110 1010100001 0011100010 1100110001 1010110110
    1000011011 0001110010 0100011110 0001010111 0100111001 0010101000 1001110101
    1100001110 1101101011 1001011001 1011111010 1110110011 0101000101 0100011000
    1110001001 1011011110 1110001011 0100010000 0001011000 1011010000 0111010101
    0000110100 1010110101 0011110010 1101001111 0010010001 0011000011 1101110110
    1110100001 0010011010 1000001001 1110001100 1000011001 1001111001 0010110011
    1001111100 1010101001 0100000010 1011011001 0110110110 0010011111 1110101101
    0010111001 1110111110 0101110101 1100111000 0011101101 0110101000 0100101100
    0101001101 1110001110 1100110010 0110110110 0010011001 1111010111 1110111010
    1011010000 1101101000 1001010111 1110000010 1010101011 0110001100 1101011000
    1010110011 1101100111 1001010000 1001100011 0101000110 1111100111 1000101011
    1100101010 1110100010 0000010010 1100111010 1110101011 1001100100 1100100101
    0001011101 0001100011 0110100000 1111011000 0010001001 0010111010 0010101100
    1110000001 0101101101 0100110111 1011001010 0010001000 1001001010 1110110001
    0011001001 1100101101 1010101001 0000100111 0010010010 0110010011 0011011101
    1101001001 0111110100 1000010100 0110010100 0100100011 0111101100 1010011110
```

```
1101101101 0111001010 1101101111 0001000001 0010000011 0100101110 0110111001
0010011010 1010100011 0001101011 1011101110 1011100011 0011011000 1011010000
1011010110 1000011011 0000010011 1101001011 1111100001 0100000000 1111101010
1010110010 0110111000 1010101111 1111000010 0010111001 1000101000 1100011100
1011110110 1010001011 0110101011 1011111010 1111011111 0101101101 1110000100
1111011001 1000111001 0111011100 0111011010 1111011110 1000001110 1110001100
0100101010 1010101011 1011010000 1100111101 1010101111 1101100010 1111110010
0000010011 1101001010 1000000101 0001010101 0100100000 1011001101 0010001000
0111101100 1111010001 0101001010 1010100111 1100111101 1110010000 1110011110
1101101000 1010111010 0001101111 1110010101 0100011101 1010001010 1011000000
0011101101 0110100010 0100011110 0011011011 1001111110 001
```

8.3 SoC Encounter Results

Gate Count

```
Gate area 54.6000 um^2
Level 0 Module slaveuniquify                        Gates=     22742 Cells=    5756 Area= 1241731.4 um^2
Level 1 Module ahb_slave_inst                       Gates=19442      Cells=4497     Area= 1061533.2 um^2
Level 2 Module ahb_slave_inst/ahb_slave_ram         Gates=347        Cells=159      Area=   18964.4 um^2
Level 2 Module ahb_slave_inst/ahb_slave_mem         Gates=19036      Cells=4296     Area= 1039402.0 um^2
```

Verify DRC

```
################################################################
#  Generated by:      Cadence Encounter 14.13-s036_1
#  OS:                Linux x86_64Host ID noyce
#  Generated on:      Fri Jul 10 00:10:16 2015
#  Design:            slaveuniquify
#  Command:           verify_drc -report reports/slaveuniquify.drc.rpt -limit 1000
################################################################

No DRC violations were found
```

Verify connectivity

```
###############################################################
#  Generated by:      Cadence Encounter 14.13-s036_1
#  OS:                Linux x86_64Host ID noyce
#  Generated on:      Fri Jul 10 00:10:19 2015
#  Design:            slaveuniquify
#  Command:           verifyConnectivity -type all -report reports/slaveuniquify.conn.rpt -error 1000 -warning 
###############################################################
Verify Connectivity Report is created on Fri Jul 10 00:10:19 2015




Begin Summary
    Found no problems or warnings.
End Summary
```

Power Report

```
*----------------------------------------------------------------
*       Encounter 14.13-s036_1 (64bit) 08/14/2014 18:19 (Linux 2.6)
*
*
*       Date & Time:    2015-Jul-10 00:10:11 (2015-Jul-09 21:10:11 GMT)
*
*----------------------------------------------------------------
*
*       Design: slaveuniquify
*
*       Liberty Libraries used:
*               ../../AMS/C35B4C3/liberty/c35_3.3V/c35_CORELIB_WC.lib
*               ../../AMS/C35B4C3/liberty/c35_3.3V/c35_IOLIB_WC.lib
*
*       Power Domain used:
*               Rail:      vdd!      Voltage:       3
*               Rail:    vdd3r1!     Voltage:    2.97
*               Rail:    vdd3r2!     Voltage:    2.97
*               Rail:     vdd3o!     Voltage:    2.97
*
*       Power View : slow
*
*       User-Defined Activity : N.A.
*
*       Activity File: N.A.
*
*       Hierarchical Global Activity: N.A.
*
*       Global Activity: N.A.
*
*       Sequential Element Activity: N.A.
*
*       Primary Input Activity: 0.500000
*
*       Default icg ratio: N.A.
*
*       Global Comb ClockGate Ratio: N.A.
*
*       Power Units = 1mW
*
*       Time Units = 1e-09 secs
*
*       report_power -outfile reports/power.rpt -sort total -clock_network all
*
-----------------------------------------------------------------

Clock                            Internal   Switching    Leakage     Total   Percentage
                                 Power      Power        Power       Power   (%)
-----------------------------------------------------------------
CLK                                   151       42.2    0.006638     193.2        54.84
-----------------------------------------------------------------
Total (excluding duplicates)          151       42.2    0.006638     193.2        54.84
-----------------------------------------------------------------


Instance Power Report for CLK clock network

Instance                       Max Toggles   Total Toggles   Internal   Switching    Leakage
Total   Percentage     Cell
                               Input Pins    Output Pins     Power      Power        Power
Power   (%)            Name
-------------------------------------------------------------------------------------------
PAD_clk                          1.613e+08       1.613e+08      2.229      0.2645  7.622e-05
2.493        0.7078    ITCK4P
clk__L3_I8                       1.613e+08       1.613e+08     0.1153      0.2067  5.383e-06
0.322        0.0914    BUF15
```

```
clk__L3_I12                       1.613e+08     1.613e+08     0.1154     0.2032   5.383e-06
0.3187    0.09046    BUF15
clk__L3_I9                        1.613e+08     1.613e+08     0.1145     0.1947    5.36e-06
0.3092    0.08779    CLKBU15
clk__L3_I2                        1.613e+08     1.613e+08     0.1157     0.1932   5.383e-06
0.3089     0.0877    BUF15
clk__L3_I30                       1.613e+08     1.613e+08     0.1157     0.1896   5.383e-06
0.3053    0.08666    BUF15
clk__L3_I13                       1.613e+08     1.613e+08     0.1149     0.1844    5.36e-06
0.2992    0.08494    CLKBU15
clk__L3_I25                       1.613e+08     1.613e+08      0.116      0.178   5.383e-06
0.294    0.08347    BUF15
clk__L3_I28                       1.613e+08     1.613e+08     0.1163     0.1721   5.383e-06
0.2885    0.08189    BUF15
clk__L3_I15                       1.613e+08     1.613e+08     0.1152     0.1729    5.36e-06
0.2881    0.08178    CLKBU15
clk__L3_I10                       1.613e+08     1.613e+08     0.1161     0.1691   5.383e-06
0.2852    0.08096    BUF15
clk__L3_I6                        1.613e+08     1.613e+08     0.1162     0.1686   5.383e-06
0.2848    0.08085    BUF15
clk__L3_I4                        1.613e+08     1.613e+08     0.1162     0.1673   5.383e-06
0.2835    0.08048    BUF15
clk__L3_I7                        1.613e+08     1.613e+08     0.1163     0.1665   5.383e-06
0.2828    0.08028    BUF15
clk__L3_I17                       1.613e+08     1.613e+08     0.1162     0.1654   5.383e-06
0.2816    0.07995    BUF15
clk__L3_I22                       1.613e+08     1.613e+08     0.1166     0.1636   5.383e-06
0.2802    0.07954    BUF15
clk__L3_I1                        1.613e+08     1.613e+08     0.1164     0.1631   5.383e-06
0.2796    0.07936    BUF15
clk__L3_I21                       1.613e+08     1.613e+08     0.1166     0.1625   5.383e-06
0.2792    0.07925    BUF15
clk__L3_I31                       1.613e+08     1.613e+08     0.1163     0.1627   5.383e-06
0.2789    0.07918    BUF15
clk__L3_I5                        1.613e+08     1.613e+08     0.1155     0.1619    5.36e-06
0.2774    0.07874    CLKBU15
clk__L3_I29                       1.613e+08     1.613e+08     0.1166       0.16   5.383e-06
0.2766    0.07852    BUF15
clk__L3_I20                       1.613e+08     1.613e+08     0.1168      0.156   5.383e-06
0.2729    0.07746    BUF15
clk__L3_I27                       1.613e+08     1.613e+08     0.1168     0.1534   5.383e-06
0.2702    0.07669    BUF15
clk__L3_I26                       1.613e+08     1.613e+08     0.1168     0.1513   5.383e-06
0.2681    0.07611    BUF15
clk__L3_I23                       1.613e+08     1.613e+08     0.1159     0.1505    5.36e-06
0.2665    0.07565    CLKBU15
clk__L3_I3                        1.613e+08     1.613e+08     0.1168      0.145   5.383e-06
0.2619    0.07434    BUF15
clk__L3_I11                       1.613e+08     1.613e+08     0.1159     0.1424    5.36e-06
0.2584    0.07334    CLKBU15
clk__L4_I82                       1.613e+08     1.613e+08     0.1176     0.1403   5.383e-06
0.2578    0.07319    BUF15
clk__L3_I19                       1.613e+08     1.613e+08     0.1171     0.1406   5.383e-06
0.2577    0.07316    BUF15
clk__L3_I0                        1.613e+08     1.613e+08     0.1169     0.1404   5.383e-06
0.2574    0.07306    BUF15
clk__L4_I247                      1.613e+08     1.613e+08     0.1175     0.1378   5.383e-06
0.2553    0.07248    BUF15
clk__L4_I162                      1.613e+08     1.613e+08     0.1176     0.1367   5.383e-06
0.2543    0.07219    BUF15
clk__L4_I56                       1.613e+08     1.613e+08     0.1176     0.1359   5.383e-06
0.2535    0.07198    BUF15
clk__L4_I246                      1.613e+08     1.613e+08     0.1176      0.135   5.383e-06
0.2526    0.07171    BUF15
...................
....................
...................
clk__L5_I105                      1.613e+08     1.613e+08     0.1227    0.01029   5.383e-06
0.133    0.03776    BUF15
```

```
clk__L5_I634                          1.613e+08     1.613e+08     0.1239     0.009141    5.383e-06
0.133     0.03776     BUF15
clk__L5_I554                          1.613e+08     1.613e+08     0.1233     0.009682    5.383e-06
0.133     0.03775     BUF15
clk__L5_I563                          1.613e+08     1.613e+08     0.1237     0.009298    5.383e-06
0.133     0.03775     BUF15
clk__L5_I186                          1.613e+08     1.613e+08     0.1242      0.00881    5.383e-06
0.133     0.03775     BUF15
clk__L5_I833                          1.613e+08     1.613e+08     0.1241      0.00882    5.383e-06
0.133     0.03775     BUF15
clk__L5_I694                          1.613e+08     1.613e+08     0.1234      0.00959    5.383e-06
0.133     0.03775     BUF15
clk__L5_I226                          1.613e+08     1.613e+08      0.124     0.008946    5.383e-06
0.133     0.03775     BUF15
clk__L5_I370                          1.613e+08     1.613e+08     0.1235     0.009417    5.383e-06
0.133     0.03774     BUF15
clk__L5_I638                          1.613e+08     1.613e+08     0.1237     0.009224    5.383e-06
0.1329     0.03774     BUF15
clk__L5_I846                          1.613e+08     1.613e+08     0.1236     0.009314    5.383e-06
0.1329     0.03774     BUF15
clk__L5_I108                          1.613e+08     1.613e+08     0.1233     0.009604    5.383e-06
0.1329     0.03773     BUF15
clk__L5_I228                          1.613e+08     1.613e+08      0.124     0.008899    5.383e-06
0.1329     0.03773     BUF15
clk__L5_I678                          1.613e+08     1.613e+08     0.1239     0.009046    5.383e-06
0.1329     0.03773     BUF15
clk__L5_I247                          1.613e+08     1.613e+08     0.1232     0.009734    5.383e-06
0.1329     0.03773     BUF15
clk__L5_I330                          1.613e+08     1.613e+08     0.1238     0.009082    5.383e-06
0.1329     0.03773     BUF15
clk__L5_I780                          1.613e+08     1.613e+08     0.1239     0.008964    5.383e-06
0.1329     0.03773     BUF15
clk__L5_I170                          1.613e+08     1.613e+08     0.1237     0.009185    5.383e-06
0.1329     0.03773     BUF15
clk__L5_I664                          1.613e+08     1.613e+08     0.1237     0.009205    5.383e-06
0.1329     0.03772     BUF15
clk__L5_I519                          1.613e+08     1.613e+08     0.1235     0.009348    5.383e-06
0.1329     0.03772     BUF15
clk__L5_I281                          1.613e+08     1.613e+08     0.1233     0.009531    5.383e-06
0.1328     0.03771     BUF15
clk__L5_I214                          1.613e+08     1.613e+08     0.1243     0.008521    5.383e-06
0.1328     0.03771     BUF15
clk__L5_I75                           1.613e+08     1.613e+08     0.1233     0.009476    5.383e-06
0.1328      0.0377     BUF15
clk__L5_I896                          1.613e+08     1.613e+08     0.1236     0.009169    5.383e-06
0.1328      0.0377     BUF15
clk__L5_I728                          1.613e+08     1.613e+08     0.1232     0.009613    5.383e-06
0.1328      0.0377     BUF15
clk__L5_I432                          1.613e+08     1.613e+08     0.1234     0.009359    5.383e-06
0.1328      0.0377     BUF15
clk__L5_I231                          1.613e+08     1.613e+08     0.1238     0.008979    5.383e-06
0.1328     0.03769     BUF15
clk__L5_I250                          1.613e+08     1.613e+08     0.1235     0.009299    5.383e-06
0.1328     0.03769     BUF15
clk__L5_I651                          1.613e+08     1.613e+08     0.1237     0.009035    5.383e-06
0.1327     0.03768     BUF15
clk__L5_I87                           1.613e+08     1.613e+08     0.1236     0.009142    5.383e-06
0.1327     0.03768     BUF15
clk__L5_I150                          1.613e+08     1.613e+08     0.1239     0.008816    5.383e-06
0.1327     0.03767     BUF15
clk__L5_I827                          1.613e+08     1.613e+08     0.1239     0.008762    5.383e-06
0.1327     0.03767     BUF15
clk__L5_I668                          1.613e+08     1.613e+08     0.1232     0.009462    5.383e-06
0.1327     0.03767     BUF15
clk__L5_I624                          1.613e+08     1.613e+08     0.1234     0.009307    5.383e-06
0.1327     0.03767     BUF15
clk__L5_I702                          1.613e+08     1.613e+08      0.123     0.009696    5.383e-06
0.1327     0.03767     BUF15
clk__L5_I573                          1.613e+08     1.613e+08     0.1236     0.009046    5.383e-06
0.1327     0.03766     BUF15
```

```
clk__L5_I723                         1.613e+08      1.613e+08      0.1229     0.009713    5.383e-06
0.1327    0.03766    BUF15
clk__L5_I542                         1.613e+08      1.613e+08      0.1234     0.009225    5.383e-06
0.1327    0.03766    BUF15
clk__L5_I374                         1.613e+08      1.613e+08      0.1237     0.008975    5.383e-06
0.1326    0.03765    BUF15
clk__L5_I391                         1.613e+08      1.613e+08      0.1239     0.008708    5.383e-06
0.1326    0.03765    BUF15
clk__L5_I259                         1.613e+08      1.613e+08      0.1241     0.008458    5.383e-06
0.1326    0.03764    BUF15
clk__L5_I230                         1.613e+08      1.613e+08      0.1238     0.008776    5.383e-06
0.1326    0.03764    BUF15
clk__L5_I781                         1.613e+08      1.613e+08       0.124     0.008625    5.383e-06
0.1326    0.03764    BUF15
clk__L5_I135                         1.613e+08      1.613e+08      0.1239       0.0087    5.383e-06
0.1326    0.03764    BUF15
clk__L5_I556                         1.613e+08      1.613e+08      0.1233     0.009285    5.383e-06
0.1326    0.03764    BUF15
clk__L5_I444                         1.613e+08      1.613e+08      0.1227      0.00987    5.383e-06
0.1326    0.03763    BUF15
clk__L5_I604                         1.613e+08      1.613e+08      0.1233     0.009257    5.383e-06
0.1326    0.03763    BUF15
clk__L5_I603                         1.613e+08      1.613e+08      0.1236     0.008963    5.383e-06
0.1325    0.03762    BUF15
clk__L5_I242                         1.613e+08      1.613e+08      0.1239     0.008617    5.383e-06
0.1325    0.03762    BUF15
clk__L5_I606                         1.613e+08      1.613e+08      0.1233     0.009221    5.383e-06
0.1325    0.03762    BUF15
clk__L5_I453                         1.613e+08      1.613e+08      0.1234     0.009116    5.383e-06
0.1325    0.03762    BUF15
clk__L5_I363                         1.613e+08      1.613e+08      0.1235     0.009015    5.383e-06
0.1325    0.03761    BUF15
clk__L5_I103                         1.613e+08      1.613e+08      0.1228     0.009712    5.383e-06
0.1325    0.03761    BUF15
clk__L5_I354                         1.613e+08      1.613e+08      0.1236     0.008825    5.383e-06
0.1325    0.03761    BUF15
clk__L5_I13                          1.613e+08      1.613e+08      0.1228     0.009633    5.383e-06
0.1325     0.0376    BUF15
clk__L5_I830                         1.613e+08      1.613e+08       0.124     0.008482    5.383e-06
0.1325     0.0376    BUF15
clk__L5_I31                          1.613e+08      1.613e+08      0.1228     0.009602    5.383e-06
0.1324     0.0376    BUF15
clk__L5_I62                          1.613e+08      1.613e+08      0.1236     0.008789    5.383e-06
0.1324    0.03759    BUF15
clk__L5_I735                         1.613e+08      1.613e+08      0.1236     0.008852    5.383e-06
0.1324    0.03759    BUF15
clk__L5_I371                         1.613e+08      1.613e+08      0.1236     0.008805    5.383e-06
0.1324    0.03759    BUF15
clk__L5_I869                         1.613e+08      1.613e+08      0.1236      0.00885    5.383e-06
0.1324    0.03759    BUF15
clk__L5_I367                         1.613e+08      1.613e+08      0.1235     0.008872    5.383e-06
0.1324    0.03758    BUF15
clk__L5_I612                         1.613e+08      1.613e+08      0.1235     0.008816    5.383e-06
0.1324    0.03757    BUF15
clk__L5_I37                          1.613e+08      1.613e+08      0.1239     0.008488    5.383e-06
0.1324    0.03757    BUF15
clk__L5_I472                         1.613e+08      1.613e+08      0.1229     0.009458    5.383e-06
0.1323    0.03757    BUF15
clk__L5_I497                         1.613e+08      1.613e+08      0.1241     0.008211    5.383e-06
0.1323    0.03757    BUF15
clk__L5_I630                         1.613e+08      1.613e+08      0.1236     0.008753    5.383e-06
0.1323    0.03757    BUF15
clk__L5_I11                          1.613e+08      1.613e+08      0.1235     0.008841    5.383e-06
0.1323    0.03756    BUF15
clk__L5_I508                         1.613e+08      1.613e+08      0.1241     0.008212    5.383e-06
0.1323    0.03755    BUF15
clk__L5_I134                         1.613e+08      1.613e+08      0.1239     0.008323    5.383e-06
0.1323    0.03754    BUF15
clk__L5_I340                         1.613e+08      1.613e+08      0.1236     0.008614    5.383e-06
0.1322    0.03754    BUF15
```

```
clk__L5_I826                            1.613e+08       1.613e+08      0.1236      0.008564     5.383e-06
0.1322      0.03753     BUF15
clk__L5_I445                            1.613e+08       1.613e+08      0.1227      0.009485     5.383e-06
0.1322      0.03753     BUF15
clk__L5_I113                            1.613e+08       1.613e+08      0.1236      0.008528     5.383e-06
0.1322      0.03752     BUF15
clk__L5_I114                            1.613e+08       1.613e+08      0.1236       0.00852     5.383e-06
0.1322      0.03752     BUF15
clk__L5_I743                            1.613e+08       1.613e+08      0.1237       0.00838     5.383e-06
0.1321      0.03751     BUF15
clk__L5_I416                            1.613e+08       1.613e+08      0.1235      0.008665     5.383e-06
0.1321      0.03751     BUF15
clk__L5_I734                            1.613e+08       1.613e+08      0.1235      0.008637     5.383e-06
0.1321      0.03751     BUF15
clk__L5_I65                             1.613e+08       1.613e+08      0.1239      0.008217     5.383e-06
0.1321       0.0375     BUF15
clk__L5_I659                            1.613e+08       1.613e+08      0.1235      0.008558     5.383e-06
0.1321       0.0375     BUF15
clk__L5_I610                            1.613e+08       1.613e+08      0.1232      0.008826     5.383e-06
0.1321      0.03749     BUF15
clk__L5_I400                            1.613e+08       1.613e+08       0.123      0.009031     5.383e-06
0.132      0.03748     BUF15
clk__L5_I294                            1.613e+08       1.613e+08      0.1223      0.009656     5.383e-06
0.1319      0.03745     BUF15
clk__L5_I74                             1.613e+08       1.613e+08      0.1231       0.00875     5.383e-06
0.1319      0.03744     BUF15
clk__L5_I43                             1.613e+08       1.613e+08      0.1235       0.00824     5.383e-06
0.1318       0.0374     BUF15
clk__L5_I498                            1.613e+08       1.613e+08      0.1228      0.008894     5.383e-06
0.1317      0.03738     BUF15
clk__L5_I464                            1.613e+08       1.613e+08      0.1226      0.009022     5.383e-06
0.1317      0.03738     BUF15
clk__L5_I882                            1.613e+08       1.613e+08      0.1231      0.008512     5.383e-06
0.1316      0.03737     BUF15
clk__L5_I19                             1.613e+08       1.613e+08      0.1234       0.00818     5.383e-06
0.1316      0.03735     BUF15
clk__L5_I423                            1.613e+08       1.613e+08       0.123      0.008499     5.383e-06
0.1315      0.03732     BUF15
-----------------------------------------------------------------------------------------------------------
Total                                                                    151          42.2      0.006638
193.2        54.84


Master Clock Power including generated clocks
------------------------------------------------------------------------------------------
CLK has total 1220 instances
generated clocks:
                             151         42.2     0.006638       193.2       54.84%
------------------------------------------------------------------------------------------
Clock network has total 1220 instances
                             151         42.2     0.006638       193.2       54.84%


------------------------------------------------------------------------------------------------------
*       Power Distribution Summary:
*               Highest Average Power:                 PAD_HREADY (BU16P):            4.004
*               Highest Leakage Power:                PAD_scan_out (BU16P):        7.944e-05
*               Total Cap:     3.73121e-10 F
*               Total instances in design:  5797
*               Total instances in design with no power:     0
*          Total instances in design with no activity:     0
*               Total Fillers and Decap:     0
------------------------------------------------------------------------------------------------------
```

Post-route Hold summary

```
######################################################################
# Generated by:      Cadence Encounter 14.13-s036_1
# OS:                Linux x86_64(Host ID noyce)
# Generated on:      Fri Jul 10 00:04:28 2015
# Design:            slaveuniquify
# Command:           optDesign -postRoute -hold
######################################################################

------------------------------------------------------------------
     optDesign Final Non-SI Timing Summary
------------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.000  |  0.259  |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    1    |    0    |    1    |
|          All Paths:|  8341   |  4173   |  6251   |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |    0 (0)         |   0.000    |     0 (0)        |
|   max_tran     |    0 (0)         |   0.000    |     0 (0)        |
|   max_fanout   |    0 (0)         |     0      |     0 (0)        |
|   max_length   |    0 (0)         |     0      |     0 (0)        |
+----------------+------------------+------------+------------------+

Density: 42.125%
------------------------------------------------------------------
```

Power results Final

```
-----------------------------------------------------------------------------------------


Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:      213.51326128            76.0291%
Total Switching Power:      67.28797235            23.9603%
Total Leakage Power:         0.02967489             0.0106%
Total Power:               280.83090876
-----------------------------------------------------------------------------------------


Group                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  %
-----------------------------------------------------------------------------------------
Sequential                         40.98       5.447     0.01736       46.45       16.54
Macro                                  0           0           0           0           0
IO                                 18.21      0.9083    0.003109       19.12       6.808
Combinational                      5.559       18.99    0.002641       24.56       8.744
Clock Combinational              148.8       41.94    0.006562       190.7       67.91
Clock Sequential                     0           0           0           0           0
-----------------------------------------------------------------------------------------
Total                              213.5       67.29     0.02967       280.8         100
-----------------------------------------------------------------------------------------


Rail                    Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  %
-----------------------------------------------------------------------------------------
vdd!                          3      195.3       66.38     0.02657       261.7       93.19
vdd3r1!                    2.97      6.069      0.3028    0.001036       6.373       2.269
vdd3r2!                    2.97      6.069      0.3028    0.001036       6.373       2.269
vdd3o!                     2.97      6.069      0.3028    0.001036       6.373       2.269


Clock                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  %
-----------------------------------------------------------------------------------------
CLK                                  151        42.2    0.006638       193.2        68.8
-----------------------------------------------------------------------------------------
Total excluding duplicates         151        42.2    0.006638       193.2        68.8
-----------------------------------------------------------------------------------------



-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------

Total leakage power = 0.0296749 mW
 Cell usage statistics:
 Library c35_IOLIB_WC , 41 cells  0.722085% , 0.00310946 mW  10.478410%
 Library c35_CORELIB_WC , 5637 cells  99.277915% , 0.0265654 mW  89.521590%
```

8.4 Tempus STA Results

Setup timing results Final

```
###############################################################
#  Generated by:      Cadence Tempus 14.13-s037_1
#  OS:                Linux x86_64Host ID noyce
#  Generated on:      Thu Jul  9 23:37:22 2015
#  Design:            slaveuniquify
#  Command:           report_critical_instance -max_insts 20 -max_slack 1 -cost_type worst_slack > slaveuniquif
###############################################################

# -----------------------------------------------------------------------------------------
# Report Critical Instance:
# Slack: max slack = 1
#        only count the paths and end points whose slack is worse than 1
# Cost:  worst slack.
# Date:  Thu Jul 09 23:37:24 EEST 2015
# -----------------------------------------------------------------------------
# Inst:combinatorial                           Cell             Cost
# -----------------------------------------------------------------------------
ahb_slave_inst/ahb_slave_mem/FE_OCPC471_n5911    CLKBU2           0.188
ahb_slave_inst/ahb_slave_mem/FE_OFC340_n4382     CLKBU2           0.188
ahb_slave_inst/ahb_slave_mem/FE_OFC329_n4382     BUF4             0.188
ahb_slave_inst/ahb_slave_mem/U688                OAI211           0.188
ahb_slave_inst/ahb_slave_mem/U323                NAND40           0.188
ahb_slave_inst/ahb_slave_mem/U325                AOI220           0.188
ahb_slave_inst/ahb_slave_mem/U972                CLKIN2           0.188
ahb_slave_inst/ahb_slave_mem/U979                NOR21            0.188
ahb_slave_inst/ahb_slave_mem/U1008               CLKIN3           0.188
ahb_slave_inst/ahb_slave_mem/U1009               NAND22           0.188
ahb_slave_inst/ahb_slave_mem/U1553               NAND41           0.188
ahb_slave_inst/ahb_slave_mem/U1690               NAND21           0.188
ahb_slave_inst/ahb_slave_mem/U1692               AOI220           0.188
ahb_slave_inst/ahb_slave_mem/FE_OCPC451_n5103    CLKBU2           0.190
ahb_slave_inst/ahb_slave_mem/FE_OFC343_n4382     CLKBU2           0.190
ahb_slave_inst/ahb_slave_mem/FE_OFC330_n4382     CLKBU6           0.190
ahb_slave_inst/ahb_slave_mem/U951                OAI210           0.190
ahb_slave_inst/ahb_slave_mem/U398                AOI220           0.190
ahb_slave_inst/ahb_slave_mem/U1566               NAND40           0.190
ahb_slave_inst/ahb_slave_mem/U1694               NAND20           0.190
# -----------------------------------------------------------------------------
```

Bibliography

[1] Advanced Microcontroller Bus Architecture (AMBA) wikipedia.

[2] Design For Testability (DFT) wikipedia.

[3] Setup/hold time wikipedia.

[4] Integrated Circuit (IC) wikipedia.

[5] Register Transfer Level (RTL) wikipedia.

[6] System on Chip (SoC) wikipedia.

[7] ARM website.

[8] Digital Asic Group Lund University (2005) Digital ASIC Design. A Tutorial on the Design Flow.

[9] AMS C35 description.

[10] AUTOMATIC TEST PATTERN GENERATION WITH POWER AWARE EFFORTS AND ITS VALIDATION OF A GIVEN DESIGN (ASIC) USED IN NETWORK APPLICATION.

[11] Low power techniques synopsys.

[12] Erik Brunvand. *Digital VLSI chip design with Cadence and Synopsys CAD tools*. Addison-Wesley, 2010.

[13] Michael Bushnell and Vishwani D Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer Science & Business Media, 2000.

[14] Design Compiler, RTL User, and Modeling Guide. Synopsys. *Inc., see http://www. synopsys. com*, 2001.

[15] Hima Bindu Kommuru and Hamid Mahmoodi. Asic design flow tutorial using synopsys tools. *Nano-Electronics & Computing Research Lab, School of Engineering, San Francisco State University San Francisco, CA, Spring*, 2009.

[16] Fred Kröger and Stephan Merz. *Temporal logic and state systems*. Springer Science & Business Media, 2008.

[17] Deepak Kumar Tala. Verilog tutorial. *EB/OL]. Available http://www. asicworld, com*, 2003.

[18] Alain Vachoux. Top-down digital design flow. *Microelectronics System Lab*, 2006.

[19] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen. *VLSI test principles and architectures: design for testability.* Academic Press, 2006.

[20] Wayne Wolf. *Modern VLSI design: IP-based design.* Pearson Education, 2008.