2013-01-01

Anasic Design And Test Methodology For An Undergraduate Design And Fabrication Project

Arun Joseph Kurian
University of Texas at El Paso, akurian@miners.utep.edu

ANASIC DESIGN AND TEST METHODOLOGY FOR AN UNDERGRADUATE DESIGN AND FABRICATION PROJECT

Arun Joseph Kurian, MSCE

Department of Electrical and Computer Engineering

APPROVED:

Eric MacDonald, Ph.D.

John Moya, Ph.D.

Ryan B. Wicker, Ph.D.

Benjamin C. Flores.
Dean of the Graduate School

Dedication

To My Parents and Friends

AN ASIC DESIGN AND TEST METHODOLOGY FOR AN UNDERGRADUATE DESIGN AND FABRICATION PROJECT

by

Arun Joseph Kurian, MSCE

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

DECEMBER 2012

ACKNOWLEDGEMENTS

I would like to thank my graduate thesis advisor Dr. Eric MacDonald for giving me the opportunity to work in the field I am most passionate about: Digital Circuit Design. I could have never asked for a better major advisor. Dr. MacDonald has not only expanded my skills in VLSI area but has elevated my confidence to do well in my professional endeavors. Thank you Dr. MacDonald, I will always be grateful for everything you have done for me.

Also, I would like to extend my most humble gratitude to my team inIntelFolsom, California who gave me an opportunity for internship.This was a great motivating experience to complete my studies and get back to working again.

Additionally, I wish to give my sincere thanks to members of UTEP's ASICS Lab for their support and interest in this research. In particular, I would to thank Praveen Palakurthi not only for his welcomed suggestions and endless assistance, but for making me feel I was not alone in the intricate but rewarding field of Digital Circuit Design.

I am also compelled to recognize that without the assistance of the UNIX systemadministratorsand the staff from the ECE Dept. I would have never completed my work.

ToDr. Moya, I am obliged for his academic instruction, for his professional advice, and for motivating me to never quit. ToDr. Wicker I am truly grateful for taking a sincere interest in my thesis topic and for highlighting areas of improvement.

Finally, I would like to thank my family and friends who have always believed that I can accomplish my goals. My parents, and my brotherTharunare everything to me and I would never be here,without their support.

ABSTRACT

During the 1990's the main focus of chip design methodologies was on the timings and area constraints. Power consumption was considered significant only after a drastic increase of device densities from 130nm on as well as dramatic increases in sub threshold leakage. As technology advanced from 130nm to 90nm and below there was a significant increase in leakage current due to lower threshold voltage and the influence of the deep submicron effects. High power consumption causes different problems such as increasing the cost of the product, reducing the reliability, reducing the battery life among others. Therefore EDA tools were designed to maximize the speed while minimizing area and only recently focused on improving power.

The main objective of this thesis is to complete a study of an ASIC (Application Specific Integrated Circuit) design and test flow to establish a full design methodology for an undergraduate class chip design and fabrication project from Verilog RTL to GDS2 for fabrication. The tools includeSynopsys Design Compiler to generate a netlist of the physical design and Synopsys IC Compiler to perform the placement and optimization followed by clock tree synthesis, routing and lastly corechecking. The core is then inserted and connected with the chip pad frame using Synopsys Custom Designer.The final chip GDS generated will be sent to Mosis for fabrication. The Verification of the final chip design will be done usingCadenceVirtuoso. This project gives an overview of different steps in the developmentof an ASIC, front end and back end design using Synopsys Design Compiler and IC compiler flow. In this thesis a simple 8 bit counter is considered as an example.

This Thesis will provide the students with familiarity with the current industry standard tools from vendors like Synopsys and Cadence and thestudents will be well versed with acomprehensive ASIC design flow. The final design will be sent to Mosis for fabrication and the student teams will

have working silicon in their hands with five packaged chip per project the demonstration of which will be beneficial when interviewing for a job in the chip industry.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 What is an ASIC?

Integrated circuits are usually fabricated on silicon wafers with the wafer containing hundreds of dicestructures. ASIC stands for Application Specific Integrated Circuit and an IC designed for a specific application while using standard cell library is referred to an ASIC. Examples of ASICS include chips designed for satellites, chips designed for automotive systems, and chips designed as interfaces between memory and CPU on a personal computer. Examples of IC's which are not ASIC include memories, microprocessors etc.

1.2 Types of ASICS

ASICS are categorized based on the technology used for manufacture. The various types are full-custom ASICS and semi-custom and semi-custom can be further classified as standard cell based ICs (CBICs), Gate Array (GA) type.

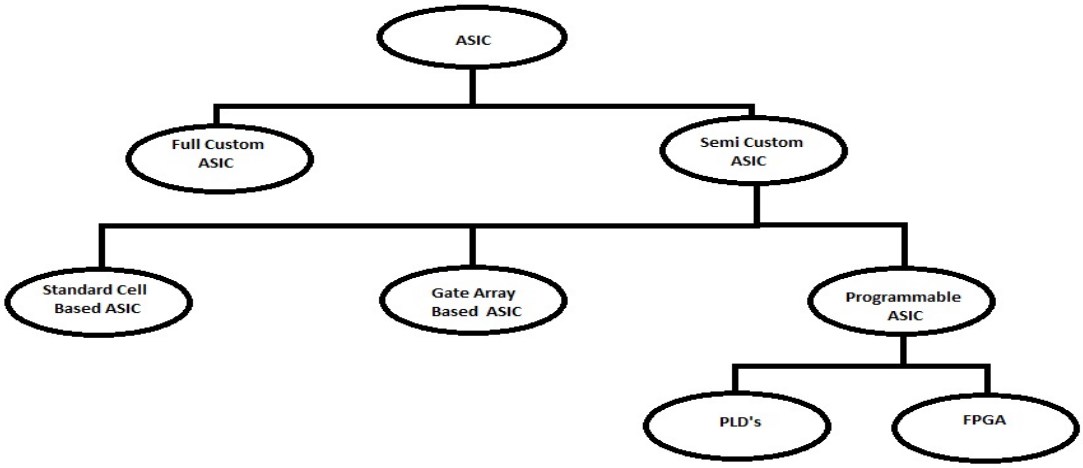

Figure 1: Types Of ASICS

1.2.1 FULL CUSTOM ASICS

The mask layers are customized in a full-custom ASIC where a majority of the design is done at the transistor level in terms of layout and simulation. Full custom ASICS are designed if there are no standard libraries available or when particular care is required to meet performance or power requirements of an application. The full custom methodology offers the highest performance with the disadvantages of increased design time, complexity, design expense, and highest risk. Microprocessors are generally full-custom, but designers are increasingly turning to semicustom ASIC techniques in this area as well. Other examples of full-custom ASICS are high-voltage automobile, analog/digital (communications), or sensors and actuators.

The disadvantages of full custom ASIC are:

1. Design complexity
2. Lack of automation in the tools
3. Substantial design time

1.2.2 SEMI-CUSTOM ASICS

Semi-custom ASICS are ASICS that are customized in one or two areas. A semi-custom ASIC is manufactured with the masks for the diffused layers already fully defined, so the transistors and other active components of the circuit are already fixed. The customization of the final ASIC product to the intended application is done by varying the masks of the interconnection layers, e.g., the metallization layers.

1.2.3 STANDARD CELL ASICS

A standard cell-based ASIC uses standard cells which are predesigned logic cells (AND gates, OR gates, multiplexers, and flip-flops, for example). The standard-cell areas (also called flexible blocks) in a Standard Cell Based ASIC are built of rows of standard cells. The standard-cell areas may be used in combination with larger predesigned cells like memories or analog circuits.

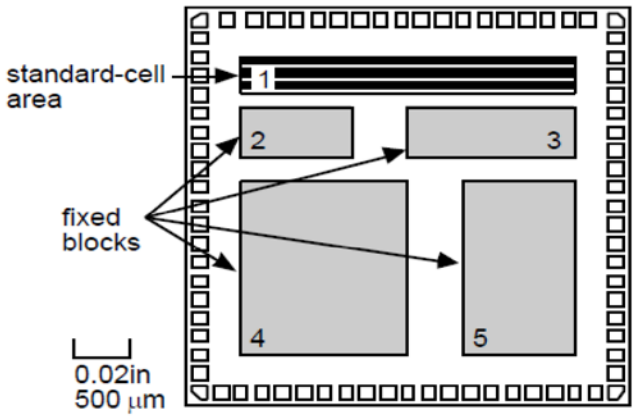

Figure 2:Standard Cell Based ASIC

1.2.4 GATE ARRAY ASICS

Gate array ASICS are partially finished with rows of transistors and resistors required but the transistors are unconnected. The chip is completed by connecting the required transistors and

resistors with the back-end metal layers. The Gate array is made of "basic cells", where individual cells contain some number of transistors and resistors depending on the vendor. Using a cell library (gates, registers, etc…) and a macro library (more complex functions),the customer designs the chip and the vendor's software generates the interconnection masks.

These final masking stages are less costly than those associated with designing a full-custom ASIC from scratch.

A gate array circuit is a prefabricated circuit with no particular function in which transistors, standard logic gates, and other active devices are placed at regular predefined positions and manufactured on a wafer, usually called **Master Slice**. Creation of a circuit with a specified function is achieved by connecting the required elements using metal layers at the time of manufacture and this can be done on wafers that have already been fabricated with complete transistors (front end) and thus eliminate the amount of time between completing the design and obtaining silicon.The gate array drawbacks are low density and low performance as compared to full custom or standard cell ASICS.

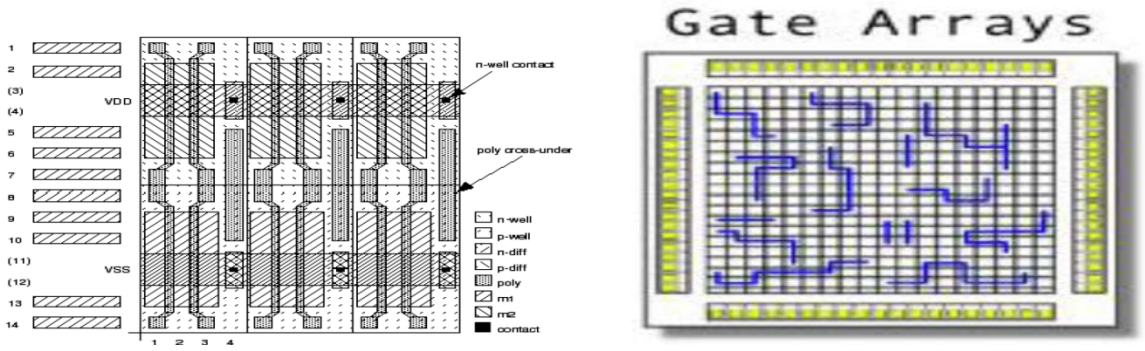

Figure 3: Gate Array ASIC

1.2.5 PROGRAMMABLE ASICS

a) PLDs are programmable logic devices that can be used to perform complex functions. PLDs can be programmed for specific applications. PLDs use different technologies to allow programming of the device.

PLDs are having these common features:

- No customized mask layers or logic cells
- Fast design turnaround
- A single large block of programmable interconnect
- A matrix of logic macro cells that usually consist of programmable array logic followed by a flip-flop or latch.

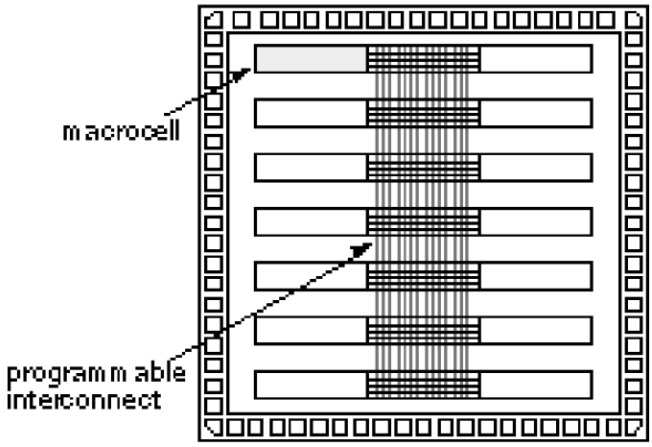

Figure 4:PLD

b) FPGA (Field Programmable Gate Array) are first developed in the mid of 80's by Xilinx. FPGAs are better in terms of functionality in comparison with PLDs and can be used for more complex and denser designs.

FPGAs are composed of logic blocks instead of unwired transistors.

The core of the FPGA is an array for logic cells that can perform combinational and sequential logic

FPGAs have the following features:

- Less dense than custom mask
- Higher unit cost
- Ready to be used out the box
- Reprogrammable
- In this project, in which a methodology is established to be used in an undergraduate design project, which ends with a free 0.5 micron CMOS fabrication, an FPGA-based emulation process is used to evaluate and validate a design written in Verilog – similar to the approach taken by industry. This significantly reduces the risk of the chip design being fabricated that is wrong.

Application Specific Integrated Circuits (ASICS) and Field Programmable Gate Arrays (FPGAs) are different types of custom chips, which differ in their properties, cost, and manufacturing process. The choice of which to use depends on the required application and its requirements and in some applications, FPGAs are used to prototype a design given the low NRE expense, but are replaced in production by ASICS to improve the per device costs [2].

Older style ASICS were of gate array type which consisted of unconnected transistors. The most common type of ASIC currently used is the standard cell based ASIC which accounts for most digital logic fabricated today. Many electronics companies with a core competency outside of the chip industry (e.g. routers, cell phones, graphics processing) can design customized chips with the

standard cell library and fabricate the chips at a silicon foundry. Examples of such fab-less companies include NVidia or Qualcomm and this business model is gaining popularity given the huge costs associated with silicon manufacturing and the availability of reliable and inexpensive foundry services. The most popular foundry services are now located in Taiwan and China with companies like the Taiwanese Semiconductor Manufacturing Company or Universal Manufacturing Company. Companies like IBM and Texas Instruments in the USA also provide these services generally providing better performance but at an increased cost per die [2].

1.3 Chapter Organization

The focus of this thesis is to develop a methodology to generate chips at an education level for undergraduatesusing standard industry EDA tools by vendors like Synopsys and Cadence. The remainder of this thesis is organized as follows:In chapter 2 the main focus is on how this new ASIC design methodology has made the whole process of chip production fast, easy and efficient. The chapter also discusses digital logic basics defining terms like propagation delays with a brief description of the power consumption in digital circuits. Chapter 3focuses mainly on the tool Design Compiler (DC) by Synopsys used for the synthesis of RTL. Chapter 4 mainly talks about the synthesis optimization techniques that could be used for improving area and speed of the final chip.Chapter5 describes focuses on the tool IC Compiler by Synopsys which is used mainly for the place and route operation to generate the final core of the chip in design. Inchapter 6 the methodology used is discussed.Chapter 7 provides the conclusion and future work. Appendix A lists the Verilog code for the simple 8 bit counter used to test the methodology. Appendix B shows the synthesis TCL script that was used. Appendix C lists the IC Compiler TCL script that was used for design place and route operation.

CHAPTER 2: BACKGROUND AND MOTIVATION

The main motivation behind this project was to provide the undergraduate students with a familiarity with the chip design process to help the students complete the entire chip design process starting from writing the RTL to fabricating the silicon. The finals designs will be sent to MOSIS for fabrication and with the final silicon in hand will be a great asset for discussion in job interviews and securing a job or finding future research opportunities depending on their interests. Power of the designed and fabricated chips will be evaluated in a subsequent semester upon arrival of the silicon hardware.

2.1 Digital Circuit Design

Digital circuit design prior to the last two decades involved developing schematics for the required functions, which required selecting the right gates and making the required connections. This method is slow, tedious and prone to error. An alternative that was developed was to use a Hardware Descriptive language (HDL) to describe the behavior or a design - the two most commonly used HDLs are VHDL and Verilog. The designs can be created much more rapidly, and the only errors are likely to be with the logic design, as opposed to the implementation – such as incorrectly connected gates.HDLs can be compiled using compilers for simulation and testing requirements. Once the design is believed to function correctly synthesis is performed to get a form required by the FPGA or ASIC. The resulting output is analyzed for real time performance and can be refined and recompiled for a better design in the future. This process can be iterative.

2.3 Digital Logic Basics

While most of the optimization is done by the synthesis tools, knowledge of the basic logic gates is still required to understand the synthesis process and optimization methods.

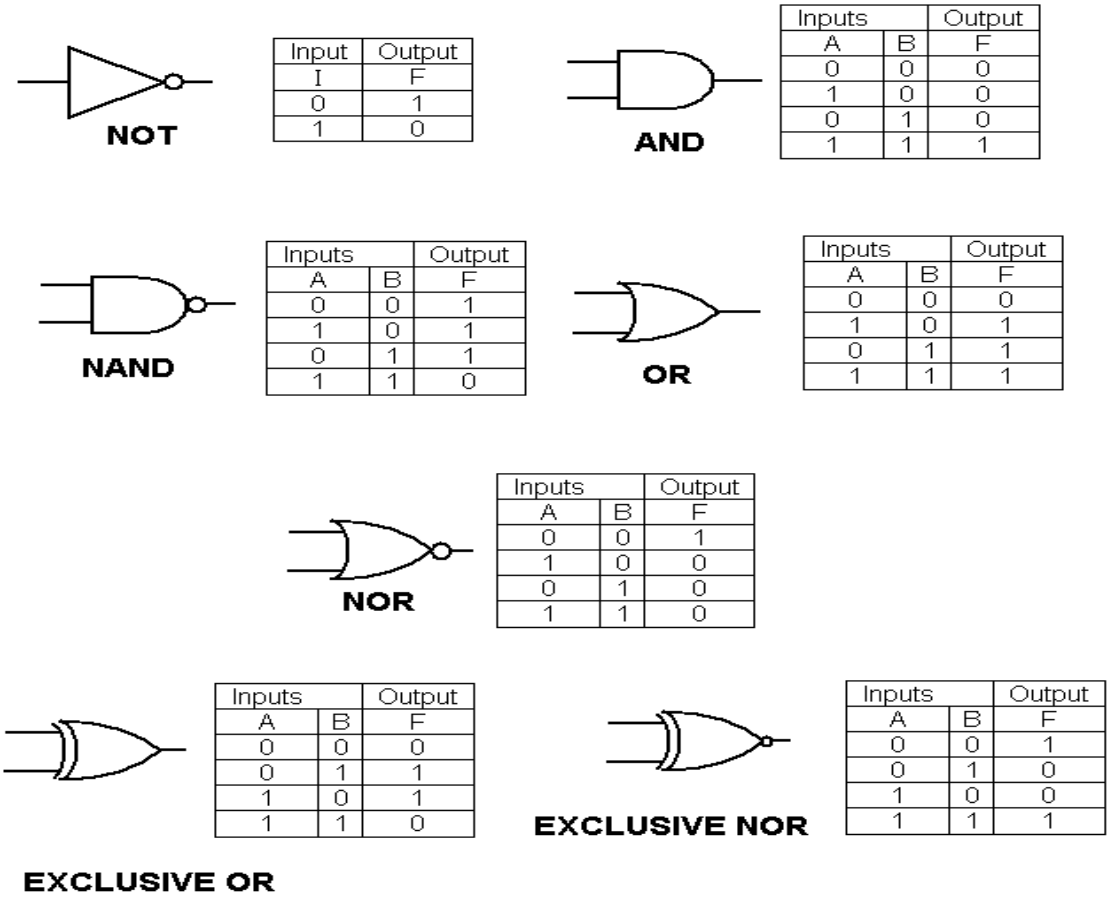

Figure 5: Gate truth Table

2.4 Implementation of gates

Individual gates are built using transistors and the characteristics of the gate depend on the type and configuration of transistors used. For example, the CMOSimplementation of a NOR gate could be represented by figure:

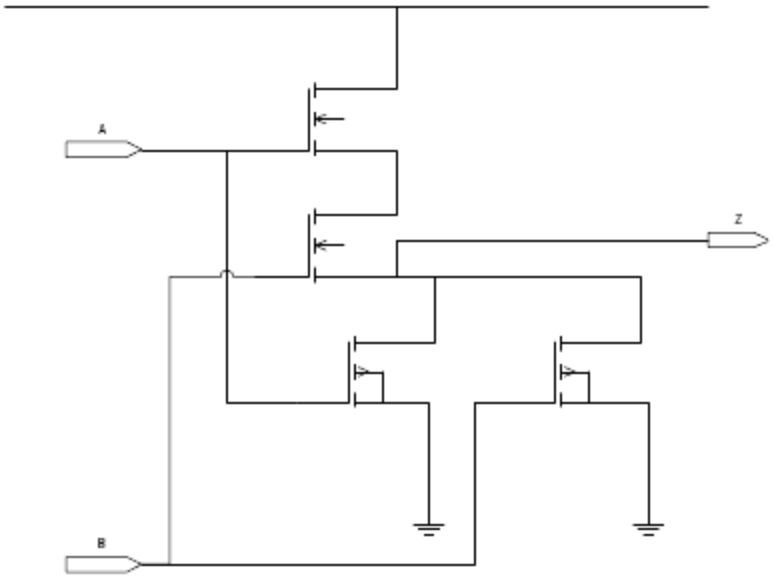

Figure 6: NOR Gate using transistors

Since these circuits are built from transistors, logic gates inherit a number of characteristics that areimportant to digital design:

- Transistors have capacitance which affects the speed and power dissipation of the device [2].
- Transistors consume current through their input transitions and provide a limited amount of power at the output pins [2]. Therefore the drive strength is limited. This number referred to asthe fan-out of the gate and is usually less than 20.

2.5 Propagation delays

In response to changes to the inputs, outputs of the logic gates change after a certain time known as the propagation delay. This is caused due to the time taken to charge and discharge the internal and load capacitances and different transistor configurations also give different delays.

In a large circuit there are a large number of gates and the propagation delay will be the sum of the propagation delays of the gates thorough which the signal passes. The path that has the

highest delay is known as the critical path for the circuit. Optimizing this path is the central objective of Static Timing Analysis (STA) which is the critical feedback mechanism for logic synthesis as well as a final check required before fabrication [2].

2.6 Power Consumption in Digital Circuits

Power consumption is an important issue while designing chips. Increased power consumption may result in thermal challenges. From a commercial stand point, the lower power consuming products are more competitive particularly in terms battery life in portable applications. Two main categories of power consumption are:

- **Static power:** This is the power used by a logic gate when thelogic is held constant. Leakage current is the component contributing to subthreshold as well as gate oxide tunneling current. This component is becoming a larger and larger fraction of the total power as the threshold voltage of the transistors is decreasing (leakage increases 10x for 70mV drop in the threshold voltage) to match the decreases in the supply voltage. The gate oxides are dropping below 25 angstroms in thickness – thus allowing for additional Fowler Nordheim Tunneling – resulting in additional leakage current [2].
- **Dynamic power:** This is the power consumed by the logic gate when anode changes states. This Dynamic power is caused due to the switching current generating the change and due to the charging and discharging of the gates capacitance to generate the required modification in voltage level [2].

Dynamic power is more significant and can be influenced by the logic design. Minimizing the number of transistors will minimize the power consumptionas well as reducing the functionality of the chip. There are other ways in which this can be attempted:

- **Disabling unused parts of the circuit:** An AND gate can be provided at the inputs with one input connected to an enable signal. The entire circuit can be kept static with the enable signal kept low by reducing input transitions [2].
- **Reducing glitches in a circuit:** A glitch is a temporary change in the signal level before the final value is reached. These can be avoided by using the right logical arrangement and timing can reduce power consumption. Glitches are normal and not a functional concern as long as static timing requirements are met [2].

2.7 Low Power Synthesis

The advent of portable devices has made low power circuit design an important research area. High power consumption increases heat dissipation and reduces the reliability of complex modern circuits with high transistor counts and fast clock rates. In CMOS the large portion of power dissipation is due to dynamic power consumption [18]. Till now power reduction has been mainly dealt with by reducing the gate switching frequency. But in reality gates have delay resulting in glitch transition that contributes to about 30% of the power consumption [5].

Typical approach in low power synthesis is to first generate a multilevel AND-OR or NAND-NOR representation of logic function. Next this representation is optimized for power. AND-XOR logic is more compact than their boolean counterparts in terms of layout area. However conventional optimization of XOR yield poor results [6].

Primary method used to reduce power is by voltage reduction. However voltage reduction below 1 volt begins to create problems. Under these situations the design and automation tools play a crucial role. In the past decades wide variety of solutions have been come up with to solve these this problem which includes RTL power management and multiple voltage assignment, to power-aware logic synthesis and physical design, to memory and bus interface design [7].

2.8 Placement Algorithms

The general nature of CMOS technology encourages the use of "gate-matrix" or "strip based" layout design style. In this style the cells are formed by two horizontal strips of diffusion with vertical poly silicon strips running between them. In a well fit design the source of one FET will become the drain of the next with the diffusion unbroken. If the circuit used different nets for the FETs then it is necessary to isolate them with a gap. To minimize the area the number of gaps need to be reduced. This is a difficult problem because search-space of possible FET ordering is large [8].

The automated tools that do the placement operation play a crucial role in transforming the circuit description into a physical description. Placement operation is divided into an initial placement phase followed by an iterative placement improvement phase. The objective of the initial placement phase is to minimize the area and wire length. An iterative placement phase is required to optimize placement as all the information is not available during the initial placement phase [9].

Placement algorithms are classified into two categories: Constructive and iterative types. Constructive algorithm takes in modules and a netlist defining the interconnection between the modules. The input to the iterative algorithm is the initial placement phase [10].

2.9 Static Timing Analysis

Due to aggressive technology scaling and limits on the optical lithography the gap between the design phase and fabrication phase is increasing leading to performance offsets between the designed and fabricated specifications. Static Timing Analysis (STA) helps in determining the circuit performance so that the circuit can be designed for extreme conditions [11].

Traditional static timing analysis is becoming insufficient to accurately evaluate the process variation impact on designs performance considering increase in the variable parameters like power supply voltage temperature corners. The use of a novel approach called statistical static timing

analysis helps to overcome these issues with traditional STA. However this novel approach needs costly additional data such asan accurate process variation information, and a statistical standard cell library characterization. STA tools can be used to estimate path delays. This helps in finding the worst case path in a circuit which is a critical information governing the speed of a circuit. There are various techniques to determine this information. During this calculation false paths in a design must be eliminated and only delays of true paths must be calculated using the various proposed methods [12].

2.10 Power Estimation and Reduction

Power estimation and reduction is becoming a critical area for ASIC designers. Reducing both the leakage and dynamic power is important to make the product more commercially viable. Major power reductions are only possible at RTL and system levels. At this level sequential modifications can be made via techniques like sequential clock gating, power gating, voltage/frequency scaling and other micro-architectural techniques [15]. Without accurate power estimation power reduction cannot be done at optimal levels. Various RTL power estimation tools have been proposed. The increase in circuit sizes and complex effects introduced due to deep sub-micron designs make the task of power estimation very challenging [14].

CHAPTER 3:LOGIC SYNTHESIS

The Design Compiler [DC] is the synthesis tool from Synopsys. DC takes a RTL [Register Transfer Logic] hardware description [design written in either Verilog/VHDL], and standard cell library as input and outputs a technology dependent gate-levelnetlist. The gate-levelnetlist is the structural representation of only standard cells based on the cells in the standard cell library. The synthesis tool internally performs many steps, which are listed below. Also below is the flowchart of synthesis process.

1) DC reads in technology libraries, DesignWare libraries, and symbol libraries. During the synthesis process, DC translates the RTL description to components extracted from the technology library and DesignWare library. The technology library consists of basic logic gates and flip-flops. The DesignWare library contains more complex cells for example adders and comparators. DC can automatically determine when to use Design Ware components and then efficiently synthesize these components into gate-level implementations.

2) Reads the RTL hardware description written in either Verilog/VHDL.

3) The synthesis tool thenperforms many steps including high-level RTL optimization, RTL to un-optimized boolean logic, technology independent optimizations, and finally technology mapping to the available standard cells in the technology library, known as the target library. This resulting gate-levelnetlist also depends on constrains given by the designer. Constraints are the designer's specification of timing and environmental restrictions [area, power, process etc.] under which synthesis is to be performed.

4) After the optimization process, the design is ready for DFT [design for test/ test synthesis]. DFT is test logic that designers can integrate into design during synthesis. This helps the designer to

test for issues early in the design cycle and also can be used for testing the chip once back from fabrication.

5) After test synthesis, the design is ready for the place and route. The place and route tools place and physically interconnect cells in the design. Based on the physical routing, the designer can back-annotate the design and can use Design Compiler again to resynthesize the design for more accurate timing analysis.

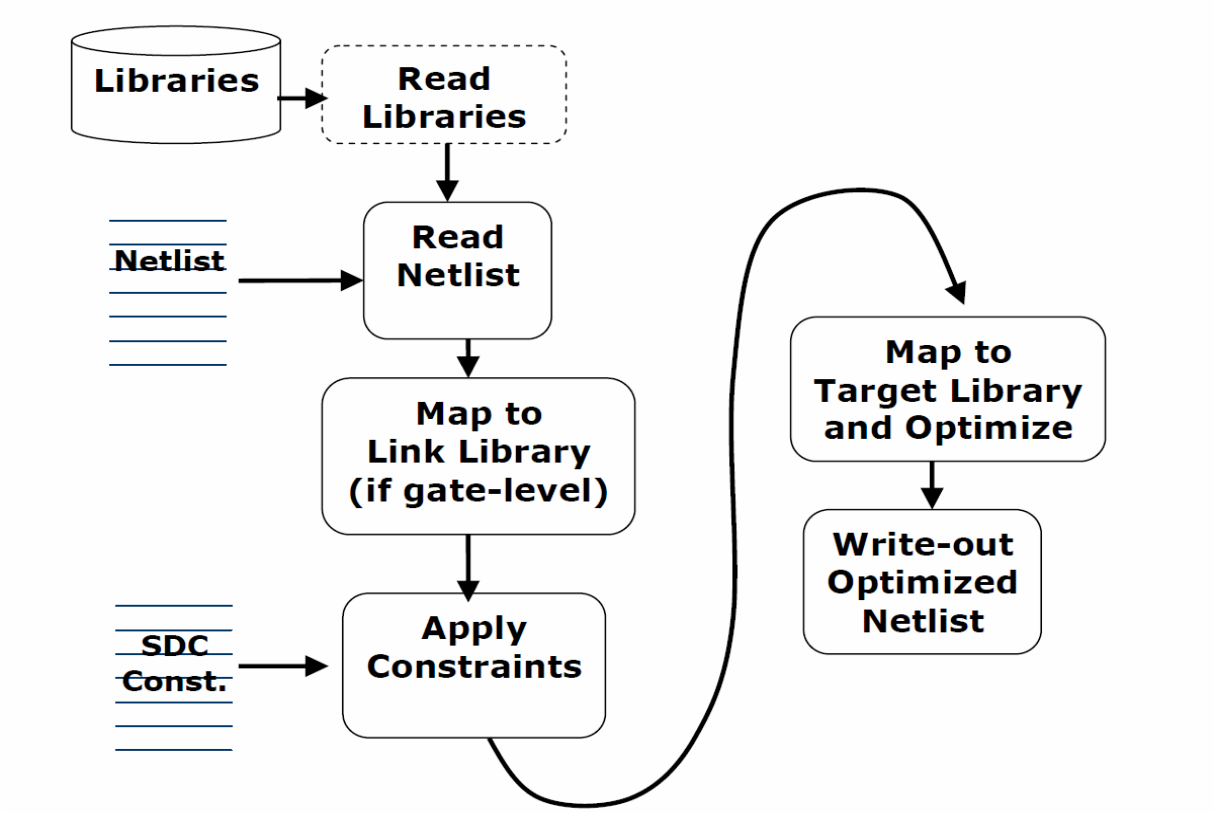

Figure 7: Synthesis Flow

3.1 Important DC Parameters

There are four important parameters that should be setup before one can start using the tool are:

- **search_path:** This parameter is used to specify to the synthesis tool the paths that should be searched when looking for a synthesis technology libraryduring synthesis.
- **target_library:** The parameter specifies the file that contains the logic cells that should be used for mapping during synthesis.
- **symbol_library:** This parameter points to the library that contains the "visual" information on the logic cells in the synthesis technology library. Logic cells have a symbolic representation and information about these symbols is stored in this library.
- **link_library:** This parameter points to the library that contains information about the logic gates in the synthesis technology library. The tool uses this library solely for reference but does not use the cells present for mapping as in the case of target_library.

3.2 Design Objects

There are eight different types of objects categorized by Design Compiler:

- **Design**: The design corresponds to the circuit description that performs some logical function. The design may be stand-alone or may include other sub-designs. The sub design can be treated as a different design.
- **Cell**: The cell is the instantiated name of the sub-design in the design. In Synopsys terminology, there is no differentiation between the cell and an instance. They are treated as a cell.
- **Reference**: This is the definition of the original design to which the cell or instance refers.
- **Ports**: These are the primary inputs, outputs of the design.
- **Pin**: The pin corresponds to the inputs, outputs of the cells in the design.

- **Net**: These are the signal names, i.e., the wires that connect the design together by connecting ports to pins and/or pins together.
- **Clock**: The port or pin that is identified as a clock source.
- **Library**: Corresponds to the collection of technology specific cells that the design is targeting for synthesis, or linking for reference.

3.3 Design Entry

Before synthesis, the design must be entered into the Design Compiler (referred to as DC from now on) in the RTL format. DC provides the following two methods of design entry:

- **Read**command
- **Analyze& elaborate** commands

The **analyze & elaborate** commands are two different commands, allowing designers to initially analyze the design for syntax errors and RTL translation before building the generic logic for the design. The generic logic or GTECH components are part of Synopsys generic technology independent library. The generic logicis an unmapped representation of boolean functions and serves as placeholders for the technology dependent library.

The **analyze** command also stores the result of the translation in the specified design library that maybe used later. So a design analyzed once need not be analyzed again and can be merely elaborated, thus saving time. Conversely **read** command performs the function of **analyze** and **elaborate** commands but does not store the analyzed results, therefore making the process slow by comparison.

The commands used for the methods in DC are as given below:

Read command:

dc_shell>read –format <format><list of file names>

Analyze and Elaborate commands:

dc_shell>analyze -format <format><list of file names>

.syn file is the file in which the analyzed information of the design analyzed is stored.

EX: The adder entity in the adder.vhd has a generic parameter "width" which can be specified during elaboration.

3.4 Technology Library

Technology libraries contain the information that the synthesis tool needs to generate a netlist for a design based on the desired logical behavior and constraints on the design. The tool referring to the information provided in a particular library would make appropriate choices to build a design. The libraries contain information regarding the area of the cell, the input-to-output timing of the cell, constraints on fanout of the cell, and the timing checks that are required for the cell. Other information stored in the technology library may be the graphical symbol of the cell for use in creating the netlist schematic.

The **target_library**, **link_library**, and **symbol_library**parameters in the **startup** file are used to set the technology library for the synthesis tool.

3.5 Design Attributes and Constraints

A designer, in order to achieve optimum results, has to methodically constrain the design, by describing the design environment, target objectives and design rules. The constraints contain timing and/or area information, usually derived from the design specifications. The synthesis tool uses these constraints to perform synthesis and tries to optimize the design with the aim of meeting these constraints.

3.5.1 DESIGN ATTRIBUTES

Design attributes set the environment in which a design is synthesized. The attributes specify the process parameters, I/O port attributes, and statistical wire-load models. The most common design attributes and the commands for their setting are given below:

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 slow a particular driver is. Signals that are arriving later than the clock can have an attribute that specifies this fact. The load attribute specifies how much capacitive 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... The command for setting this attribute is given below:

set_load<value><object_list>

EX:**dc_shell>set_load 1.5 x_bus**

***Drive*:** The drive specifies the drive strength at the input port, usually specified as a resistance value. This value controls how much current a particular driver can source. The larger a driver isi.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. The command for setting the drive for a particular object is given below

set_drive<value><object_list>

EX:**dc_shell>set_drive 2.7 ybus**

3.5.2 DESIGN CONSTRAINTS

Design constraints specify the goals for the design. The twoimportant constraints are area and timing. Depending on how the design is constrained the DC/DA tries to meet the set

objectives. Realistic specification is important, because unrealistic constraints might result in excess area, increased power and/or degrading in timing. The basic commands to constrain the design are:

- **set_max_area**: This constraint specifies the maximum area a particular design should have. The value is specified in units used to describe the gate-level macro cells in the technology library.

  EX:**dc_shell>set_max_area 0**

  Specifying a 0 area willresult in the tool prioritizing size over power or performance.**create_clock**: This command is used to define a clock object with a particular period and waveform. The –**period** option defines the clock period, while the –**waveform** option controls the duty cycle and the starting edge of the clock. This command is applied to a pin or port, object types.

  Following example specifies that a port named CLK is of type "clock" that has a period of 40 ns, with 50% duty cycle. The positive edge of the clock starts at time 0 ns, with the falling edge occurring at 20 ns. By changing the falling edge value, the duty cycle of the clock may be altered.

  EX:**dc_shell>create_clock –period 40 –waveform {0 20} CLK**

- **set_don't_touch_network**: This is a very important command, usually used for clock networks and resets. This command is used to set a **dont_touch**property on a port, or on the net. Note setting this property will also prevent DC from buffering the net. In addition,a gate coming in contact with the "don't_touch" net will also inherit the attribute.

  EX:**dc_shell>set_dont_touch_network {CLK, RST}**

- **set_don't_touch**: This is used to set a dont_touch property on the **current_design,** cells, references, or nets. This command is frequently used during hierarchical compilation of blocks for preventing the DC from optimizing the don't_touch object.

EX:**dc_shell>set_don't_touchcurrent_design**

- **current_design**is the variable referencing the current working design. This variable can be set using the **current_design**command as follows

  dc_shell>current_design<design_name>

- **set_input_delay**: This command specifies the input arrival time of a signal in relation to the clock. This parameter specifies the time taken for the data to be stable after the clock edge.The timing specification of the design usually contains this information, as the setup/hold time requirements for the input signals. From the top-level timing specifications the sub-level timing specifications may also be extracted.

  EX:**dc_shell>set_input_delay –max 23.0 –clock CLK {datain}**

  dc_shell>set_input_delay –min 0.0 –clock CLK {datain}

  The CLK has a period of 30 ns with 50% duty cycle. For the above given specification of max and min input delays for the datain with respect to CLK, the setup-time requirement for the input signal datain is 7ns, while the hold-time requirement is 0ns.

- **set_output_delay**: This command is used at the output port, to define the time taken for the data to be available before the clock edge. This information is usually is provided in the timing specification.

  EX:**dc_shell>set_output_delay – max 19.0 –clock CLK {dataout}**

  The CLK has a period of 30 ns with 50% duty cycle. For the above given specification of max output delay for the dataout with respect to CLK, the data is valid for 11 ns after the clock edge.

- **set_max_delay**: This command defines the maximum delay required in terms of time units for a particular path. In general used for blocks that contains combination logic only.

However can also be used to constrain a block that is driven by multiple clocks, each with a different frequency. This command has precedence over DC derived timing requirements.

EX:**dc_shell>set_max_delay 5 –from all_inputs() – to_all_outputs()**

- **set_min_delay**: This defines the minimum delay required in terms of time units for a particular path. Opposite of the set_max_delay command this command has precedence over DC derived timing requirements.

  EX:**dc_shell>set_max_delay 3 –from all_inputs() – to_all_outputs()**

CHAPTER 4: SYNTHESIS OPTIMIZATION TECHNIQUES

An optimized design is a design that has satisfied the timing and area requirements.This can be done in two stages; one at the code level and the other during synthesis. The code level optimization involves modifications to the RTL and is required toavoidinconsistencies between simulation results before and after modifications.

4.1 Model Optimization

Model optimizations are important as the logic that is generated by the synthesis tool is sensitive to the RTL code that is provided as input. Different RTL codes generate different logic depending on the synthesis tool. The changes made to the RTL can result in an increase or decrease in the number of synthesized gates and change timing characteristics. A logic optimizer results in different endpoints for best area and best speed depending on the starting point provided by a netlist. The different starting points are obtained by modifying the same HDL code using different constructs. Some of the optimizations that are options for a better design are as follows:

4.1.1 RESOURCE ALLOCATION

Thefollowingmethod refers to the process of sharing a hardware resource. Consider the following *if* statement:

```
if A = '1' then
E = B + C;
else
E = B + D;
end if;
```

The code above would generate two ALUs one for the addition of B+C and other for the addition B + D.A single ALU can be shared for both the addition operations. The above code would synthesize as follows:

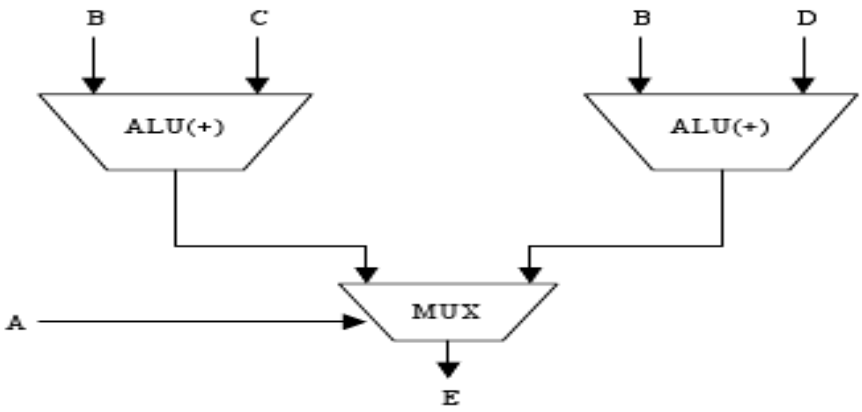

Figure 8: Bad resource allocation

The code above can be rewritten as follows :

if A = '1' then

temp= C; // A temporary variable introduced.

else

temp= D;

end if;

E = B + temp;

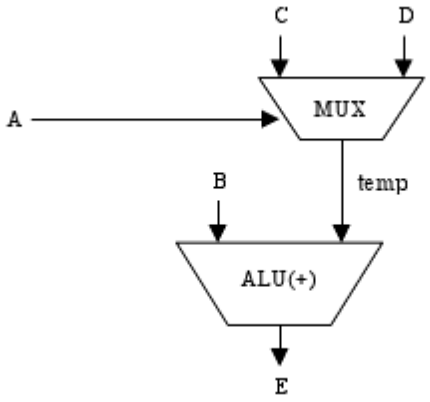

Figure 9: Good resource allocation

From the figure above we can see that one ALU has been removed and only one ALU being shared for both the addition operations. However a multiplexer is introduced at the input to the ALU that contributes to path delay. Earlier only the multiplexercontributed to the delay.Withresource allocation the multiplexer and ALU contribute to the delay. Therefore due to resource sharing the area of the design has decreased but the delay is increased. This is the trade-off that the designer has to decide on. No resource sharing is generally performed for timing-critical designs.

4.1.2 COMMON SUB-EXPRESSIONS AND COMMON FACTORING

Identifying common sub expressions and reusing the pre computed values wherever possible is a good practice. A simple example is given below.

B= R1 + R2;

C <= R3 – (R1 + R2);

If the sub expression R1 + R2 in the signal assignment for C is replaced with B as given below, only one adder will be generated for the computation instead of two.

C <= R3 – B;

Such changes if made by the designer can cause the tool to synthesize better logic and also enable the tool for better optimizations.

4.1.3 REMOVING REDUNDANT CODE

Cases exist where the value of an expression might not change within a loop. The synthesis tool handles the loop statement by repeating the loop a specified number of times. In such cases redundant code getsgeneratedcausing additional logic to be synthesized. This can be avoided by moving the expression outside the loop, thus optimizing the design. An example is given below:

C =A + B;

for *C* in range 0 to 5 loop

T = C – 6;

Assumption: *C i*s not assigned a new value within the loop, thus the above expression would remain constant on every loop iteration.

end loop;

The above code would generate six subtracters for the expression when only one is necessary. Thus by modifying the code as given below we could avoid the generation of unnecessary logic.

C = A + B;

temp: = C – 6; // A temporary variable is introduced

for c in range 0 to 5 loop

T = temp;

Assumption: C is not assigned a new value within the loop, thus the above expression would remain constant on every iteration of the loop.

end loop;

4.1.4 CONSTANT FOLDING AND DEAD CODE ELIMINATION

Designer might accidentally leave expressions that are constant in value.

EX:

C: = 4;

Y = 2 * C;

Assigning the value of Y as 8 within the code can avoid the above unnecessary code. This method is called constant folding. Dead code elimination refers to certain code sections that never get executed.

EX:

A: = 2;

B: = 4;

if (A > B) then

end if;

The *if* statement would never be executed and can be eliminated from the code. The logic optimizer performs these optimizations automatically but the optimization time can be reduced if the designer keeps account of the above factors while designing.

4.1.5 FLIP-FLOP AND LATCH OPTIMIZATIONS

The designer should try to remove the unnecessary flip-flop and latch elements in the design. Only the clock sensitive signals should be placed in positive edge statements to avoid unnecessary flip-flops. Latches can be eliminated by specifying an else statement for if statements.

4.1.6 USING PARENTHESES

The correct parenthesis usage is critical to generate a well-timed design.

EX:

Result <= R1 + R2 - P + M;

The hardware generated for the above code is as given below in Figure

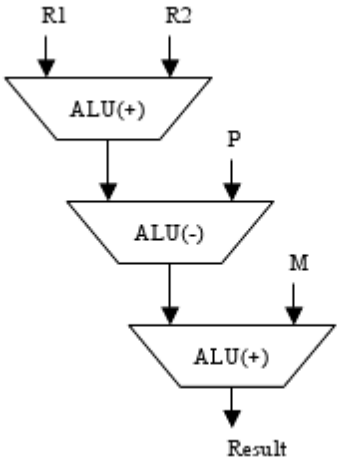

Figure 10: Synthesis without parenthesis

If the expression has been written using parentheses as given below, the hardware synthesized would be as given in Figure.

Result <= (R1 + R2) – (P - M);

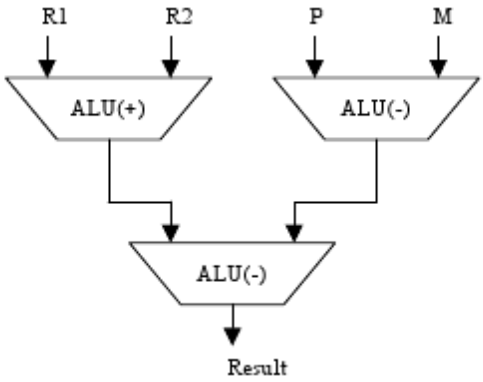

Figure 11: Synthesis with parenthesis

Therefore using parentheses the timing path for the datapath has been reduced as it does not need to go through one more ALU as in the earlier case.

4.2 Optimizations using Design Compiler

For obtaining an optimized design a lot of experimentation and synthesis iterations are required.For optimization, changing of HDL code may affect other blocks in the design. For this reason, HDL code modification is generally avoided.

Resolving multiple instances:

Before proceeding for optimization, one needs to resolve multiple instances of sub-blocks in the design. This is a necessary step as DC does not permit compilation until multiple instances are resolved.

EX: Let's say module A has been synthesized. Now moduleB that has two instantiations of module A - namely U1 and U2.The compilation will error out with a message stating that module A is instantiated 2 times in moduleB. This can be resolved as follows:

Use the**dont_touch**attribute on moduleA before synthesizing moduleB, or **uniquify**moduleB. **Uniquify**command creates unique definitions of multiple instances. So for the above case synthesizer will generate moduleA_u1 and moduleA_u2 , corresponding to instance U1 and U2 respectively.

Various DC optimization commands are given below:

4.2.1 COMPILE THE DESIGN

The compilation process translates the HDL code to gates from the target library. This is done through the **compile** command. The syntax is given below:

compile –map_effort<low | medium | high>

-incremental_mapping

-in_place

-no_design_rule | -only_design_rule

-scan

The default compile option is –map_effort medium. This produces good results for most of the designs. The map_effort high should only be used, if target objectives are not met through default compile. The -incremental_mapping is used to improve timing of the logic.

4.2.2 FLATTENING AND STRUCTURING

Flattening means reducing the design to an equivalent two level AND/OR form. This helps in optimizing the design by removing the intermediate variables. This option is set to "false" by default. There are two stages to this optimization. The first stage consists of flattening and structuring and the second stage consists of mapping of the resulting design to actual gates using optimization techniques.

Flattening reduces the number of logic levels between the input and output. This results in faster logic but is generally recommended for unstructured designs. He flattened design can then be structured during optimization. Flattening has a significant impact on the areareduction. In general one should compile the design using default settings (flatten and structure are set as false). If there are timing violations flattening and structuring should be employed. If the design is still failing apply only flattening without structuring.

The command for flattening is given below:

set_flatten<true | false>

-design <list of designs>

-effort <low | medium | high>

-phase <true | false>

The –phase option if set to true enables the DC to compare inverted and non- inverted form of the logic equations. The default setting for structuring is "true".

For example:

Before structuring after structuring

P = ax + ay + c

P = aI + c

Q = x + y + z Q = I + z

I = x + y

The command for structuring is given below.

set_structure<true | false>

-design <list of designs>

-boolean<low | medium | high>

-timing <true | false>

4.2.3 REMOVING HIERARCHY

Hierarchy is a logic boundary that prevents DC from optimizing across this boundary. Unnecessary hierarchy leads to cumbersome designs that are very difficult to optimize across boundaries and only optimizing within the boundary. To allow DC to optimize across these boundaries we can use the following commands.

dc_shell>**current_design<**design name>

dc_shell>**ungroup –flatten –all**

This allows the DC to optimize the logic separated by boundaries as one logic resulting in

better timing and a more optimized solution.

4.2.4 OPTIMIZING FOR AREA

DC by default does optimizations for timing. Designs that are not critical on timing can be optimized for area. Usage of the dont_touch attribute on the high drive strength gates will result in better area due to gate elimination. Once the design is mapped to gates the design should be recompiled with the new constraints. Incremental compile ensures that the design doesn't change much from initial stage bloating unnecessary logic.

4.3 Timing Issues

There are two kinds of timing issues that are important in a design- setup and hold timing violations:

- **Setup Time:** The time for which the input data should not change before the clock edge is referred to as set time. Input change during this period is going to result in a setup time violation. Figure illustrates an example with setup time equal to 2 ns. This means that signal DATA must be valid 2 ns before the clock edge; i.e. the data should not change during this 2ns period before the clock edge.
- **Hold Time:** The time for which the data should be held constant after the clock edge is referred to as hold time. Change of data during this period would trigger a hold timing violation. Figure illustrates an example with-hold time equal to 1 ns. This means that signal DATA must be held valid 1 ns after the clock edge; i.e. the data should not change during the 1 ns period after the clock edge.

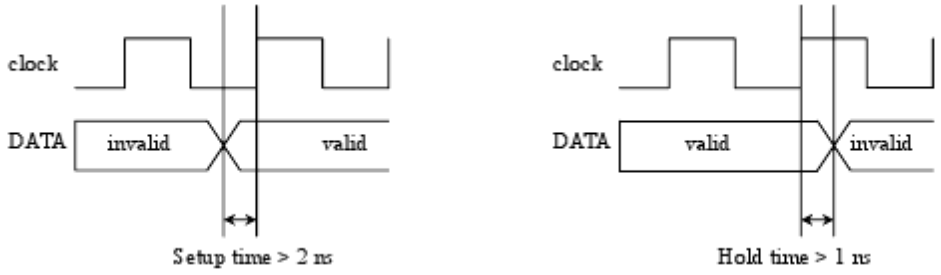

Figure 12: Setup and Hold times

The synthesis tool automatically checks for setup and hold time violations on the paths that have timing constraints imposed. The following are the equations used to check for these violations:

Tprop + Tdelay<Tclock - Tsetup (1)

Tdelay + Tprop>Thold (2)

Tprop is the propagation delay from input clock to output of the device,Tdelay is the propagation delay across the combinational logic through which the input arrives,Tsetup is the setup time requirement of the device, Tclock is clock period and Thold is the hold time requirement of the device.

So if the propagation delay across the combinational logic, Tdelay is such that the equation (1) fails i.e. Tprop + Tdelay is more than Tclock – Tsetup then a setup timing violation is reported. Similarly if Tdelay + Tpropis greater than Thold then a hold timing violation is reported. The input data arriving late due to large combinational circuit delay Tdelay can result in a setup time violation as the flip-flop doesn't get enough time to read the data. In case of hold time violation the data arrives faster than usual (insufficient Tdelay+Tprop) and the flip flop doesn't get enough time to store the data.

Various synthesis commands can be used for optimizing the design to avoid timing violations:

a) Compilation with a map_effort high option

b) Group critical paths together and assigning a weight factor

c) Register balancing

d) Choose a specific implementation for a module

e) Balancing heavy loading

4.3.1 COMPILATION WITH MAP_EFFORT HIGH

The default compilation option for map_effort is medium. This usually gives the best results with flattening and structuring options. The map_effort high option should be used in case of violations with the default option. This usually takes a long time to run and thus is not used as the first option. This compilation could improve design performance by about 10%.

4.3.2 GROUP CRITICAL PATHS AND ASSIGN A WEIGHT FACTOR

The group_path command can be used to assign weight factor to certain paths indicating the effort that is needed for specified paths. The weight factor dictates the effort needed to spend on a path for optimization.

group_path –name <group_name> -from <starting_point> -to <ending_point> -weight <value>

4.3.3 REGISTER BALANCING

This command is more useful for pipelined designs. The command reshuffles the logic from one pipeline stage to another by moving extra logic from overly constrained pipeline stages to less constrained ones with additional timing. The command is balance_registers.

4.3.4 CHOOSE A SPECIFIC IMPLEMENTATION FOR A MODULE

Depending upon the map_effort option set, DC will automatically choose different implementations for a functional module. For example the adder has the following kinds of implementation.

a) Ripple carry – rpl

b) Carry look ahead –cla

c) Fast carry look ahead –clf

d) Simulation model –sim

Implementation types rpl, cla, and clf are for synthesis; clf is the faster implementation followed by cla; the slowest being rpl. If compilation of map_effort low is set the designer can manually set the implementation using the set_implementation command. If the map_effort is set to medium the design compiler would automatically choose the appropriate implementation depending upon the optimization algorithm.

4.3.5 BALANCING HEAVY LOADING DESIGNS

A large load cannot be driven by a single net. This leads to unnecessary delays and thus timing violations. The**balance_buffers**commandsolves this problem. When used DC will create buffer trees to drive a large fanout thus balancing the heavy load.

CHAPTER 5: IC COMPILER

IC compiler is a tool from Synopsys that is used for physical implementation. Following is a detailed flowchart for the ASIC design process.

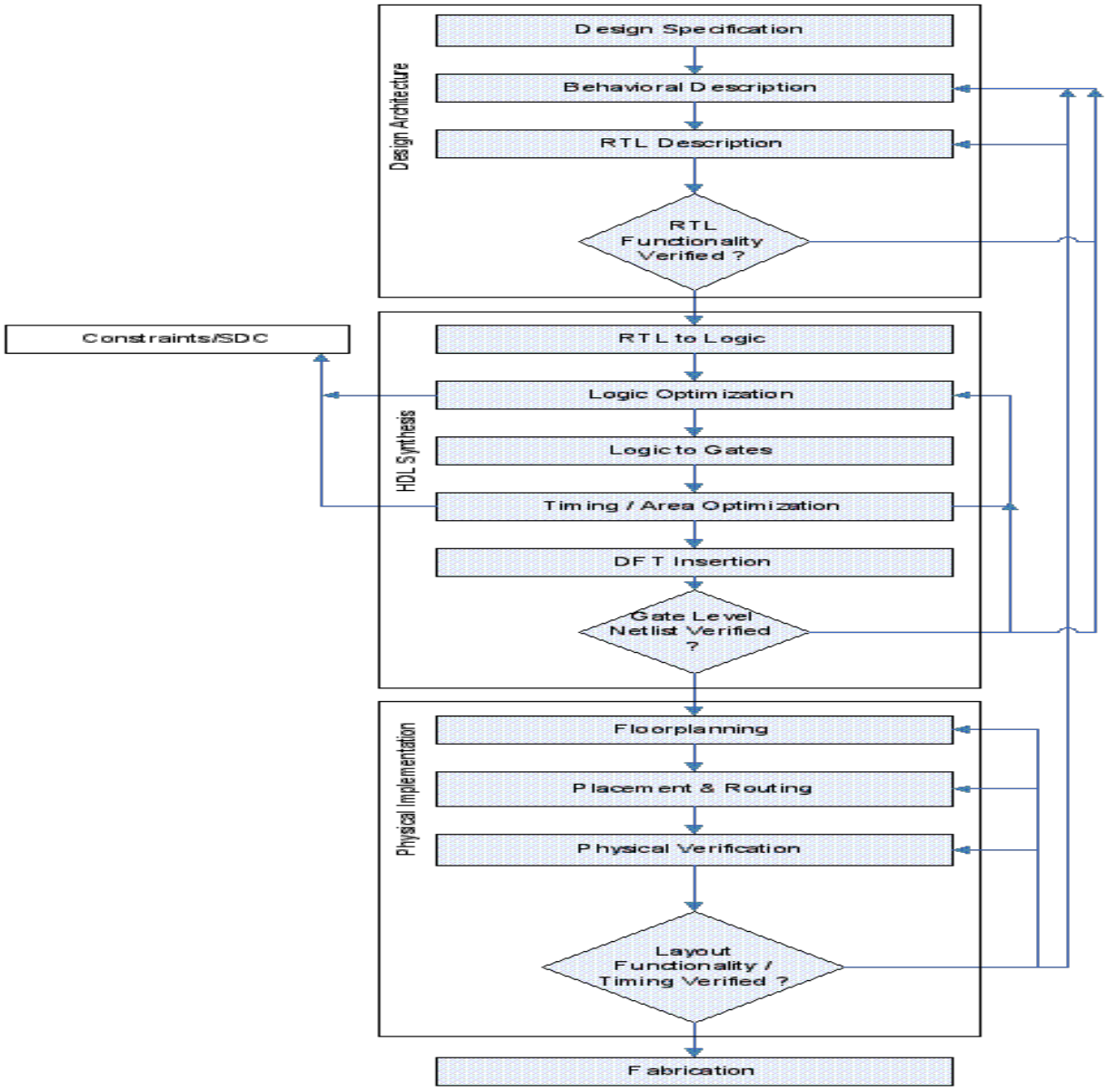

Figure 13: ASIC Design Flowchart

The physical implementation consists of three steps:

A)Floor planning

B)Placement

C) Clock Tree Synthesis (CTS)

D)Routing

5.1 Floor Planning

At the floor planning stage, thenetlist describes the design and the various blocks of the design and the interconnection between the different blocks. The netlist is the logical description of the ASIC and the floor plan is the physical description of the ASIC. Floor planning is the process of mapping the logical description of the design to the physical description. The main objectives of floor planning are to minimize

a. Area

b. Timing (delay)

During floor planning, the following are done:

- The size of the chip is estimated.
- The various blocks in the design, are arranged on the chip.
- Pin assignment is done.
- The I/O and power planning are done.
- The type of clock distribution is decided

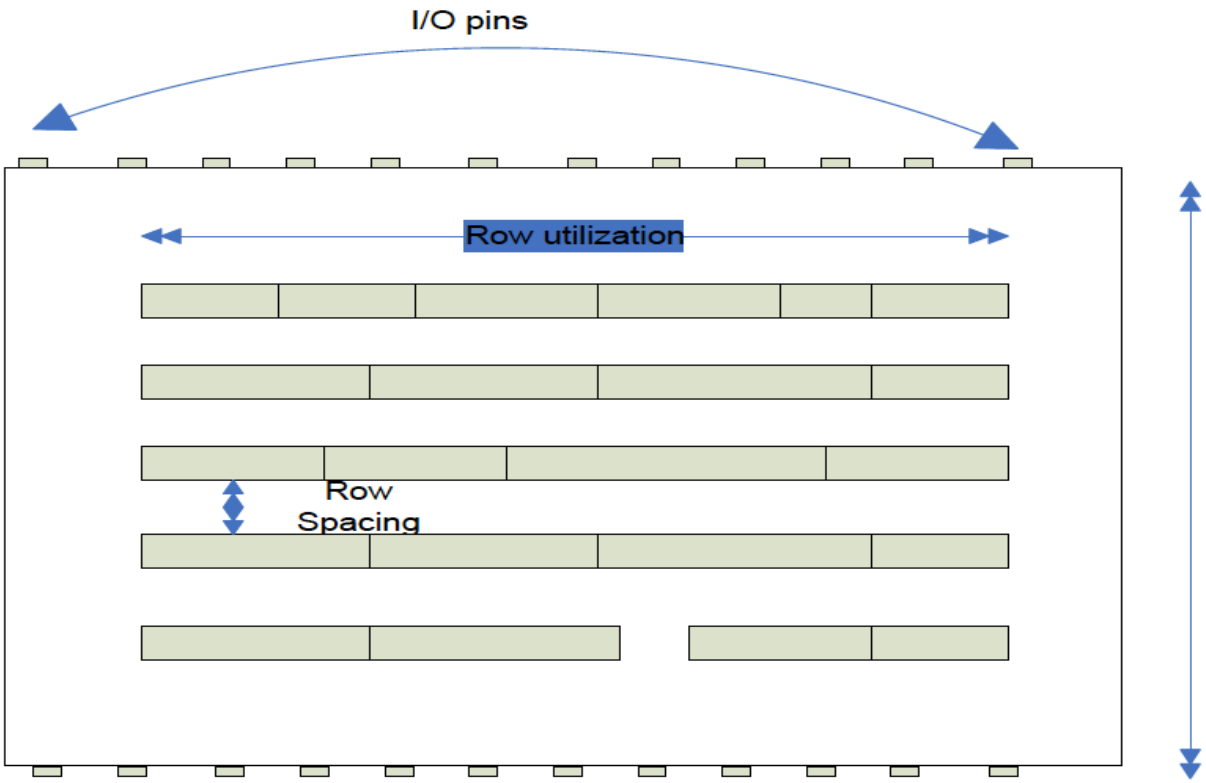

Figure 14: Floor Plan Example

The final timing and quality of the chip depends on the floor plan design. The three basic elements of chip are:

1. Standard Cells: The design is made up of standard cells.
2. I/O cells: These cells connect the chip to the outside world.
3. Macros (Memories): Sequential storage elements like flip flop take up a lot of area. Therefore special memory elements are used which store the data efficiently and also do not occupy much space on the chip. These memory cells are called macros. Examples of memory cells include 6T SRAM (Static Dynamic Access Memory), DRAM (Dynamic Random Access Memory) etc.

The above figure shows a basic floor plan. The following are the basic floor planning steps (and terminology):

- **Aspect ratio (AR)**: Aspect Ratio is defined as the ratio of the width and length of

the chip. From the figure, we can say that aspect ratio is x/y. The aspect ratio should take into account the number of routing resources available. If there are more horizontal layers, then the rectangle should be long and width should be small and vice versa if there are more vertical layers in the design.

Normally, METAL1 is used up by the standard cells. Usually, odd numbered layers are horizontal layers and even numbered layers are vertical. So for a 5 layer design, AR = 2/2 = 1.

For a 6 layer design, AR = 2/3 = 0.66.

- **Concept of Rows**: The standard cells for a particular design are placed in rows. The rows have equal height and spacing between. The width of the rows can vary. VDD and ground rails are placed on either side of the cell rows. Power for the cells in the rows can be derived from these voltage rails.
- **Core**: Core is defined as the inner block, which contains the standard cells and macros. There is an outer block which covers the inner block. The I/O pins are placed on the outer block.
- **Power Planning**: Power supply is required for a chip to work. A power ring is designed around the core. The power ring contains the VDD and VSS rings. Once the ring is placed, a power mesh is designed such that the power reaches the cells easily. The power mesh is nothing but horizontal and vertical lines on the chip. One needs to assign the metal layers through which the power needs to be routed. The VDD and VSS rails also have to be defined.
- **I/O Placement**: There are two types of I/O's.
    i. **Chip I/O:** TheI/O placement consists of the placement of I/O pins and the I/O pads.

ii. **Block I/O:** The core consists of several blocks.Eachblock has block I/O pins which communicate with other blocks in the chip.

- **Concept of Utilization**: Utilization is defined as the percentage of area that has been utilized in the chip. In the initial stages of the floor plan design, if the size of the chip is unknown, then the starting point of the floor plan design is utilization. There are three different kinds of utilizations.

  i. **Chip Level utilization:** Chip level utilization is the ratio of the area of standard cells, macros and the pad cells with respect to area of chip.

  $$\frac{\text{Area (Standard Cells) + Area (Macros) + Area (Pad Cells)}}{\text{Area (chip)}}$$

  ii. **Floor plan Utilization**: Floor plan utilizationis defined as the ratio of the area of standard cells, macros, and the pad cells to the area of the chip minus the area of the sub floor plan.

  $$\frac{\text{Area (Standard Cells) + Area (Macros) + Area (Pad Cells)}}{\text{Area (Chip) – Area (sub floor plan)}}$$

  iii. **Cell Row Utilization**: Cell row utilization is defined as the ratio of the area of the standard cells to the area of the chip minus the area of the macros and area of blockages.

  $$\frac{\text{Area (Standard Cells)}}{\text{Area (Chip) - Area (Macro) – Area Region Blockages)}}$$

- **Macro Placement**: During floor planning the initial placement of macros in the core is performed first. Depending on the placement of macros standard cells are place in the core. Blockages should be put in the areas where two macros are placed close by.This helps prevent tool from placing standard cells in those small areas and

avoid congestion.Few of the different kinds of placement blockages are:

i. **Standard Cell Blockage**: The tool does not put standard cells in the area specified by the standard cell blockage.

ii. **Non Buffer Blockage**: The tool can place only buffers in the area specified by the non-buffer blockage.

iii. **Blockages below power lines**: Blockages should be created under power lines, so that congestion problems don't occur later. If there are lot of errors post routing, use of placement blockages can ease congestion.

- **I/O Cells in the Floor plan**: The I/O cells are the cells which help the internal blocks of the chip to communicate with the outside world. The I/O cells provide voltage to the core. There are a lot of resistances and capacitances due to the different elements in the I/O construction. Due to this, the voltage may need to be higher outside so that the correct voltage can be provided to the cells inside.

  So now the next question is how the chips can communicate between different voltages?

  The answer is provided by the I/O cells. These I/O cells are basicallylevel shifters. Level shiftersconvert the voltage from one level to another. The input I/O cells reduce the voltage coming from the outside to that of the voltage needed inside the chip and output I/O cells increase the voltage which is needed outside of the chip. The I/O cells also act like buffers.

5.2 Concept of Flattened Verilog Netlist

Verilognetlist is usually in the hierarchical form. In a hierarchical design the design is broken down into sub modules. This greatly simplifies the design process. Hierarchical netlist is good only

until physical implementation. During placement and routing, flattened netlist is a better option. In a flattened netlist the blocks have been basically opened up and there are no more sub blocks. There is just one top block. This helps in achieving better routing and good quality optimization. Conventional hierarchical flow can lead to sub-optimal timing for critical paths traveling through the blocks and for critical nets routed around the blocks.

The following gives an example of a netlist in the hierarchical mode as well as the flattened netlist mode:

Hierarchical Model:

```
module top (a,out1)
input a;
output out1;
wire n1;
SUB1 U1 (.in (a), .out (n1))
SUB1 U2 (.in (n1), .out (out1))
endmodule
module SUB1 (b,outb)
input b;
outputoutb;
wire n1, n2;
INVX1 V1 (.in (b), .out (n1))
INVX1 V2 (.in (n1), .out (n2))
INVX1 V3 (.in (n2), .out (outb))
endmodule
```

In verilog, the instance name of a module is unique. In the flattened netlist, the instance name would be the top level instance name/lower level instance name etc… Also the input and output ports of the sub modules get lost. In the above example the input and output ports a, out1, b and outb get lost.

The above hierarchical model, when converted to the flattened netlist, will look like this:

Flattened Model:

```
module top (in1, out1)
input in1;
output out1;
wire topn1;

INVX1 U1/V1 (.in (in1), .out (V1/n1)
INVX1 U1/V2 (.in (V1/n1), .out (V2/n2)
INVX1 U1/V3 (.in (V2/n2), .out (topn1 )
INVX1 U2/V1 (.in (n1), .out (V1/n1 )
INVX1 U2/V2 (.in (V1/n1), .out (V2/n2 )
INVX1 U2/V3 (.in (V2/n2), .out ( out1 )
endmodule
```

The following is chart for the floorplanning process:

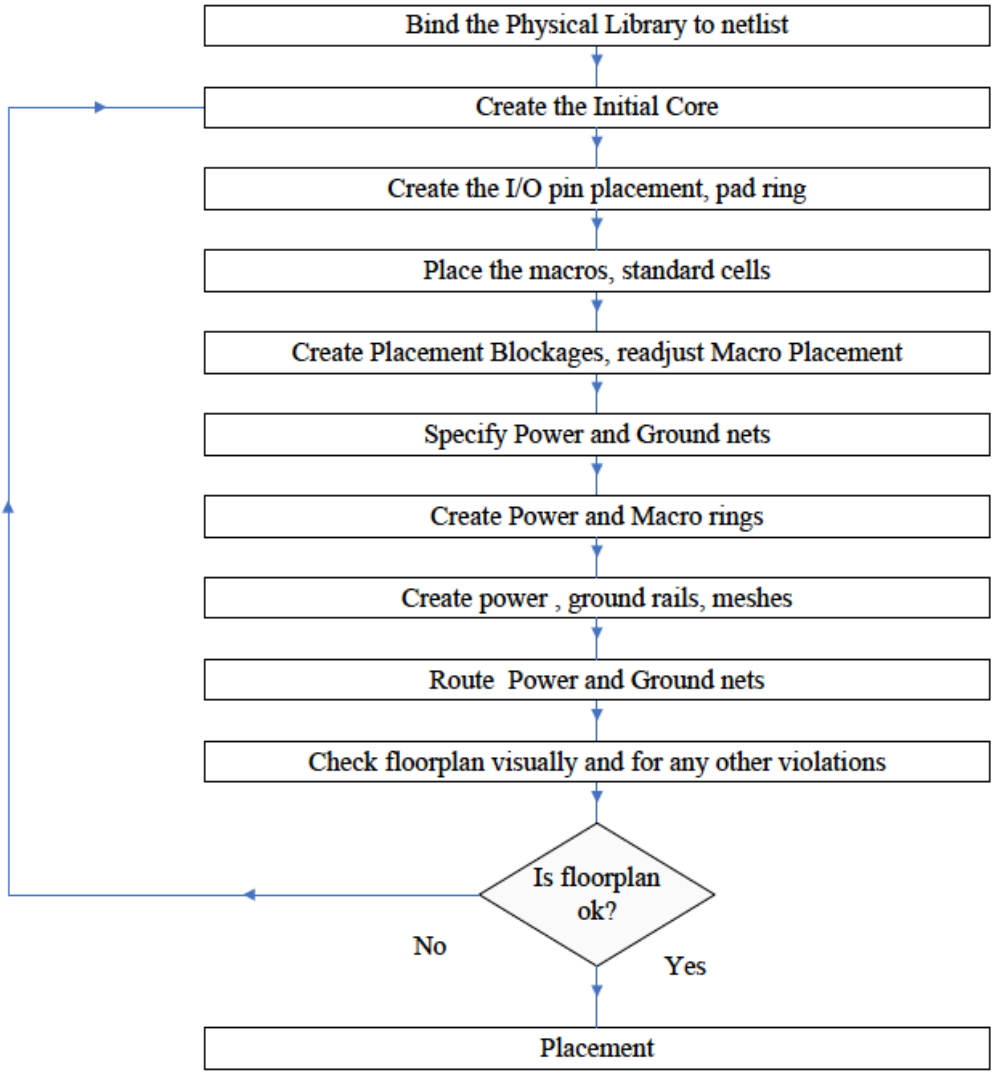

Figure 15: Floor planning flowchart

5.3 Placement

Placement is the step in the physical implementation process where place for the standard cells are allocated on the row. Space is set aside for interconnect to each logic/standard cell. Once the placement process is complete we can get accurate estimates of the capacitive loads that each standard cell has to drive. The tool places these cells based on the algorithms used internally.

The main objectives of the placement algorithm are:

- Making the chip as dense as possible (Area Constraint)
- Minimize the total wire length (reduce the length for critical nets).The number of horizontal/vertical wire segments crossing a line.

Constraints for doing the above are:

- The placement should be routable (no cell overlaps; no density overflow).
- Timing constraints are met

There are different algorithms to do placement. The most popular ones are as follows:

a. **Constructive algorithms:** This type of algorithm uses a set of rules to arrive at the optimized placement. Example: Cluster growth, min cut, etc.

b. **Iterative algorithms:** Intermediate placements are modified to achieve a better design. Already constructed placements are used initially and iterates on that to get a better placement.

   Example: Force-directed method.

c. **Nondeterministicapproaches:** simulated annealing, genetic algorithm, etc.

5.4 Clock Tree Synthesis

Clock tree synthesis is a layout technique used to provide balanced buffer distribution to clock pins in an attempt to minimize the clock skew [19]. The following figure shows balancing effect between various connections and wire lengths:

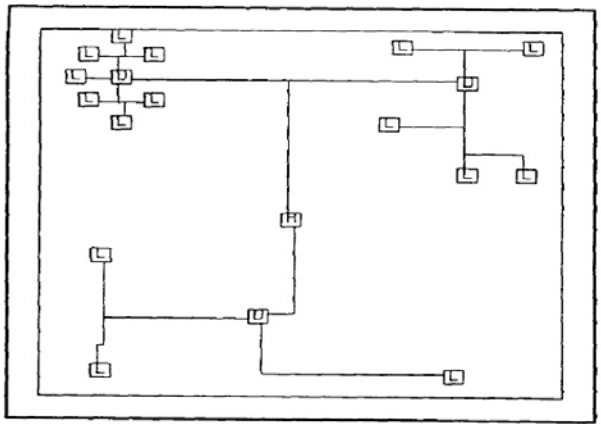

Figure 16: Clock Tree Synthesis Strategy

In Figure 16 the point 'R' is termed as the root point for clock distribution. The root drives the distribution macros 'D'. The distribution macros will drive the leaves 'L' which are clock inputs to sequential macros or another level of distribution macros as illustrated in Figure 17.

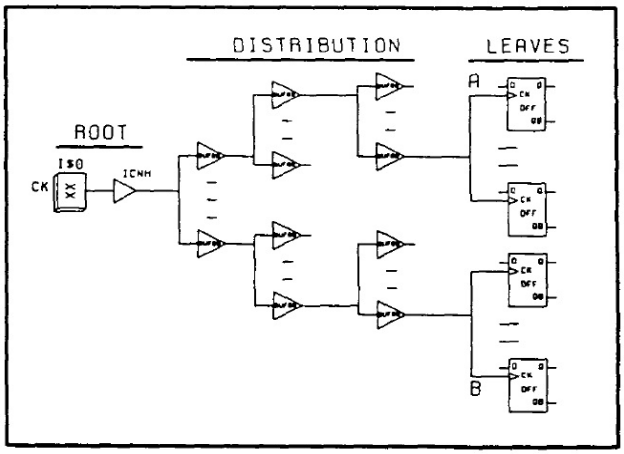

Figure 17: Clock Distribution Levels

The tree in Figure 17 is said to have three levels. The function of the distribution level is to achieve balanced loading when driving the leaves.

Figure 18 shows three waveforms. The waveform labeled CK is the input to the root CK in Figure 17.Waveforms A and B are the clock inputs to two sequential elements (D type flip-flops) on different branches. The time t1 and t2 are termed as clock insertion delays which is the time required to propagate through the clock distribution. The difference between t1 and t2 is termed as

clock skew.

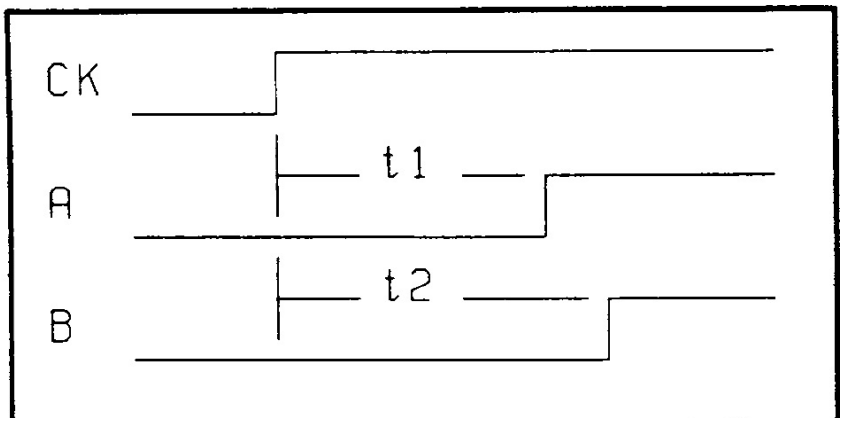

Figure 18 : Waveform 1

Interconnect and loading capacitances are the largest effect of balancing [19]. There are various aspects like metal capacitance for each routing level, their individual capacitance effects, fringing and parallel plate, and the physical size difference of macros which cannot be controlled by the designer [19]. The only aspects controlled by the designer are the choice of macros and buffer distribution.The factors influencing balancing are the layout tool, the size of the array, and the existence of large blocks [19].

The capability if the layout tool is fixed. The designer has to use the command appropriately to get the best out of the tool. The layout tool has difficulty placing and routing the blocks that is driven by the root. The larger the area the more difficult it will be finding the center of gravity points for all the destination inputs [19]. The existence of large blocks can disturb the balancing process by making the layout tool work around these blocks.

The strength of the root macro is very critical as it determines the skew and insertion delays [19]. Best results can be achieved by using larger buffers at the root followed by medium strength buffers at the distribution level that drives the leaves [19]. Usually the last level contains the most number of distribution buffers [19]. Therefore using low drive strength buffers here will prevent

excessive power dissipation still conforming to the clock skew requirements [19].

5.5 ROUTING

Next step after floor planning and placement is routing. Routing is the process of connecting various placed blocks. Until now, the blocks were only just placed on the chip.

Routing also is spilt into two steps:

1. **Global routing**: Global router plans the routing between different blocks placed. The main objective here is to minimize the total interconnect length and critical path delay.

The chip is divided into small blocks called routing bins. The size of the routing bin depends on the algorithm the tool uses. Each routing bin is also called a gcell. Each gcell has a finite number of horizontal and vertical tracks. Global routing assigns nets to specific gcells but does not define the specific tracks for the nets. The global router connects two different gcells from the center point of each gcell.

The global router keeps track of the total number of interconnections going in each direction. This is called the routing demand. The number of routing layers available depends on the design. Each routing layer has a minimum width spacing rule, and routing capacity.

EX: Consider a 5 metal layer design. If metals 1,4,5 are used for intercell connection, pin, vdd, vss connections then only metals 2 and 3 are available for routing. The routing demand going over the routing supply results in congestion causing DRC errors.

2. **Detailed Routing**: In this step the nets are connected. Actual via and metal connections are created. The main objective in this step is to minimize the total area, wire length, delay in the critical paths. Each layer has its own routing grid, rules. During the final routing, the width, layer, and exact location of the interconnection are decided.

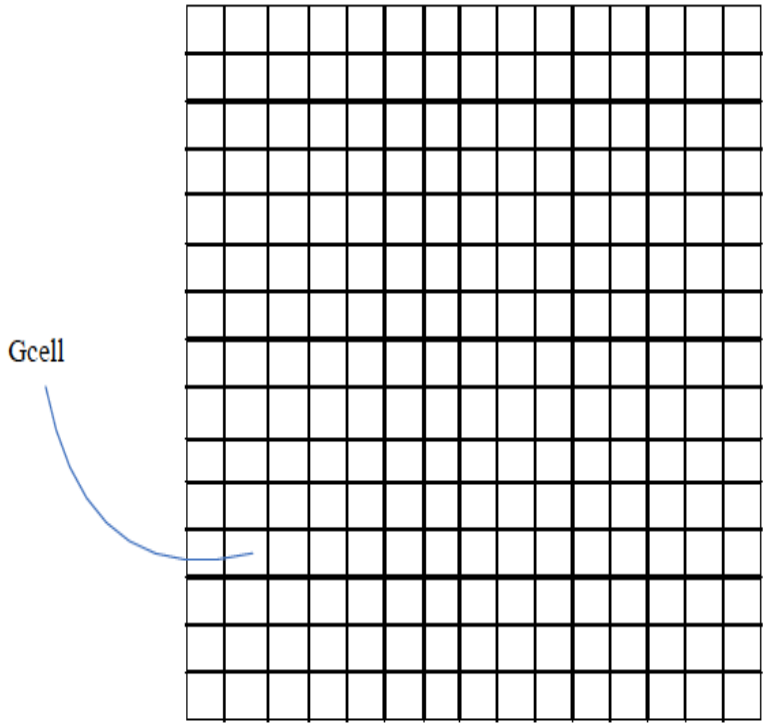

Figure 19: Floor partitioned into gcells

After detailed routing is complete, the exact length and the position of each interconnect for every net in the design is known. The actual delay estimation is now performed by extracting the parasitic capacitance and resistance. The parasitic extraction is done by extraction tools. This information is used by the static timing analysis tools to generate accurate timing statistics. After timing is met and verification is performed such as DRC and LVS the design is sent to the foundry to manufacture the chip.

CHAPTER 6: FULL METHODOLOGY

6.1 Design Compiler

1) Invoke the script with **"dc_shell-t -f synthesis.tcl"** after modifying the script for your design. You will need to update the script for your design and library locations. Comments are included.

2) After synthesis, you will have a post_synth netlist and a sdc file constraining your timing.

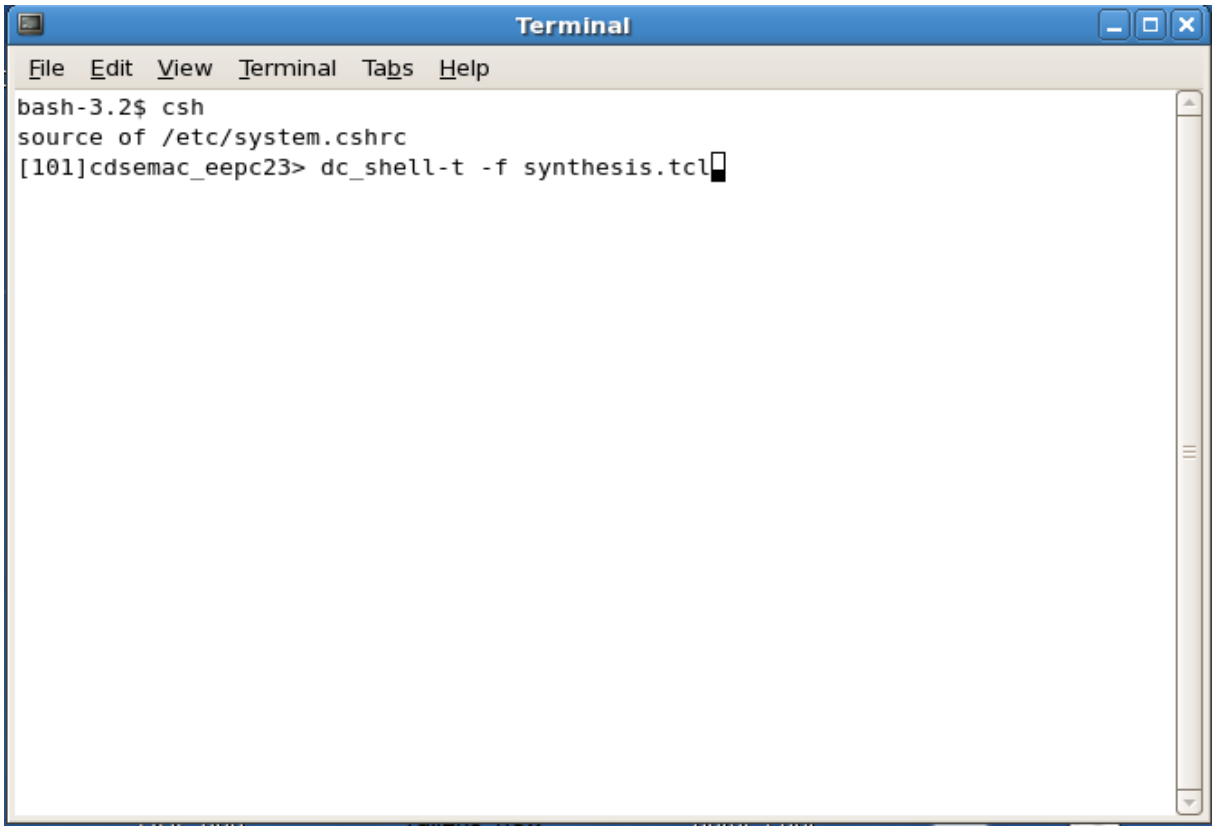

Figure 20: Design Compiler Invoke

6.2 IC_Compiler

1) Change the directory to the directory with the startupICCscript(icc_ONC5_version3.tcl).

2) Execute the following on the command line: **icc_shell -gui -f icc_ONC5_version3.tcl**

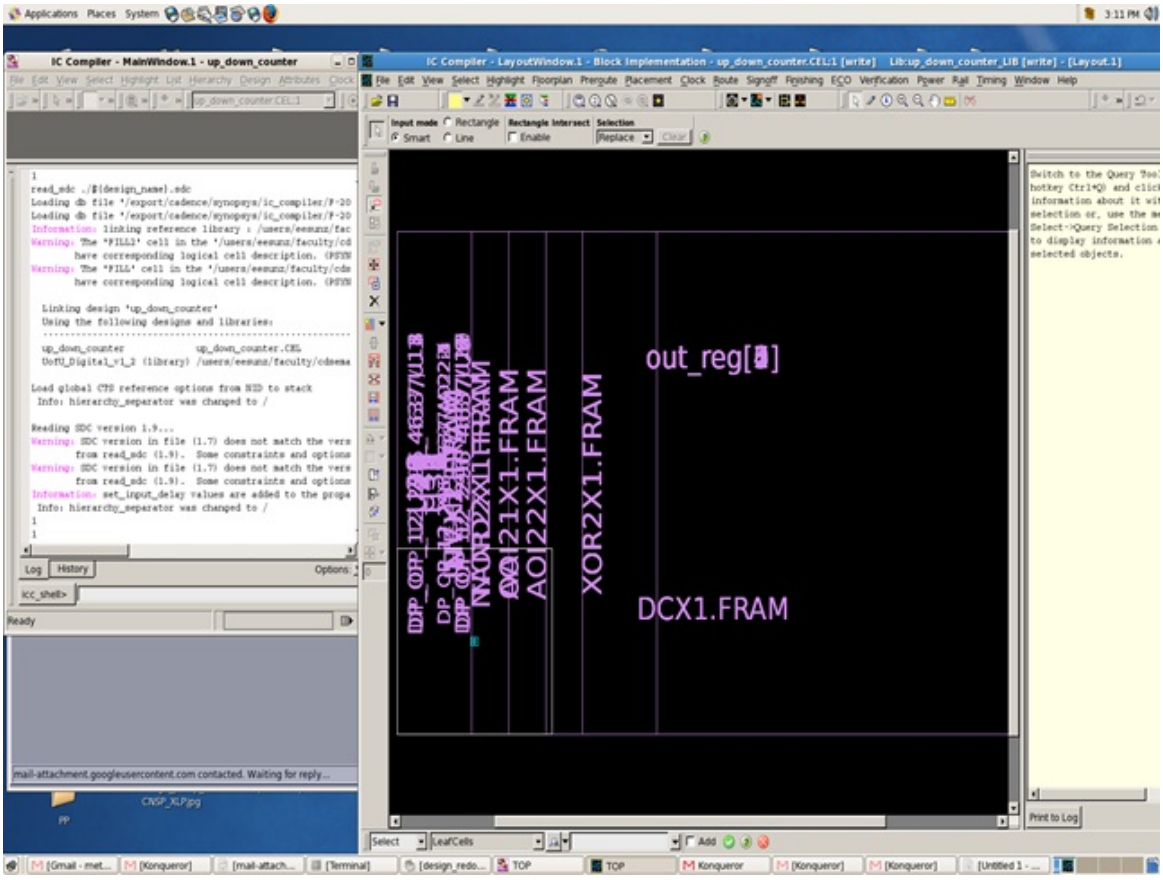

Figure 21: IC compiler design import

3) The following command will allocate space for the chip and place the pins evenly on the border. The purple boxes on the right are the unplaced cells. The core_utilization will determine how dense the design will be at the expense of routability later.

create_floorplan -control_type "aspect_ratio" -core_aspect_ratio "1" -core_utilization "0.4" -row_core_ratio "1" -start_first_row -left_io2core 24 -bottom_io2core 27 -right_io2core 24 -top_io2core 27

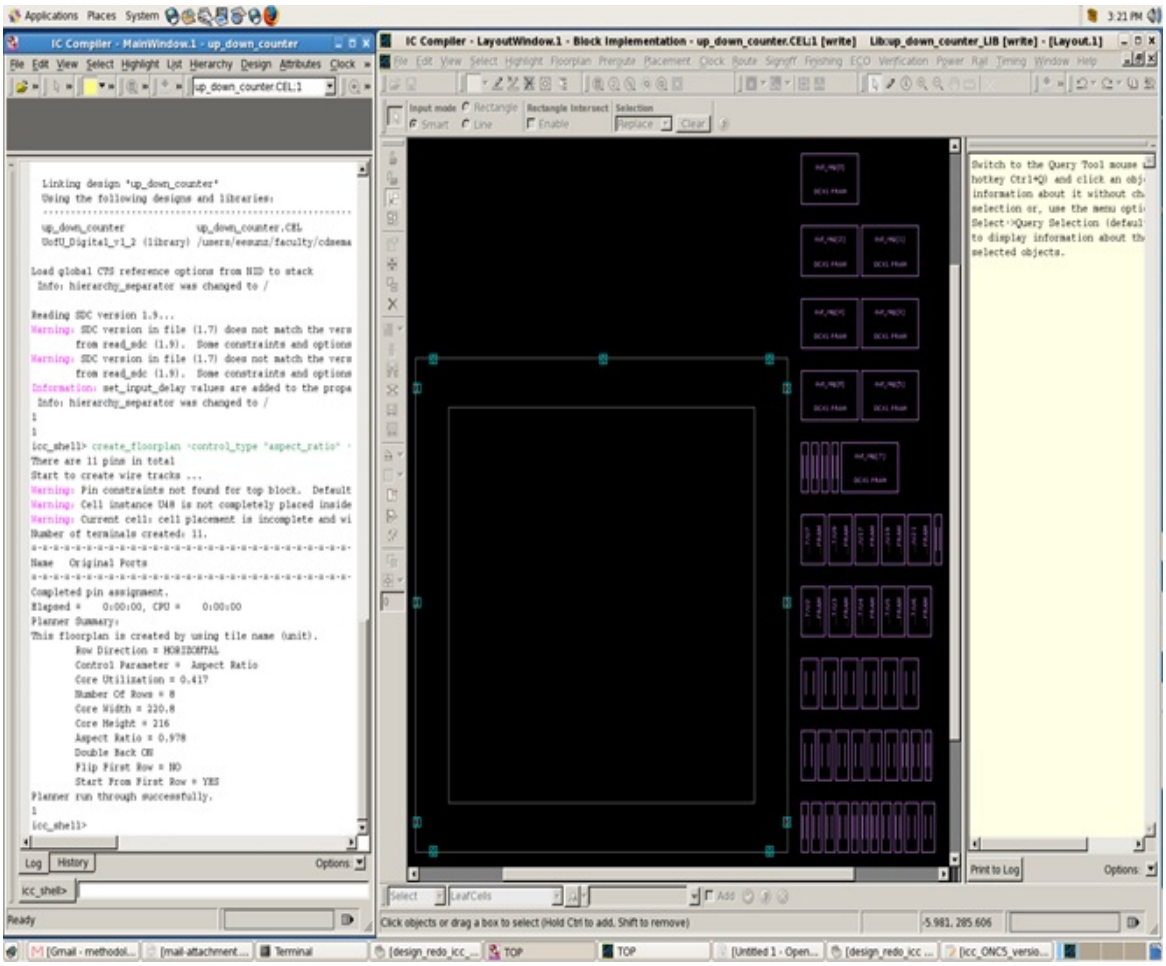

Figure 22: Floor Plan creation

4) Now we specify the vdd and gnd used by the standard cells using the following command:

derive_pg_connection -power_net {vdd!}-ground_net {gnd!}

5) The following command will create power rings around the edge:

create_rectilinear_rings -nets {vdd! gnd!} -offset {3 3} -width {4.5 4.5} -space {3 3}

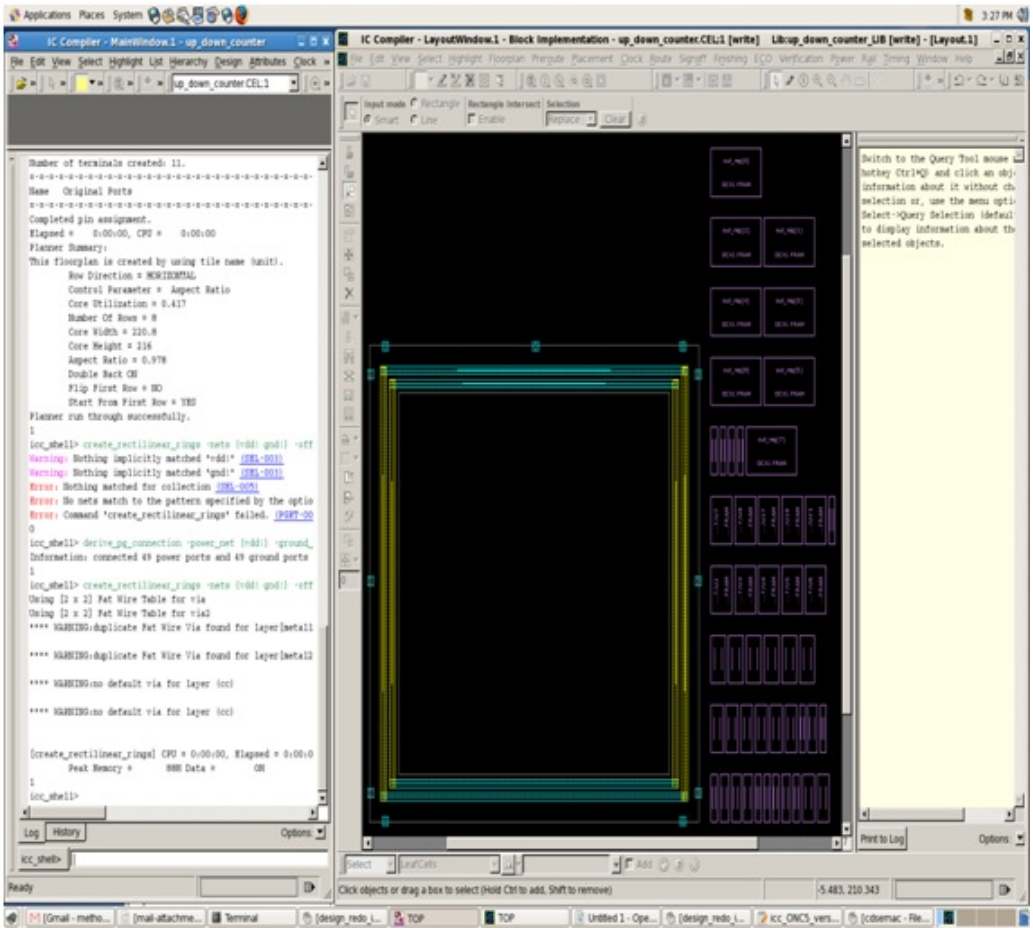

Figure 23: Power Ring creation

Blue is metal 1, yellow metal 2 and Red is metal 3. Note also that via1 squares connect metal 1 to 2 and via2 squares connect metal 2 to 3.

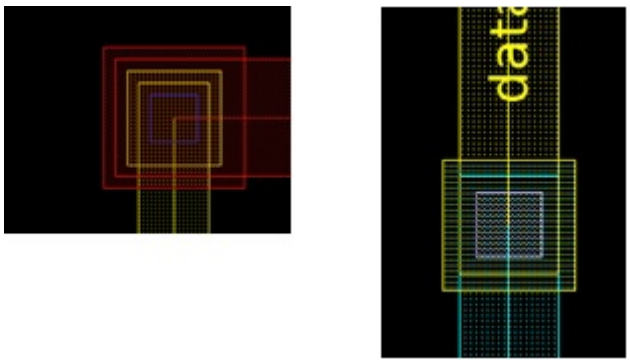

Figure 24: : Metal layers

6) This command creates additional power network robustness but at the expense of routability later.

create_power_straps -direction vertical -num_placement_strap 1 -start_at 400 -increment_x_or_y 200 -nets {vdd! gnd!} -width 1.800 -layer metal3

7) The command "place_opt" places cells. The options "-effort high" and "-congestion" may help. The congestion option spreads cells apart that may be in areas of high routing to provide extra tracks and increase the odds of a successful route later.

place_opt -effort high -congestion

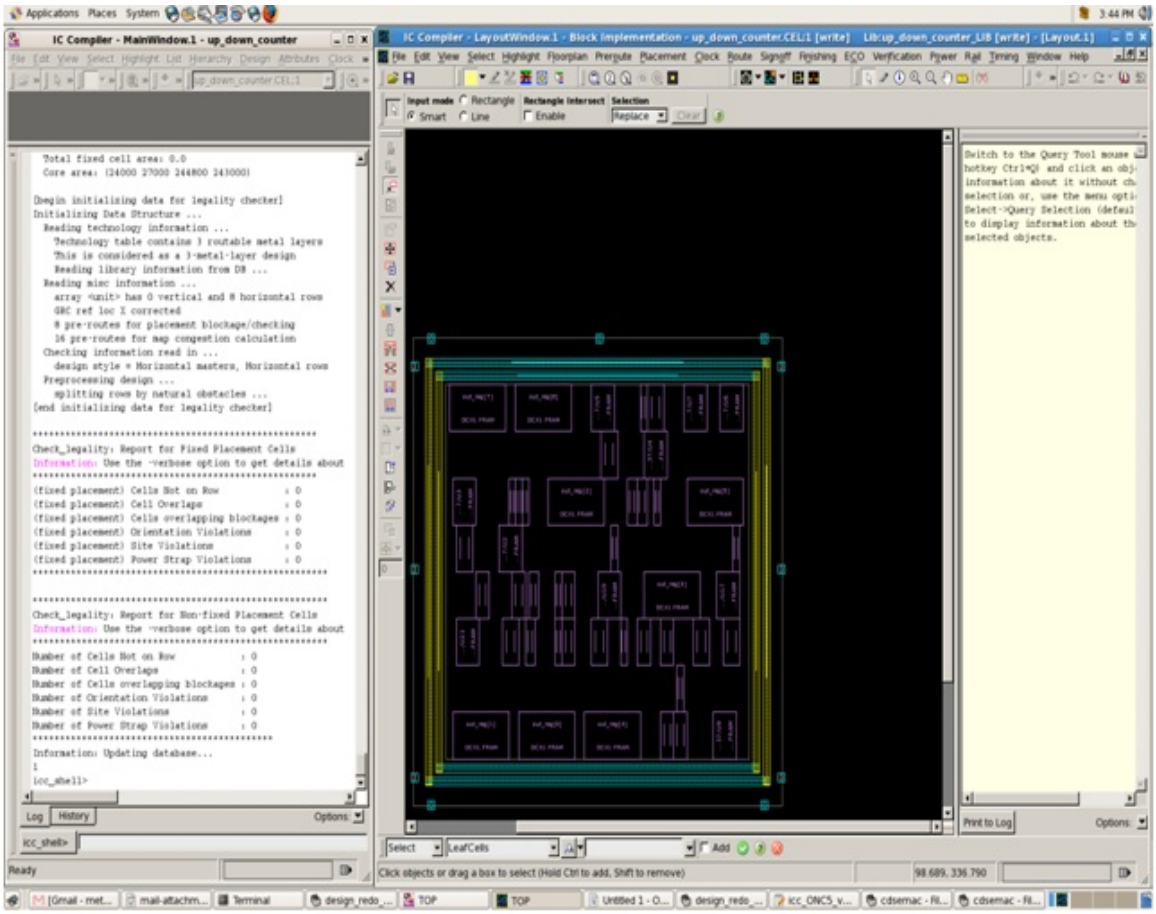

Figure 25: Placement of cells

8) This command connects the rows to the rings/straps.

preroute_standard_cells -nets {vdd! gnd!} -connect horizontal -extend_to_boundaries_and_generate_pins

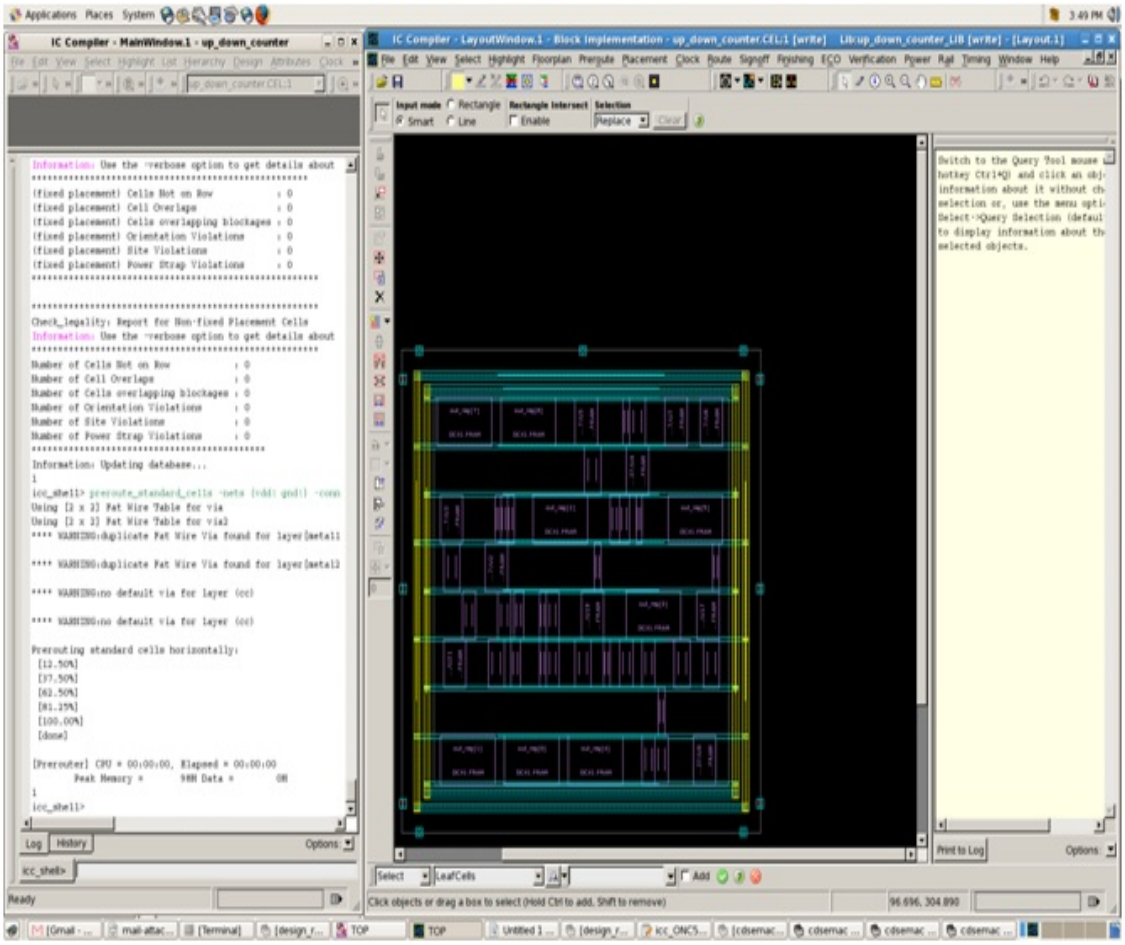

Figure 26: Adding power straps

9) This command inserts the clock tree. See the metal 2 and 3 lines below which are the clock tree.

clock_opt -fix_hold_all_clocks

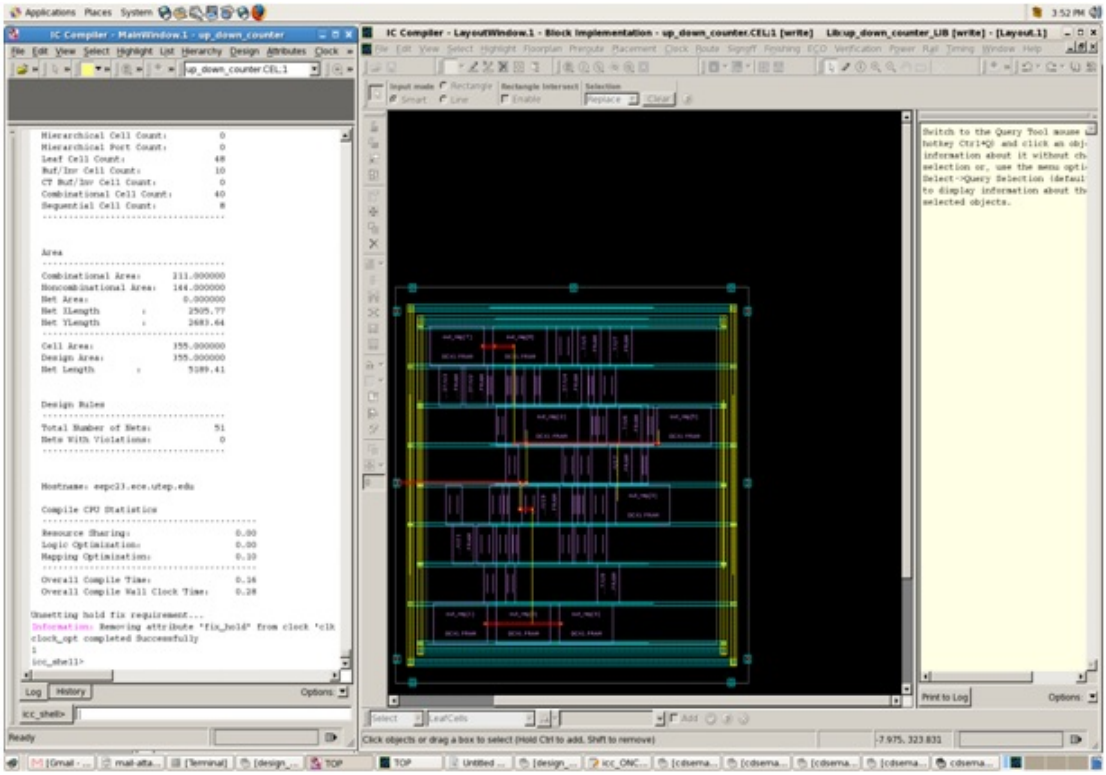

Figure 27: Inserting clock tree

10) **report_clock_tree** command reports the clock tree information. A clock skew less than 200ps is a good rule of thumb.

11) **report_timing** command shows the worst case path. Positive slack is good. A negative slack means there's a problem.

12) **route_opt –effort high** routes the rest of the design. This step takes the longest time and can be where the most problems occur if you design is too dense.

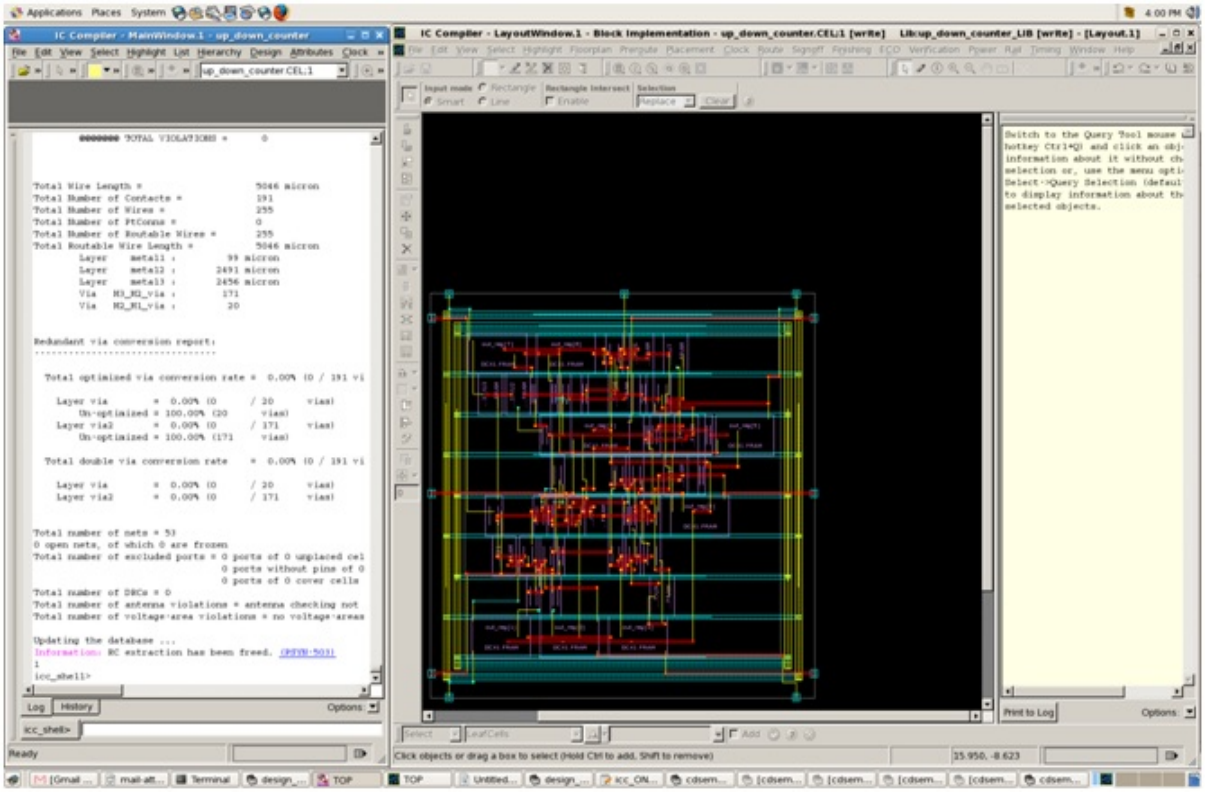

Figure 28: Routing

13) Check the "route_opt" log for violations and good timing. To avoid violations, makeyour design less dense, use less straps for power, or use a technology with moremetallayers. These may not be an option normally. This command will fix many problems:

Route_search_repair -rerun_drc -loop "100"

14) After everything else is finalized, we need to "fill" the empty space between cells to ensure continuity across the circuit rows. No more cells can be added after this step so we are almost done.

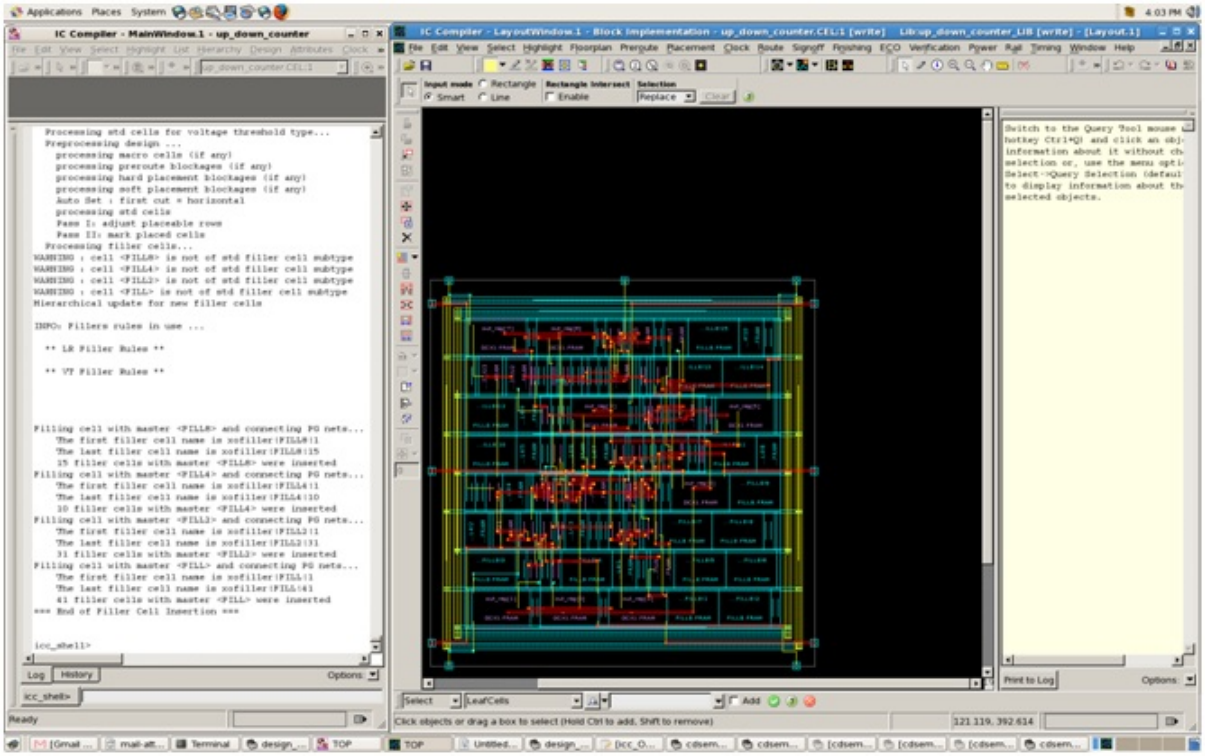

Figure 29: Adding fills

In the above figure the purple cells are the standard cells and blue are fill.

15) DRC is Design Rule Check. This is a check to make sure whether the geometry of the physical layout generated is in accordance with the Design Rules.

16) LVS is Layout Versus Schematics. Basically this is a test for your connections. There are problems reported below. Under the verification menu, if two shorts exist between ground and vdd with the net NULL. Rerun the **derive_pg_connections** command from the earlier in the stop and re-run LVS.

6.3 Custom designer

1) Create a working directory.

2) Add cds.lib, lib.def and color map files into the working directory.

3) Invoke the tool with **cdesigner&.**

4) Open the library manager and create a new library.

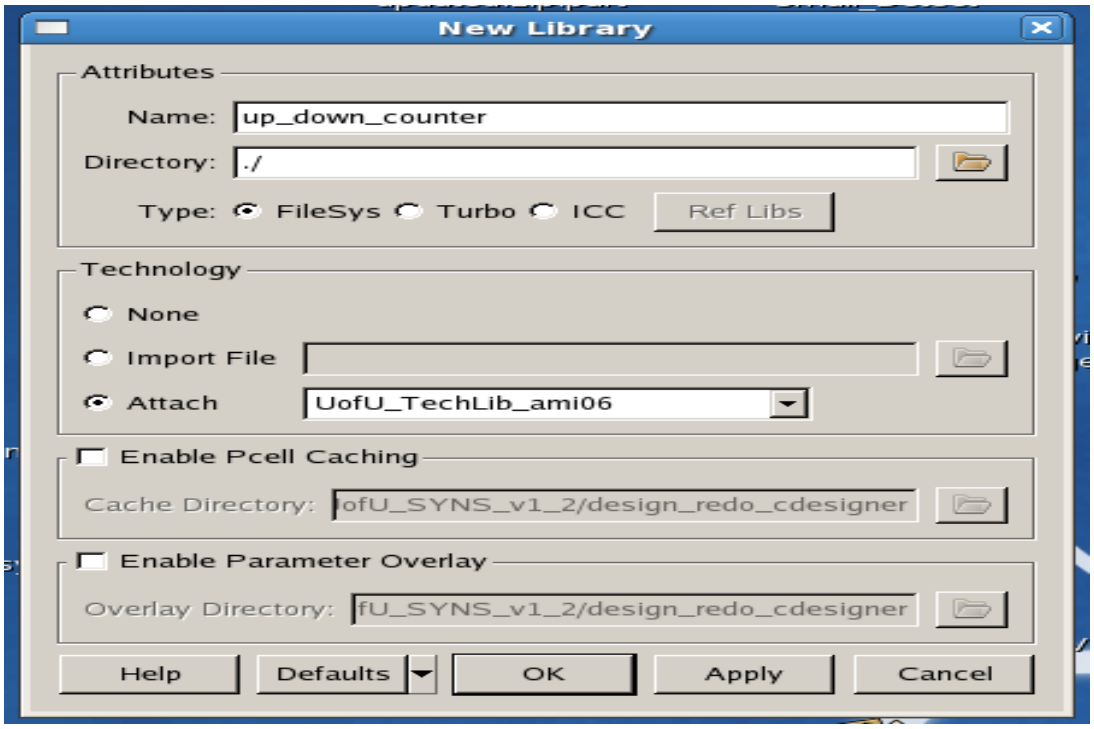

Figure 30: Creating new library

5) Fill the user created working directory:

a) Select and copy the cells from the UofU_Digital_v1_2 library into your new library using Library Manager. Right click on the cells to copy. When copying libraries you may see a dialog box about replacing cells. These are the same so just re-copy and replace redundant cells.

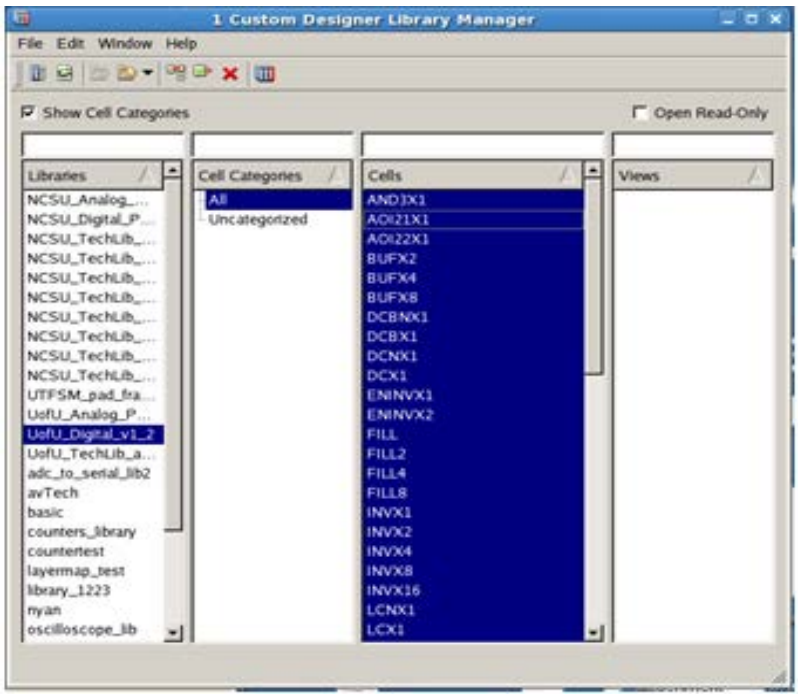

Figure 31: Copying contents to newly created library

b) Also copy the UTFSM_pad_frame cell into your library and rename to final_chip. You will edit this cell by including your core and connecting it in layout and schematic. This will be the final design to be fabricated.

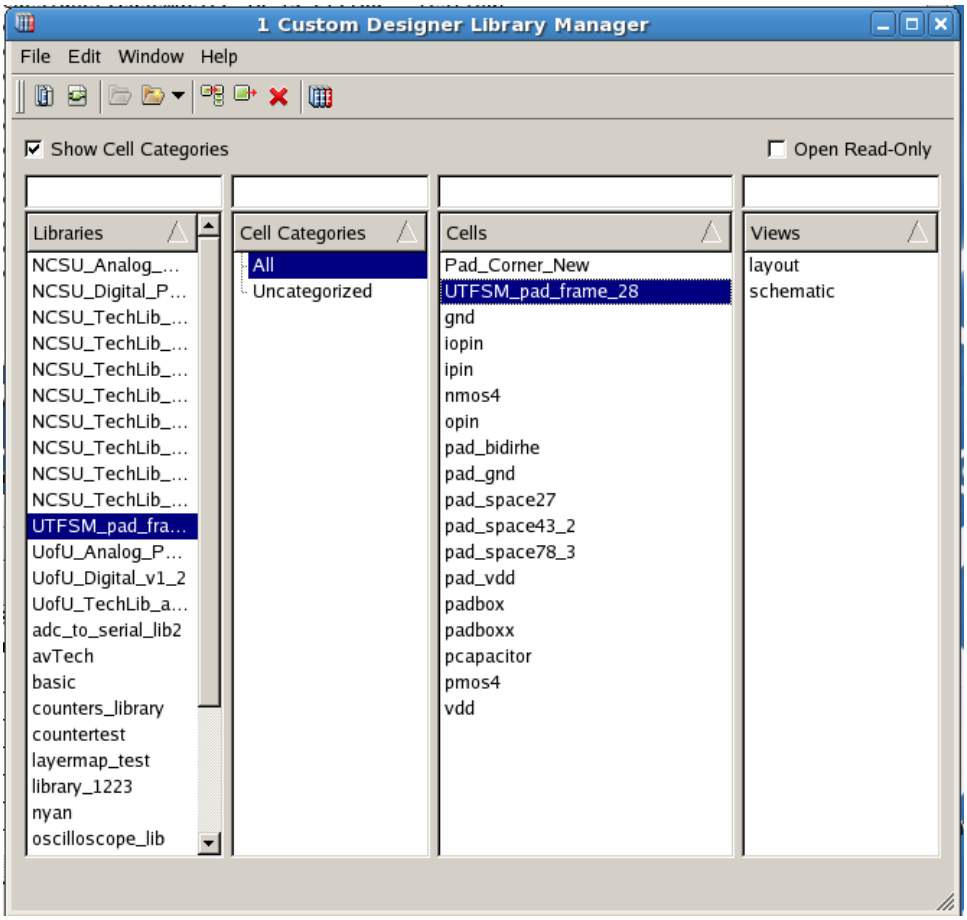

Figure 32: Adding the pad frame to the new library

6) Importing the GDS:

   a) From the Console, choose File > Import > Stream.

   b) Under the Main tab, specify therun directory,input GDS Stream file and top cell, Output details: Output Library, view: Layout

   c) Under the Options tab, specify optional details. Under the Map Files tab, specify optional map file details. Click OK.

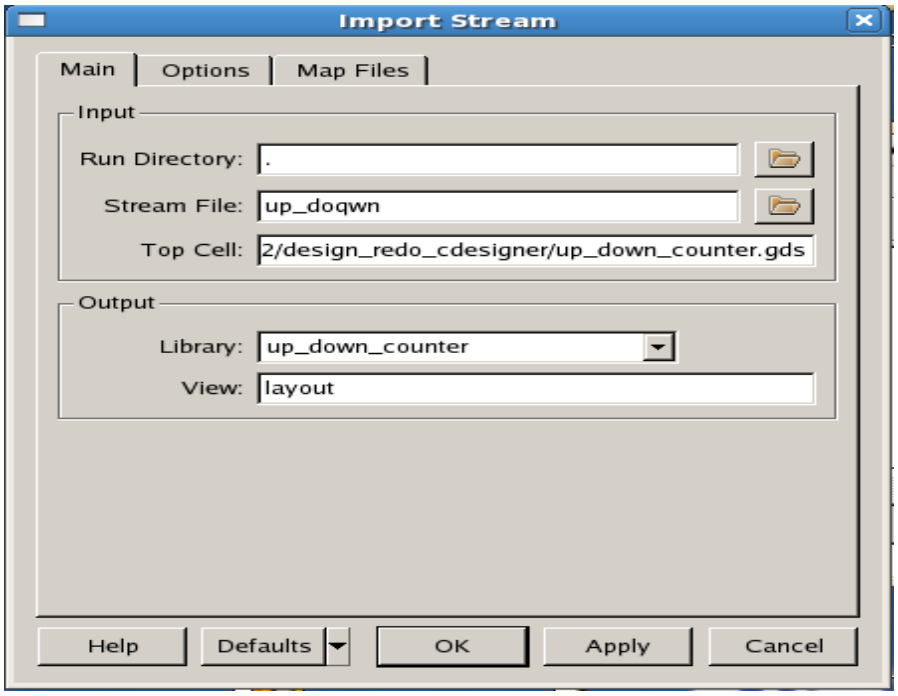

Figure 33: Import GDS (main)

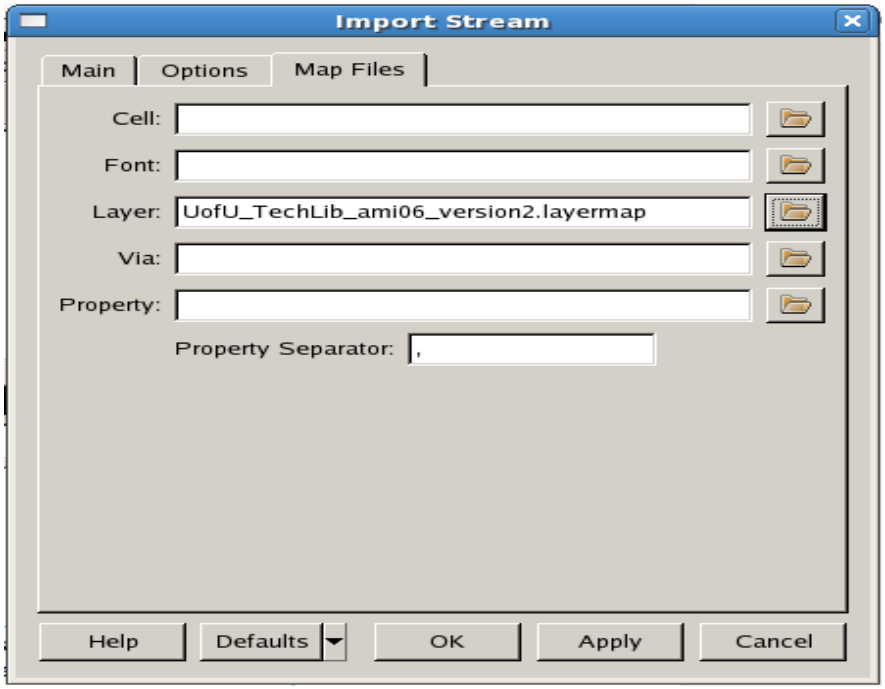

Figure 34: Import GDS (layer map file)

7) Import Verilog to create Symbol and Netlist view for the Core in Custom Designer:

a) From Custom Designer Console : Go to File --> Import --> Text.

b) Select the language as Verilog

c) Browse for your post-place- and-route netlist. Remove fill cells first. Use the following Unix/Linux command to remove fill:

grep –v FILL netlist.v>netlist.nofill.v

d) Choose the library that you created previously.

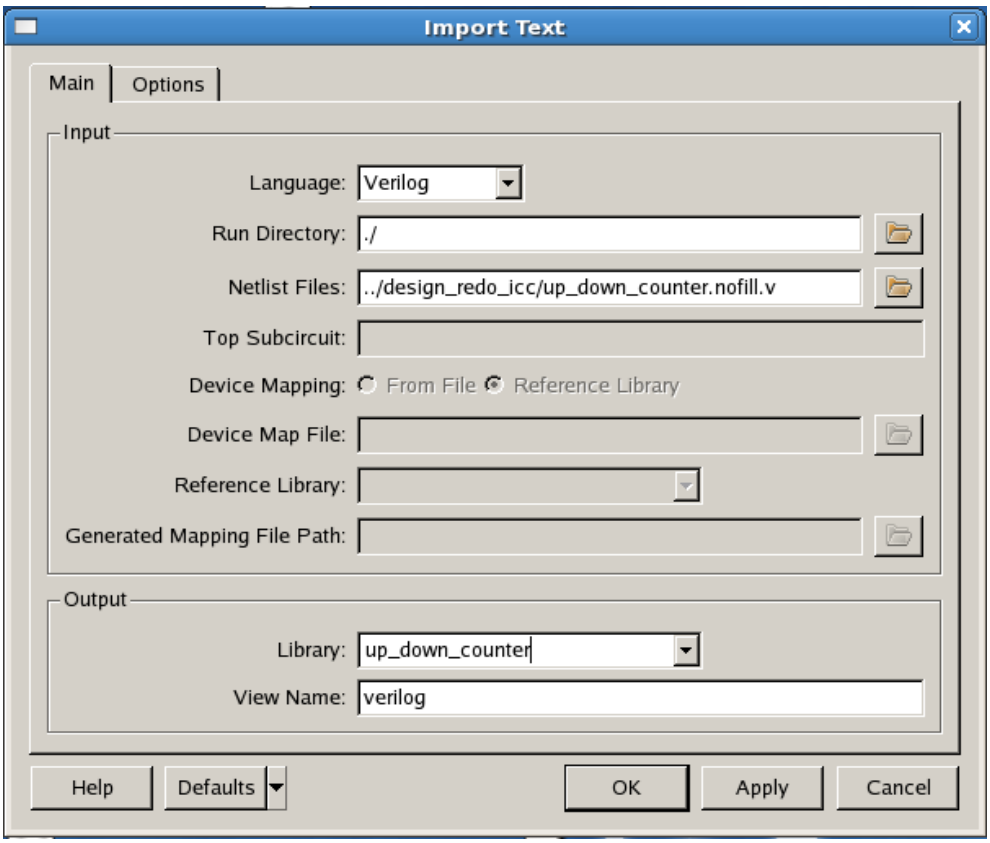

Figure 35: Creating symbol and schematic (main)

e) Under the Options tab select Generate symbols and Generate Schematics.

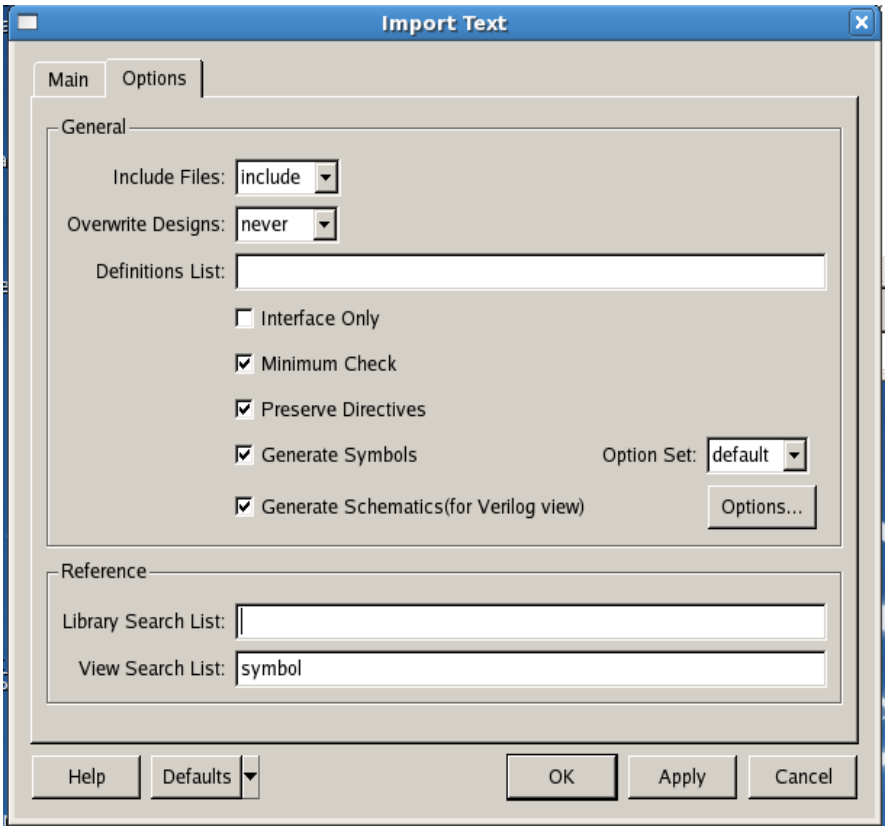

Figure 36: Creating symbol and schematic (options)

f) Click Ok to create a symbol for your chip level schematic and a schematic for LVS.

g) Go to your core cell view and double click the symbol to see the symbol. This will be used in your chip level schematic to represent and to connect to your core.

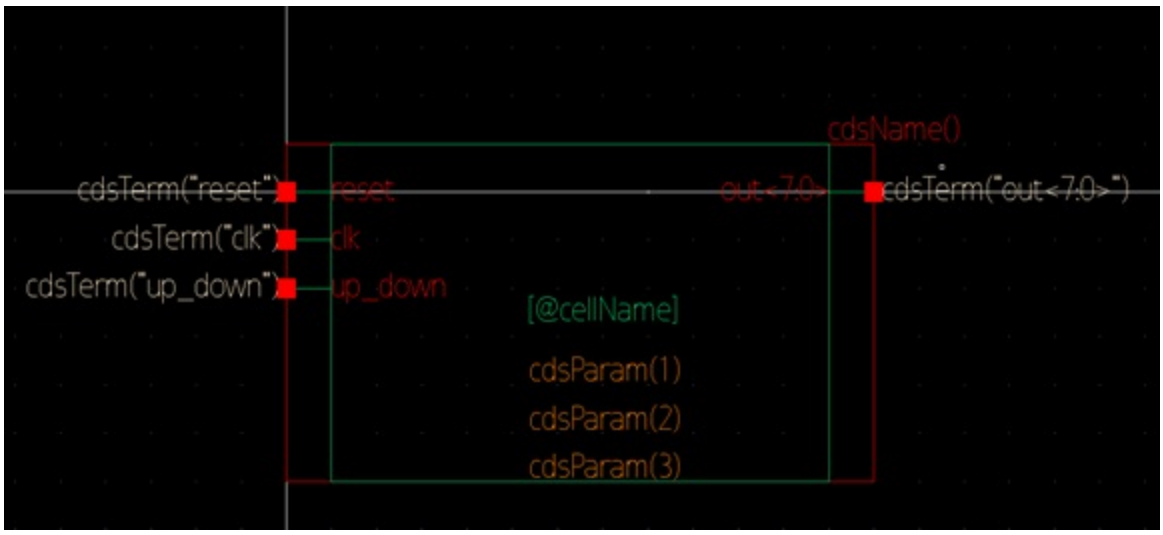

Figure 37: Symbol created

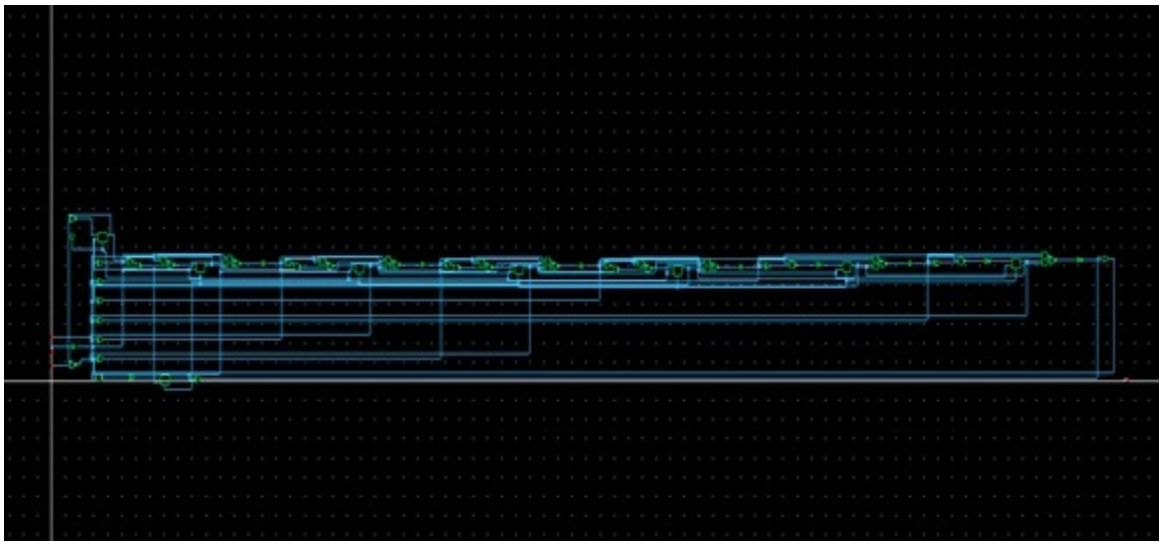

Figure 38: Schematic created

8) Fixing the Core Ports:

The GDS import will have included text for the pins of your core but pins must be instantiated to tell LVS where connections in layout will be made. We lost this info when we converted to and from GDS. Follow the following steps to fix the core ports:

a) Click the text name near the port, press q for properties and change the layer to text drawing.

b) Click the metal square at the end of the port, press q and update the net name to match the text however change square brackets to triangle brackets for the indices.

c) Press create pin , type the name of the net into the tool bar input, press enter and select two opposing corners of the pin to define the connection point. You can easily do many pins consecutively. Remember to select the correct metal (metal 2 or 3). This is symbolic for the tool and provides information as where your core should be contacted.

d) Select the newly created pin, press q and change the input Output to either input or output.

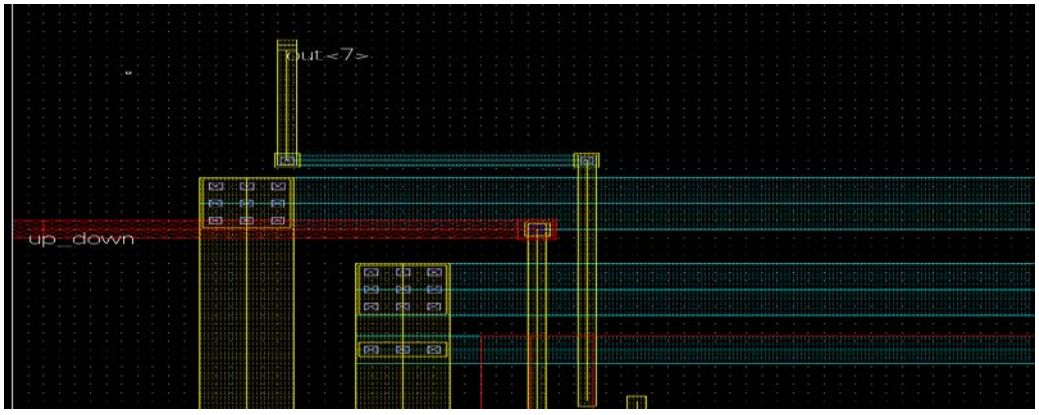

Figure 39: Core ports

9) Editing the Chip Layout:

Use i (or edit->add instance) to create an instance. Select your core and place in the center.

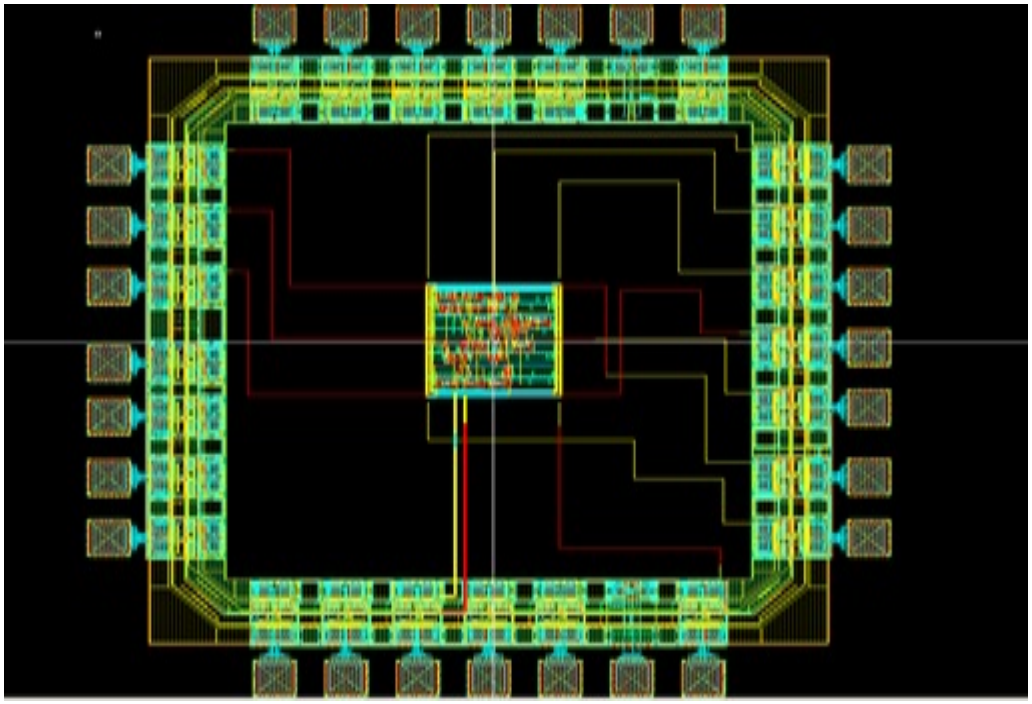

Figure 40: Chip layout

10) Add connections to layout:

Create Interconnect or press p to add wires to make IO connections, shift-v and Ctrl-v to switch metal layers through avia as you connect from one pin to the other. Here is a picture of some connections between the core and padring.

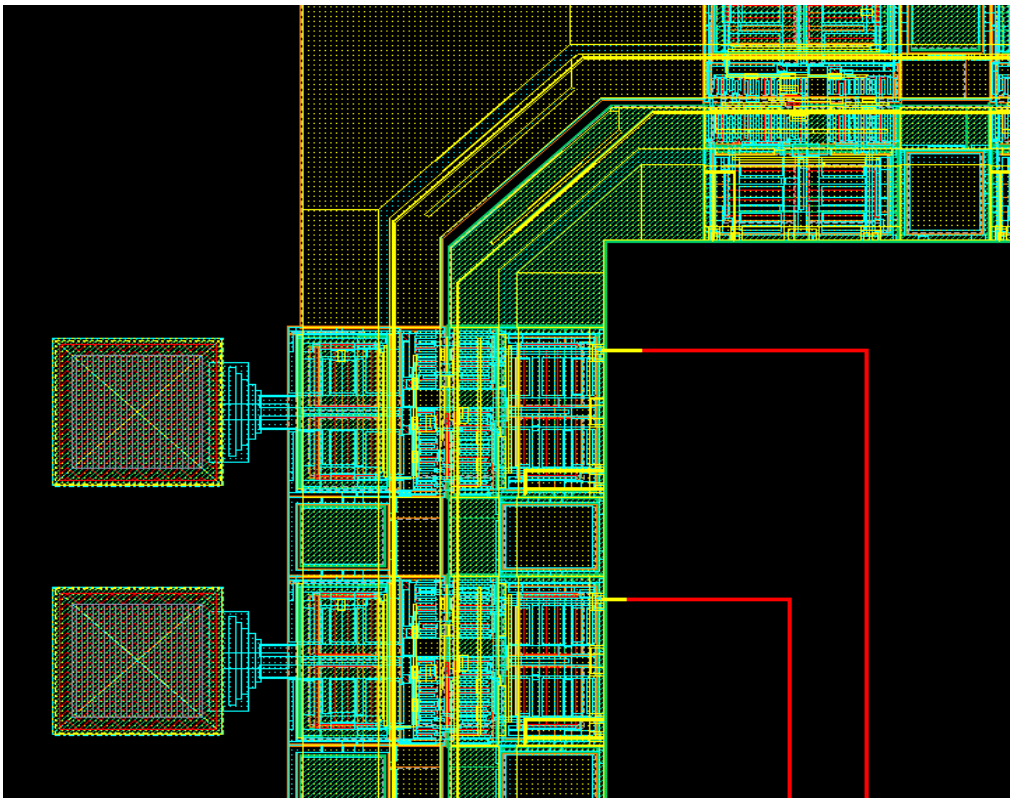

Figure 41: Chip connections

All I/O's have an enable which should be tied to ground (inputs) or vdd (outputs). Tie the signal to either a !gnd pin (purple box in the inner I/O ring) or !vdd pin (purple box in the second ring from the inside). If I/O is an output, tie the Data Output signal to the appropriate pin on your core.

Let the two Data Input signals float. If I/O is an input, tie the Data Input signal to the appropriate pin on your core. Tie the Data Output pin to ground. Data Input N can float. On four sides we need to connect the power ring of the IO to the power ring of the core. We need similar connections between the ground rings as well. When you press p for creating interconnect, a width option appears in the menu bar. Use 0.9u for signals and 4.5u for power. Put at least one ground and vdd connection per side of the core. More is better.

11) Vdd and Gnd Connections:

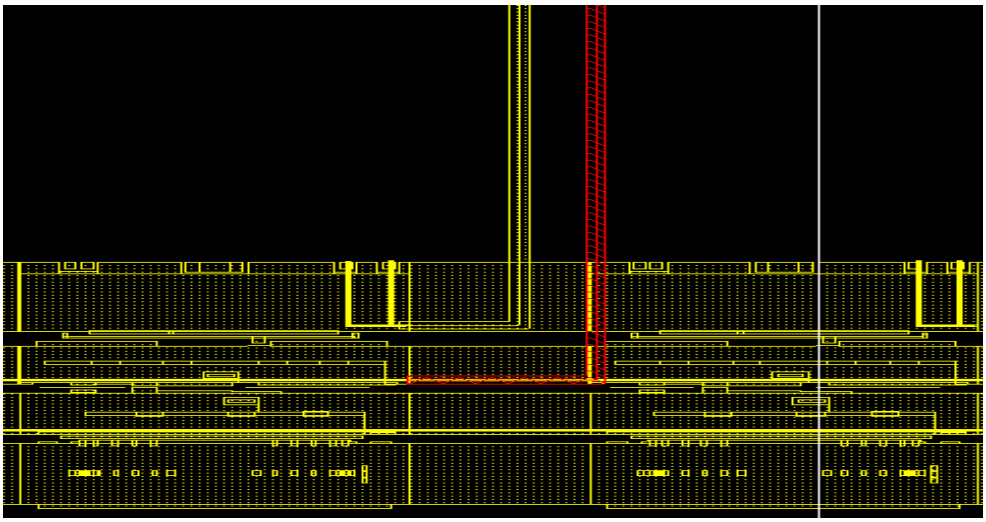

Figure 42: Vdd and Gnd connection

Yellow is the gnd and Red is the vdd.

Enable Tied low for data input:

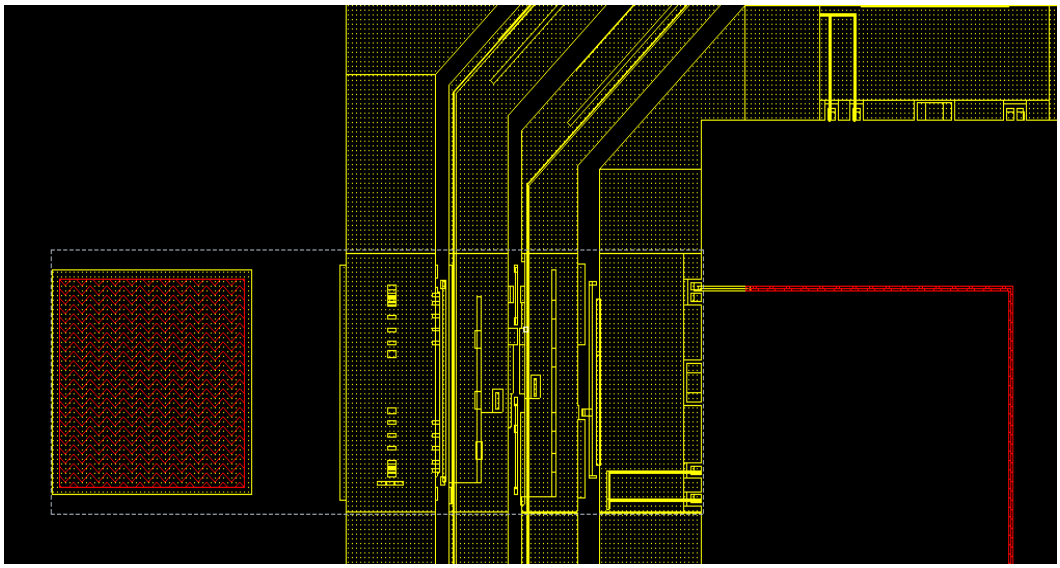

Figure 433: Enable tied low

Enable Tied high for data output:

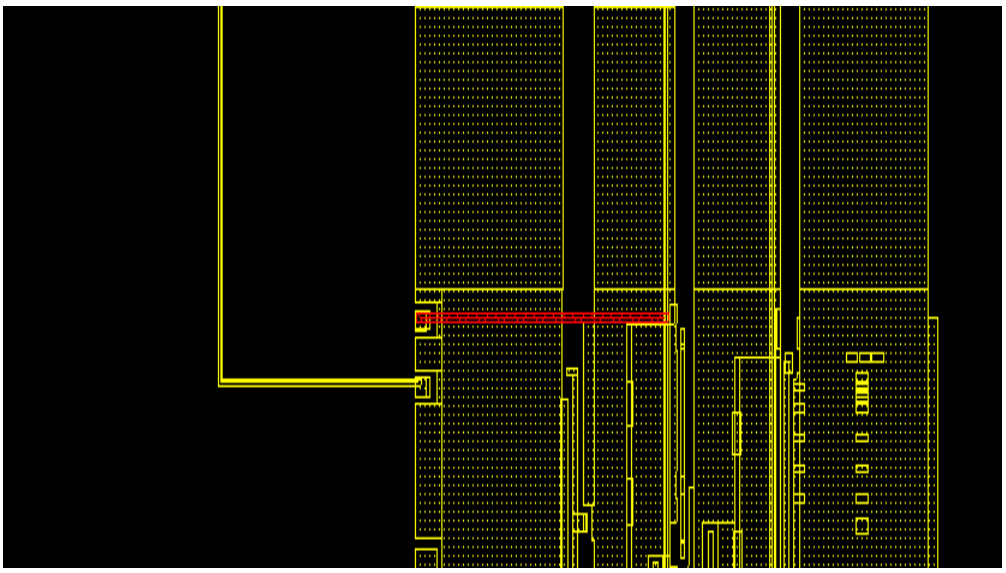

Figure 444: Enable tied high

12) Edit your chip Schematic:

In your chip level schematic view, you will see the 28 pin pad frame. You need to instantiate you core in the middle and connect to the pad frame. An instantiation window will come up and we can start to draw wires and assign names.You can make connections through labeling. This approach is less likely to introduce errors through making wrong connections.Instantiate gnd and vdd symbols from the UofU_Analog_Parts library. The connections in the schematic should match the layout for LVS later.

Save and check the design in the end.

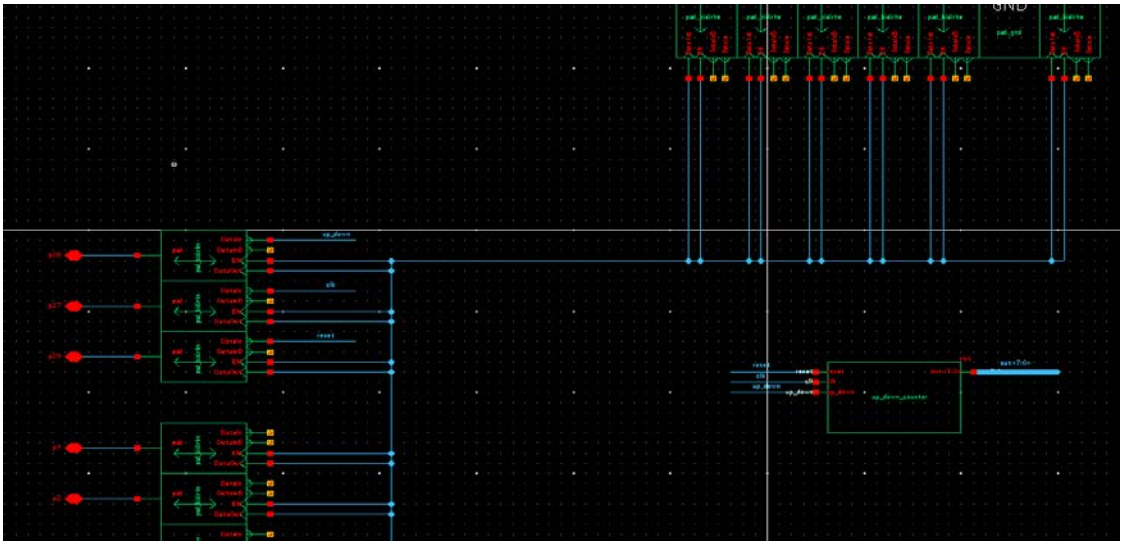

Figure 455: Final chip schematic

13) Run DRC and LVS in Cadence virtuoso using the appropriate rule files.

14) Export Final GDS to be fabricated:

To export design data in GSDII format:

a) From the Console, choose File > Export > Stream.

b) In Main, enter the run directory and GDS file in Stream file name.

c) Browse the library and cell that you want to export and leave the view as layout. Select the cell that includes the pad ring and core connected.

d) Choose the Options tab and specify option details. Choose the Map Files tab and specify map file details. The layer map should be different than the one used to read your core OA2GDS.layermap

e) Click OK.

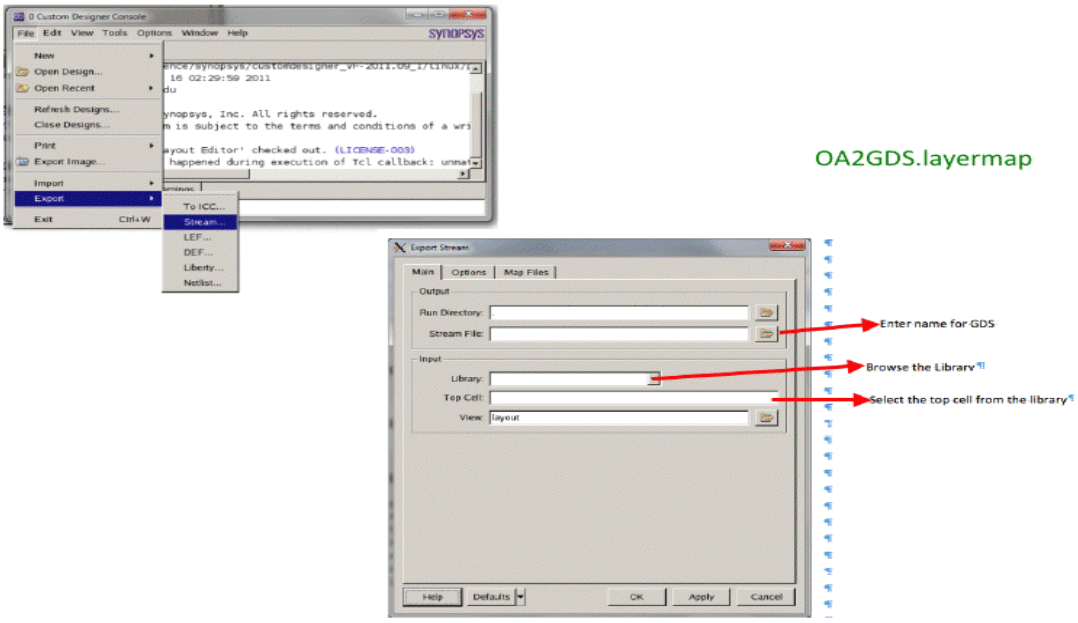

Figure 466: GDS Export

CHAPTER 7: CONCLUSION AND FUTURE WORK

A simple counter design was written in RTL. The RTL was then synthesized to get the gate level netlist. This netlist was then used to generate a core corresponding to the counter functionality. The final core generated was integrated with the 28 pin pad frame. The design GDS was then exported and sent to MOSIS for fabrication.

The thesis resulted in the development of a methodology for chip design for undergraduate students using the tools available at UTEP. Exposure to up to date tools used in the industry for chip design is provided. The final silicon produced could be a great asset for students looking for jobs and future research in the area of digital chip design.

The final silicon coming after fabrication from MOSIS needs to be tested to verify correct functionality. The future work associated with this thesis would to develop a methodology for low power or small area or faster design for the same design method. This will require modifying the commands used during synthesis and IC compiler phase of the design methodology.

The existing flow uses both Synopsys and Cadence. Exclusive flows need to be developed for both Synopsys and Cadence tools.

REFERENCES

[1] IC Compiler User Guide Version A-2007.12-SP2, March 2008. [Dec 2011]

[2] Himanshu Bhatnagar. "Advanced ASIC chip Synthesis Using Synopsys Design Compiler, Physical Compiler and Primetime" 2004. [Jan 2012]

[3]Design Compiler User Guide Version A-2007.12, December 2007 [Nov 2011]

[4] IC Compiler 1 Manual by Synopsys Version 2010.12-SP2 [Nov 2011]

[5] Narayanan, U, "Low power logic synthesis under a general delay model," Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium,pp.209-214,10-12 Aug 1998.

[6] Balasubramanian, P, "Low Power Synthesis of XOR-XNOR Intensive Combinational Logic," Electrical and Computer Engineering, 20007 Canadian Conference,pp.243-246,22-26 April 2007.

[7] Pedram, M.; Abdollahi, A, "Low-power RT-level synthesis techniques: a tutorial," Computers and Digital Techniques, IEE Proceedings -Volume: 152 , Issue: 3,pp.333-343,6 May 2005

[8] Hill, D.D.; Aranha, M.A.; Shugard, D.D, "Placement algorithms for CMOS cell synthesis" Computer Design: VLSI in Computers and Processors, 1990. ICCD '90. Proceedings., 1990 IEEE International Conference,pp.444-458,1990.

[9] Richard, B.D, "A Standard Cell Initial Placement Strategy," Design Automation, 1984. 21st Conference,pp.392 - 398,1984

[10] Hossain, M.; Thumma, B.; Ashtaputre, S, "A new faster algorithm for iterative placement improvement",VLSI, 1996. Proceedings., Sixth Great Lakes Symposium,pp.44-49,1996.

[11] Forzan, C.; Pandini, D, "Why we need statistical static timing analysis",Computer Design, 2007. ICCD 2007. 25th International Conference ,pp.91-96,2007.

[12] Tsukiyama, S.; Tanaka, M.; Fukui, M, "Techniques to remove false paths in statistical static timing analysis",ASIC, 2001. Proceedings. 4th International Conference,pp.39-44,2001.

[13] Gattiker, A.; Nassif, S.; Dinakar, R.; Long, C, " Timing yield estimation from static timing analysis",Quality Electronic Design, 2001 International Symposium,pp.437-442,2001.

[14] Ravi, S.; Raghunathan, A.; Chakradhar, S, " Efficient RTL power estimation for large designs ",VLSI Design, 2003. Proceedings. 16th International Conference,pp.431-439,2003.

[15] Mathur, A.; Qi Wang, "Power Reduction Techniques and Flows at RTL and System Level ",VLSI Design, 2009 22nd International Conference,pp.28-29,2009.

[16] Tsutomu Sasao, "Switching Theory for Logic Synthesis", Kluwer Academic Publishers,April 1999.

[17] H. Rahaman, et al., "Testing of stuck-open faults in generalized Reed-Muller and EXOR sum of-products CMOS circuits," IEE Proc. on CDT, vol.151, no.1, January 2004, pp.83-91.

[18] Anantha P. Chandrakasan and Robert W. Broderson. Low Power Dzgztal CMOS Deszgn. Kluwer Academic Publishers,1995.

[19] Burkis, J. ,"Clock Tree Synthesis for High Performance ASICS", ASIC Conference and Exhibit, 1991. Proceedings., Fourth Annual IEEE International, 23-27 Sep 1991

APPENDIXAVERILOG CODE

```
//-------------------------------------------------

// Design Name :up_down_counter

// File Name   : up_down_counter.v

// Function    : Up down counter

//-------------------------------------------------

moduleup_down_counter    (

out      ,  // Output of the counter

up_down  ,  // up_down control for counter

clk      ,  // clock input

reset       // reset input

);

//----------Output Ports--------------

output [7:0] out;

//------------Input Ports--------------

inputup_down, clk, reset;
```

```
//------------Internal Variables--------

reg [7:0] out;

//-------------Code Starts Here-------

always @(posedgeclk)

if (reset) begin // active high reset

out<= 8'b0 ;

end else if (up_down) begin

out<= out + 1;

end else begin

out<= out - 1;

end



endmodule
```

APPENDIX B SYNTHESIS TCL SCRIPT

```
setlib_path  "/users/eesunz/faculty/cdsemac/EE5375/UTFSM_libraries"
setsyn_path  "/export/cadence/synopsys/Synthesis/libraries/syn"
setsearch_path [list "$lib_path/MW_UTAH" "$syn_path"]

settarget_library [list UofU_Digital_v1_2.db]

setsynthetic_library [list dw_foundation.sldbstandard.sldb]

setlink_library [concat $target_library $synthetic_library]


# These cells have two outputs which causes a DRC error in ICC
set_dont_use UofU_Digital_v1_2/DCBX1
set_dont_use UofU_Digital_v1_2/DCBNX1

define_design_lib work -path ./work


################################
#  Change this to your file(s)
################################
analyze -format verilog ./up_down_counter.v

################################
#  Change this to your module name
################################
elaborateup_down_counter

################################
#  Change this to match your clocks and timing constriaints
################################
set_max_delay 25 -to [all_outputs]
create_clock "clk" -period 100
#create_clock "clk_in2" -period 100
#create_clock "clk_in3" -period 100

################################
# if you have an internally generated clock instantiate a buffer (BUFX2)
# to provide a pin to identify the source of your new clock
################################
create_generated_clock -divide_by 1024 -source clk [get_pins {buffer/Y}]
```

```
report_clocks

check_design>check_design.output

ungroup -flatten -all

set_flatten true -effort high
uniquify

compile_ultra

################################
# Check these files once completed
################################
report_area>area.rpt
report_hierarchy>hierarchy.rpt
report_constraints>constraints.rpt
report_timing>timing.rpt

set_propagated_clock [all_clocks]

################################
# Your timing file for IC Compiler
################################
write_sdcup_down_counter.sdc

################################
# Your output netlist
################################
write -f verilogup_down_counter -output up_down_counter.post_synth.v -hierarchy

exit
```

APPENDIX C PLACE AND ROUTE TCL SCRIPT

```
    ##########################################################################
    # ON C5 0.5u CMOS ASIC Place and Route Script using IC Compiler - Version 3.0
    ##########################################################################
    # This script assumes that the post synthesis netlist and sdc file produced by
    #   synthesis are both in the local directory
    ##########################################################################
    # Update the lib_path and design_name for your design
    ##########################################################################
    setlib_path    "/users/eesunz/faculty/cdsemac/UofU_SYNS_v1_2/UTFSM_libraries"
    setdesign_name    "up_down_counter"

    set_app_varsearch_path "$lib_path/MW_UTAH"
    set_app_vartarget_library "UofU_Digital_v1_2.db"
    set_app_varlink_library "* $target_library"

    ######change design here
    #if { ( file exists "[set design_name]_LIB" ) } { shrm -r "[set design_name]_LIB" }
    shrm -r ${design_name}_LIB
```

```
######change design here
create_mw_lib  -tech  "$lib_path/MW_UTAH/UofU_Digital_MW.tf"  -mw_reference_library "$lib_path/MW_UTAH/UofU_Digital_MW"  ${design_name}_LIB

######change design here
open_mw_lib  "${design_name}_LIB"

set_tlu_plus_files  \
   -max_tluplus  "$lib_path/MW_UTAH/ami500.tluplus"  \
   -min_tluplus  "$lib_path/MW_UTAH/ami500.tluplus"  \
   -tech2itf_map "$lib_path/MW_UTAH/ami500hxkx_3m.map"

import_design  "./${design_name}.post_synth.v"  -format "verilog" -top ${design_name} -cel ${design_name}

read_sdc ./${design_name}.sdc

###### Adjust density here to alleviate LVS errors after routing at the expense of a larger design
create_floorplan -control_type "aspect_ratio" -core_aspect_ratio "1" -core_utilization "0.4" -row_core_ratio "1" -start_first_row  -left_io2core 24 -bottom_io2core 27 -right_io2core 24 -top_io2core 27
```

derive_pg_connection -power_net {vdd!}-ground_net {gnd!}

create_rectilinear_rings -nets {vdd! gnd!} -offset {3 3} -width {4.5 4.5} -space {3 3}

Adjust the number of straps here to alleviate LVS problems after routing - at the expense of a less robust power network

create_power_straps -direction vertical -num_placement_strap 1 -start_at 400 -increment_x_or_y 200 -nets {vdd! gnd!} -width 1.800 -layer metal3

place_opt -effort high -congestion

preroute_standard_cells -nets {vdd! gnd!} -connect horizontal -extend_to_boundaries_and_generate_pins

clock_opt -fix_hold_all_clocks

report_clock_tree

report_timing

route_zrt_auto -max_detail_route_iterations 1000

route_zrt_detail -incremental true

insert_stdcell_filler -cell_without_metal "FILL8 FILL4 FILL2 FILL" -connect_to_power "vdd!" -connect_to_ground "gnd!"

preroute_standard_cells -nets {vdd! gnd!} -connect horizontal -extend_to_boundaries_and_generate_pins

derive_pg_connection -power_net {vdd!}-ground_net {gnd!}

This command shows a warning that the command is old and no longer valid.

Not really true. For designs in deep submicron (65nm and below) you should use the new checker.

However for designs older than 65nm, this is the appropriate checker.

verify_drc

verify_lvs

route_zrt_detail -incremental true

report_timing> ${design_name}.timing

change_names -rules verilog -hierarchy

write_verilog ${design_name}.pnr.v

```
shgrep -v FILL ${design_name}.pnr.v> ${design_name}.nofill.v


set_write_stream_options -output_pin {text geometry} -keep_data_type
write_stream -lib_name ${design_name}_LIB -format gds ${design_name}.gds

write_sdc     ${design_name}.pnr.sdc

extract_rc -coupling_cap

write_parasitics -format SBPF -output "${design_name}.pnr.sbpf"

verify_pg_nets

report_timing

report_clock_tree

save_mw_cel

##close_mw_cel

##exit
```

VITA

Arun Joseph Kurian was born on January 28, 1987 in Kerala, India. The first born son of Mr. Joseph Kurian and Geetha Joseph, he graduated from Mumbai University in India in 2005. He entered the University Of Texas at El Paso (UTEP) in Fall 2010 to pursue a Master of Science Degree in Computer Engineering. In the spring of 2012, he went for an internship at Intel Folsom California and worked with the Design Automation team for their Nand Solutions Group. Upon Graduation he will be joining Intel Corporation, in Folsom, California as a fulltime.

Permanent address: 1532 Upson Drive

El Paso, TX 79902