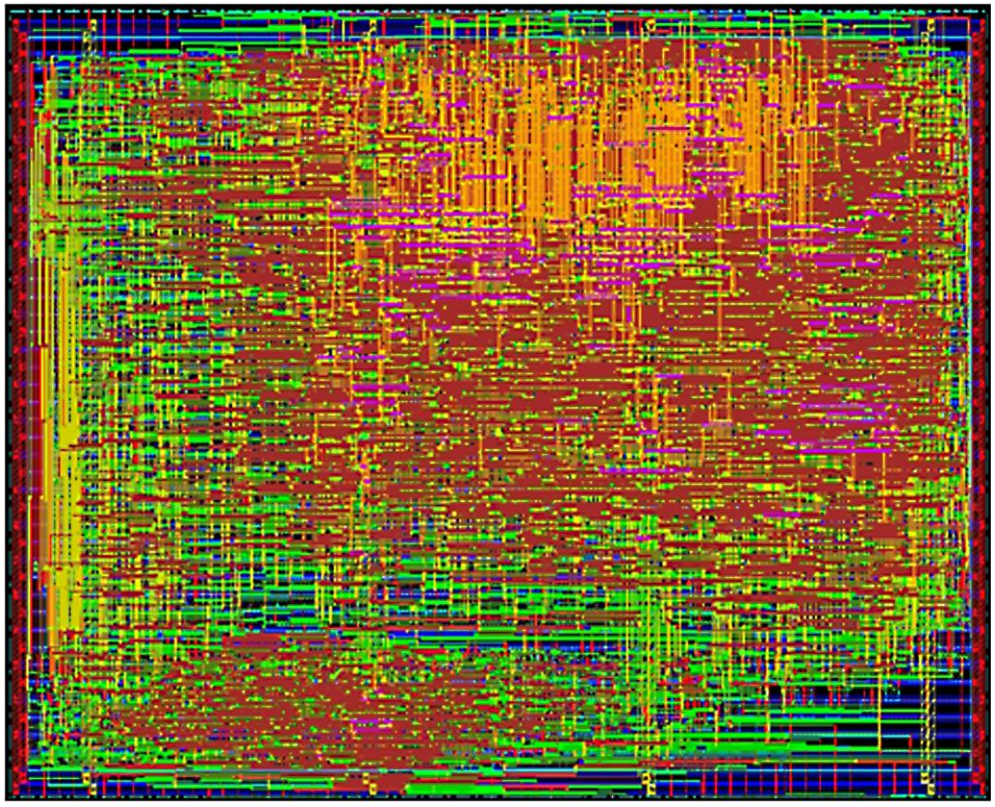
CHALMERS



Impact of Pin Orientation on Routing Regularity of HPM Architectures

Master of Science Thesis in Integrated Electronic System Design

AFFAQ QAMAR

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

Impact of Pin Orientation on Routing Regularity of HPM Architectures

Affaq Qamar

Examiner: Prof. Per Larsson-Edefors

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover:
A fully routed 32-Bit HPM Multiplier with a rectangular PPRT]

Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

Abstract

In the context of regular arithmetic circuits, the effect of pin placement on the quality of layout and routing is not well understood. Current methodologies depend on library-based flows to design such circuits. However, the benefits of regularity are lost in the process of automated place and route techniques employed by these methodologies. As process technologies grow smaller, this will have a large effect on the yield and variability. Enforcing regularity to combat variability is being advocated in the form of restricted design rules. This thesis attempts to develop a methodology to implement customized pin orientations for the cells. These cells are used in the design to harness the benefits of regularity and in the process, mitigate variability. HPM multiplier is taken as a case study and different pin orientations are tried out for the cells constituting rectangular PPRT of the multiplier.

The tool-set to be used for this project include Cadence Virtuoso for implementing the standard cell layouts, Cadence Encounter Library Characterizer to perform characterization of the implemented layouts and Cadence SoC Encounter to implement the HPM multiplier using the customized standard cells.

— Life is pretty simple: You do some stuff. Most fails. Some works. You do more of what works. If it works big, others quickly copy it. Then you do something else. The trick is doing something else.

Leonardo da Vinci

— Whatever you do in life will be insignificant, but it's very important that you do it. Because nobody else will.

Tyler (Remember Me)

Acknowledgments

Below follows list of people who helped me to complete this thesis.

- Professor Per Larsson-Edefors has been an excellent supervisor. He provided tremendous support, guidance, and motivation throughout this project. Thanks for always having time to answer questions and replying to emails almost instantaneously. It has been a rewarding and unique experience to work and study under your supervision!
- Kasyab, many thanks to you for your tremendous help in technical issues and sharing your knowledge about EDA tools. You always helped me to solve 'freaky' problems associated with the tools.
- Azhar and Fahad, thanks to you guys for opposing the thesis and listening about my work although I knew you don't understand what I was telling.
- Lars Kollberg and Lennart Hansson for tool support.
- Very special thanks to all of the fellow students and the faculty members who made IESD program an interesting yet an enjoyable course. I will treasure the friendships that I have taken from here.

Contents

List of Figures

List of Tables

Abbreviations

HPM	**H**igh **P**erformance **M**ultiplier
PPG	**P**artial **P**roduct **G**eneration
PPRT	**P**artial **P**roduct **R**eduction **T**ree
EDA	**E**lectronic **D**esign **A**utomation
HA	**H**alf **A**dder
FA	**F**ull **A**dder
ELC	**E**ncounter **L**ibrary **C**haracterizer
LEF	**L**ibrary **E**xchange **F**ormat
LIB	**LIB**erty library format
DEF	**D**esign **E**xchange **F**ormat
PLS	**P**ost **L**ayout **S**imulation
SPICE	**S**imulation **P**rogram with **I**ntegrated **C**ircuit **E**mphasis
DRC	**D**esign **R**ule **C**heck
LVS	**L**ayout **V**ersus **S**chematic
GDSII	**G**raphics **D**esign **S**tation **II**
PR	**P**lace -and- **R**oute
RTL	**R**egister **T**ransfer **L**ogic
TDM	**T**hree **D**imensional **M**ultiplier
SOC	**S**ystem **O**n **C**hip

For my parents. Their patience and love knows no bounds.

1

Introduction

1.1 Column Compression Multipliers

Integer multiplication is an important and commonly occurring operation in logic circuits of medium to high complexity. The alternatives to designing the parallel multipliers may be classified into array multipliers, routinely presented in text books, or logarithmic-depth multipliers, such as those proposed by Dadda [1] and Wallace [2]. Another approach which relies on finding a globally optimal interconnection pattern, determined heuristically for the reduction network, is called the Three Dimensional Method [3]. These architectures all consist of three basic structures to achieve multiplication:

- Partial Product Generator (PPG),
- Partial Product Reduction Tree (PPRT) and
- Final adder, to compute the product from the reduction tree.

The common goal in all these architectures is to reduce the delay of the PPRT, since it is the critical part which determines the overall performance of the respective architecture.

However, a shortcoming common to all of these approaches is the lack of regularity in terms of layout and routing, leading to high-effort custom design. One proposed variant of the Dadda architecture, the HPM architecture, has a regular layout while retaining all the performance benefits of Dadda architectures. Regularity of layout is an extremely desirable trait, if it delivers the high performance of existing algorithms at the lowest possible design effort. Previous work [4] has shown, that the HPM architecture can match the performance of a Dadda implementation and perform better than a Wallace implementation. Another project [5] implemented regular PPRT structures using standard-cell design techniques. The evaluation of results from that project indicated that the routing algorithms employed by the EDA tools were unable to harness the possibility of regular routing; consequently the performance suffered. One of the possible reasons for this is that the foundry-provided cells lack the proper pin orientations when their layout orientation is flipped, to remain consistent with the abutted placement philosophy. In order to be able to exploit the benefits of a regular layout completely, proper pin orientations then become necessary [6].

1.2 Motivation and Scope of Thesis

Various standard cell libraries like the one provided by STMicroelectronics, provide different variants of cells in terms of the drive strength but they lack such utility when it comes to having same cells with different pin orientations. Thus when the cells with fixed pin orientations are incorporated in the design, the designers have to rely on routing tools without providing any assistance to ease routing. The placement tool flips the cells horizontally in every alternating row in order to share common supply rails (vdd and gnd) thereby changing the pin order. This idea is well depicted in Fig. 1.1, where 32-bit rectangular PPRT of HPM architecture is taken as an example to demonstrate the flipping action.

It is evident from the Fig. 1.1 that the abutted placement philosophy results in pin order to change for every alternating row, which is an undesired trait. The HPM algorithm displays a regular layout with a possibility of regular routing structure [4] and we can attain more benefits out of this routing regularity by trying out different pin orientations. An added benefit of having flexibility in terms of pin orientation is, that it may reults in shorter wires and fewer number of vias, which in turn result in lower variability. These starting clues thus provided the primary motivation of this work.

As predicted in [5], by customizing the cell's pin placement, we might able to achieve more routing regularity. But in order to validate this hypothesis, it is necessary to have cell

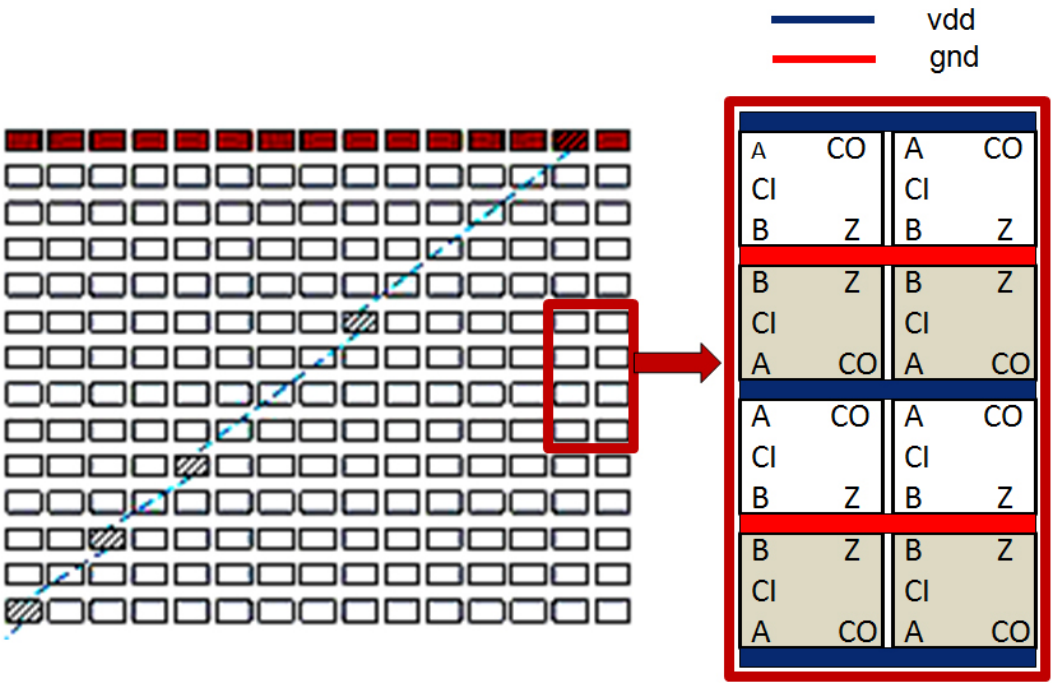


FIGURE 1.1: 32-bit rectangular PPRT depecting flipping

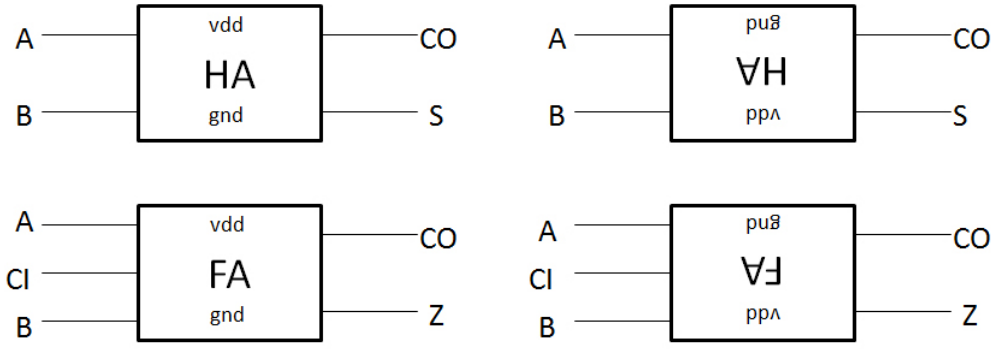
library that support such cells. Till now, we have not come across any standard cell libraries supporting such idea. This necessitated the goal of creating custom cell library having cells needed to form PPRT i.e. half adder (HA) and full adder (FA) cells. This custom cell library is so designed to have cells with different pin orientations.

Since custom cell design with different pin orientations is still in its infancy, it is very important to develop a methodology for carrying out this investigation. A reliable and stable methodology shall enable researchers to take this concept further beyond and apply it to other arithmetic circuits.

As a candidate to test the methodology, the HPM multiplier was chosen, since present research is the continuation of the research conducted by Subramaniyan [5]. This facilitated the reuse of scripts developed in [5] while helping to keep focus on new design challenges. Furthermore, rectangular PPRT was used as a design case in order to see the impact of pin placement since long wires lengths in rectangular PPRT enables to observe the gain in terms of routing regularity.

In order to complement the fast layout exploration flow described in [7], two test cases were designed, each using 32-bit HPM multiplier with rectangular PPRT. In the first case, a single type of custom cells was used, as depicted in Fig. 1.2(a). This forms a reference case, since we cannot directly compare the results with that of optimized, foundry-characterized cells.

In the second case, we use flipped cells as shown in Fig. 1.2(b). These are designed to be the flipped copy of normal cells, replacing them at the flipped instances in the PPRT thus preserving the pin positions in all cell rows as shown in Fig. 1.3.



(a) Normal cells (b) Flipped cells

FIGURE 1.2: Design approach for custom cells

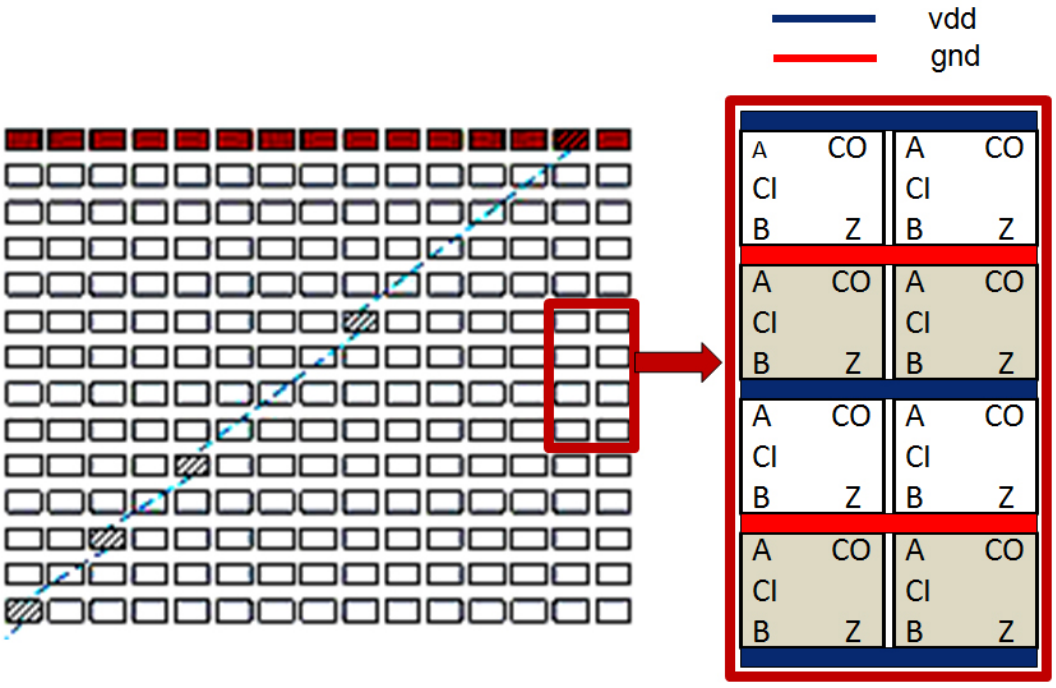


FIGURE 1.3: 32-bit rectangular PPRT with preserved pin positions in all cell rows

Summarizing: this thesis has a focus on the development of a design methodology to enable layout exploration of full custom cells with different pin orientations. As a candidate to test the methodology, the HPM multiplier was chosen due to its inherent benefits from a regularity of layout standpoint and reusability of the previous work carried out in department. Further, a custom cell library consisting of cells needed for PPRT was used to test the hypothesis that the pin orientation of the cells does impact the routing regularity.

1.3 Organization

Any well developed methodology has clarity of flow. To highlight all the important aspects of the newly developed methodology, the research was carried out in two distinct steps: forming custom cell library and layout exploration. These, then form the first two parts of the document. The implementations carried out as a result of this methodology are addressed in the third part of the thesis as a case study.

Part I deals with the design of Full Custom Library. Chapter 2 in this part describes the requirements and introduces the tool flow. Chapter 3 introduces layout of custom cell design. Chapter 4 deals with the abstract generation in order to capture geometry information to the custom cells. Chapter 5 discusses custom library characterization used to capture timing information of the custom cells. The output from the abstraction generation along with library characterization thus forms full custom cell library.

Part II deals with the details of Layout Exploration. Chapter 6 deals with the integrated flow to implement a complete design. It also discusses Wired as a passing topic as it was not used in thesis directly but the outputs produced from Wired in previous work [5] was rendered. These together constitute the methodology from a layout exploration standpoint. Part III of this document deals with the implementations carried out during the course of this thesis. Section 7.1 of Chapter 7 describes the various implementations while section 7.2 reports the results from those implementations. Section 7.3 is a discussion of the results and how this work may be further developed.

Part I
Custom Cell Library

2

Tool Flow Methodology

Full custom design flows have been used to achieve the highest performance. The designers can alter any design parameters down to the transistor level to increase accuracy of design. However, this accuracy comes at the cost of design effort and time. The Non-Recurring Engineering (NRE) costs for a full custom design is high and the scope of reuse limited. This is true for any circuit designed using full custom techniques. With shrinking process technologies, there has been an up rise in complexity of design and with growing design rule checks, full custom techniques are now limited to the design of the most performance critical circuits. In the case of parallel column compression multipliers this has proved vital, as multiplication operation is known to occur in most of the designs of medium to high complexity and a circuit employing such a multiplier will have a critical path through it more often than not. However, given the effort required to successfully create a high performance column compression multiplier in a custom design flow, even the slow array multiplier may be considered, if performance requirements are low [5].

2.1 Requirements

An effort into developing a methodology to implement column compression multipliers; it was considered from the very beginning that such a methodology must be capable of having flexibility, so that it can be extended to the implementation of other arithmetic circuits such as shifters and multiplexers etc. Given the primary requirement of design exploration in the context of custom pin placement, the design flow may include:

- Compatibility with existing design flows. This is a requirement because the means to achieving the end goal should not come at the cost of a steep learning curve.
- Developing a full custom library having same logical cells with different pin orientations.
- Compatibility of the custom cell library to that of the standard cell libraries so that multiplier as a whole can be implemented by following a semi-custom approach where performance critical part i.e. PPRT incorporates cells from the custom cell library, while rest of the implementation uses foundry provided cells.

Since the effort was directed at nanometer geometries, an effort was made to build upon previous related work [5]. The primary need, in view of the existing research, became an effort to identifying key aspects of implementing a portion of complete multiplier (PPRT) using a custom cell library approach. Fig. 2.1 describes the process description of the developed methodology. The layout design along with verification and SPICE-netlist generation is discussed in custom cell design chapter, while chapter 4 and chapter 5 focuses on forming custom cell library. The frontend-backend integration step is covered in part II of the thesis.

Given the fact that this work was aimed at nanometer regime designs, implementations were set to be carried out using the 90nm process technology for laying out custom cells while having standard cell libraries from STMicroelectronics for rest of the blocks.

2.2 Introduction to Tool Flow

The following tools are used to implement the methodology:

1. Cadence Virtuoso Custom Design Platform (IC5.10.41) for constructing cell layouts.

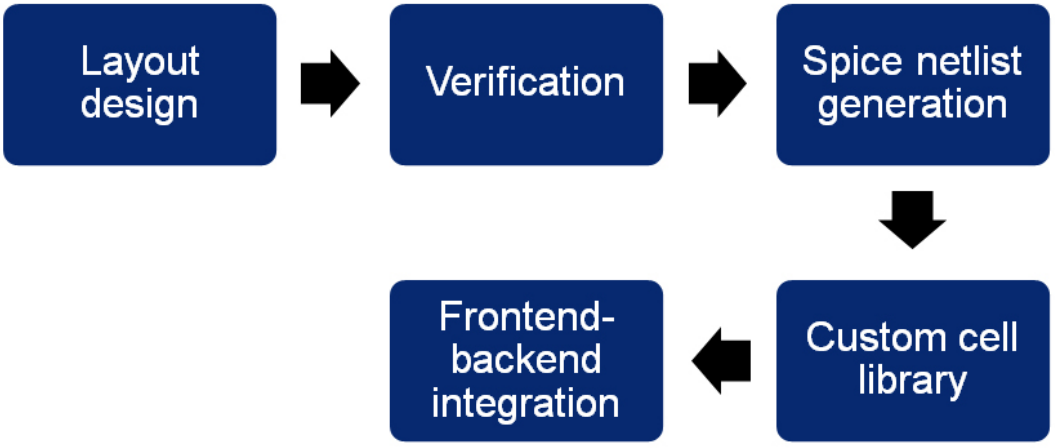


FIGURE 2.1: Methodology

2. Calibre Interactive by Mentor Graphics for DRC and LVS check.
3. Virtuoso Abstract Generator by Cadence for capturing geometric information of custom cells.
4. Encounter Library Characterizer by Cadence for capturing timing information of custom cells.
5. Cadence Encounter for physical synthesis.

The following chapters discuss how the tools were applied.

3

Custom Cell Design

The necessity of creating a custom cell library arises from the fact that there are not many standard cell libraries supporting the idea of having cells with same logic but different pin orientations. It must be noted here that we only aim to have cells needed to form PPRT of a multiplier, since it is the critical part of HPM. So the custom cell library we talk about, contains two versions of half adder and full adder cells each, with different pin orientations as depicted by Fig. 3.1.

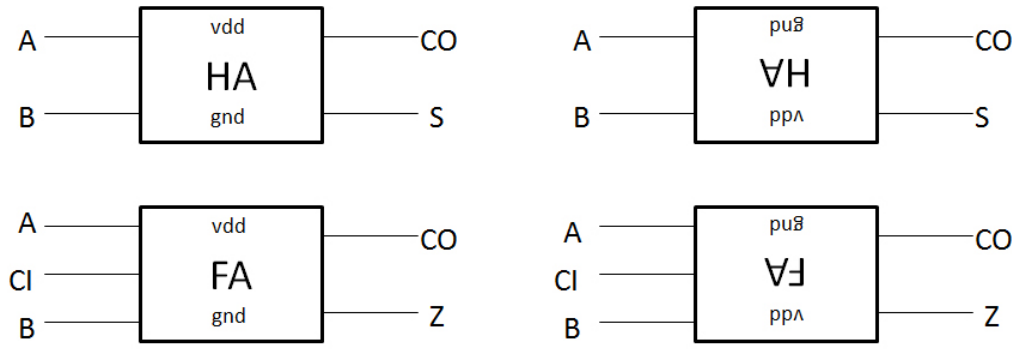


FIGURE 3.1: Symbolic representation of different versions of half adder and full adder cells used in the flow

It is evident from the Fig. 3.1 that each version of cell is logically same as their respective cell and the only difference is the pin order, so arranged to have order symmetry when used in a design that follows abutted placement. While writing the placement aware description of the PPRT of the multiplier, we make sure that the tool picks the respective cell while following abutted placement.

In the next sections, we shall discuss the logic functionality of the cells and process of layout and SPICE netlist generation of each cell.

3.1 Layout Design

CMOS technology is highly used by design community due to its high noise immunity and low static power consumption [8]. In this project, the custom layout of adder cells are implemented in CMOS technology in Cadence Virtuoso using 90-nm process from ST.

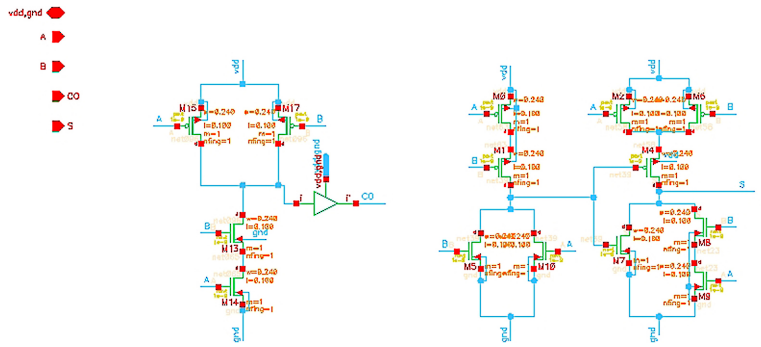
As mentioned in section 1.2, we are going to use same cells in both of the design cases and make comparison between them. This makes the design choice of constructing adder cells much easier, as now, we can use any of the design without focusing much on optimization.

3.1.1 Half Adder

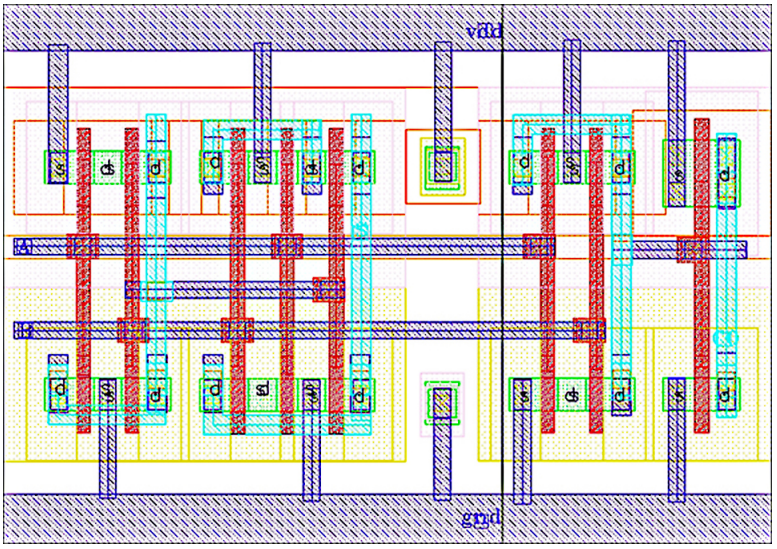
A half adder is a logical circuit that performs an addition operation on two one-bit binary numbers often written as A and B. The half adder output is a carry and a sum of the two inputs usually represented with the signals CO and S, respectively. Following is the logic description for a half adder:

$CO = A.B$
$S = A \oplus B$
The sum equation can be extended to have the following;
$S = (A'B + AB')''$
$S = (AB + A'B')'$
$S = (AB + (A + B)')'$
This can be implemented by using an OR gate along with AOI12 gate as depicted in Fig. 3.2(a).

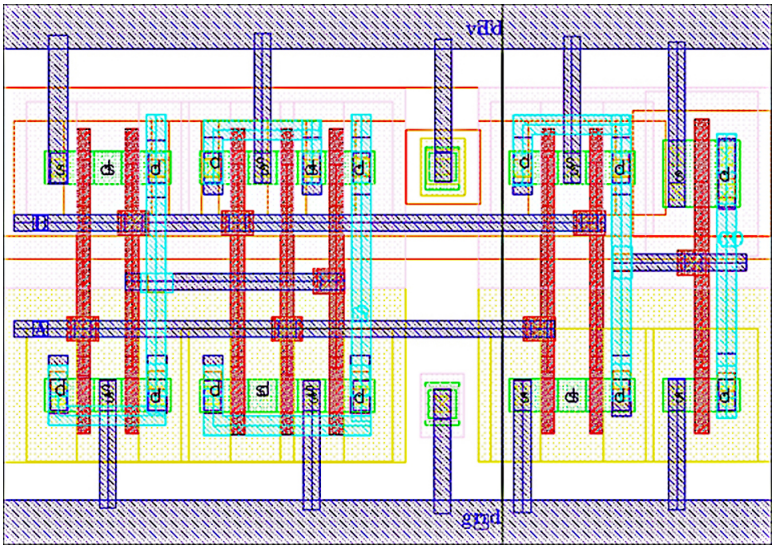
The corresponding layouts of the half adder cell are shown in Fig. 3.2(b) and Fig. 3.2(c). The circuit consists of 16 total transistors (10 transistors used for construction of sum function while 6 transistors for carry). For the first case, named ”Normal Cells” in section 1.2 of the



(a) Half Adder schematic



(b) Layout of half adder cell used in normal case



(c) Layout of half adder cell used in flipped case

FIGURE 3.2: Half Adder cells

report, only the cell in Fig. 3.2(b) is used. The second case, named "Flipped Cells" used both cells shown in Fig. 3.2(b) and Fig. 3.2(c). These cells are logically same having the aspect ratio (L/H) of 1.339 with only difference of inverted pin order.

3.1.2 Full Adder

A one-bit full adder is a logic circuit with three single bit binary inputs (A, B, CI) and two single bit binary outputs (CO, Z), where CO represent carry and Z represents sum output of the full adder. Of the three inputs, CI is the fast arriving signal in multiplier implementation while A and B are two late arriving signals. The basic logic relationship of the full adder can be described as:

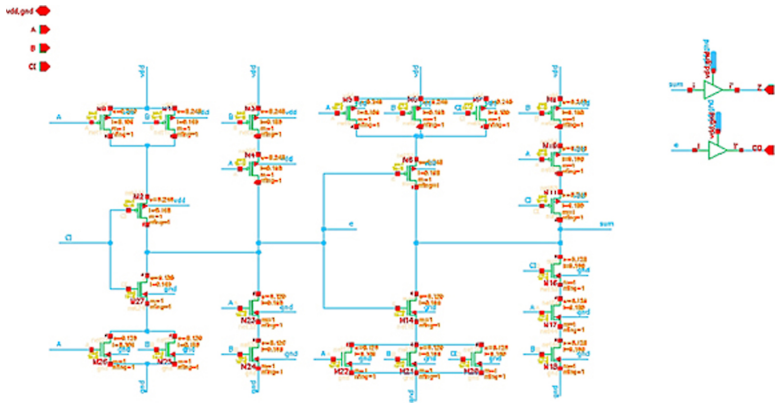
$CO = (A.B) + (B.CI) + (CI.A)$
$Z = A \oplus B \oplus CI$
In the present research, the full adder is implemented as a Mirror Full Adder. It optimizes the output signals and consists of 28 transistors totally (4 transistors used for the construction of two inverters), as shown in Fig. 3.3(a). The design symmetry of the mirror adder makes the layout design much easier. Fig. 3.3(b) and Fig. 3.3(c) shows the layouts of full adder cell with two different pin orders.

The aspect ratio (L/H) of both the layouts is 2.417. For the "Normal Cells" design case, the cell in Fig. 3.3(b) is used, while for the "Flipped Cells" design case, both the cells shown in Fig. 3.3(b) and Fig. 3.3(c) are used.

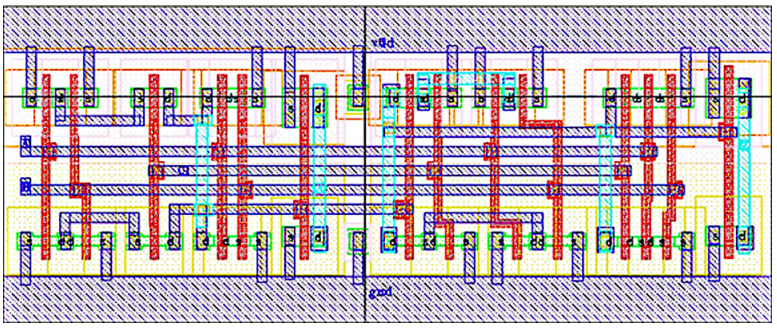
3.1.3 Design Constraints

Looking at Fig. 3.2 and Fig. 3.3, it can be seen that the cell layouts are not optimized for area. It is entirely due to the fact that we are focusing on impact of pin placement and following relative comparison of design cases by employing same kind of cells, so presently, the area optimization is not a major concern. As we follow a semi-custom design approach for the multiplier design as a whole, where full custom cells are being used along with the foundry provided cells, so we have following design constraints to harmonize integration of custom library cells.

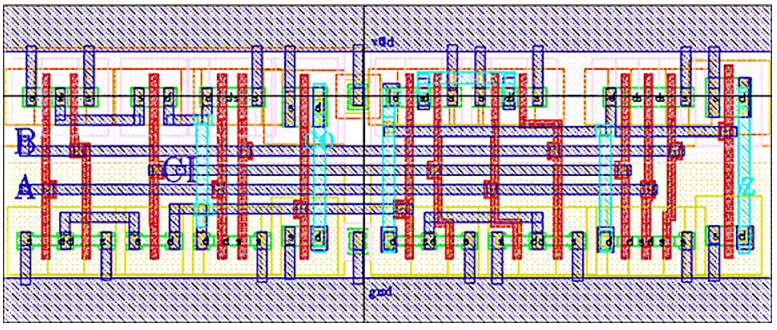
- The naming convention is kept consistent to standard cell libraries.
- Cell pins, with the exception of abutment pins (vdd and gnd) must be placed on the intersections of the vertical and horizontal routing grids.



(a) Full adder schematic



(b) Layout of full adder cell used in normal case



(c) Layout of full adder cell used in flipped case

FIGURE 3.3: Full Adder cells

- The widths and heights of the cells are kept multiple of the metal 1 and metal 2 pitches, respectively.
- The heights of all the cells are made equal to the standard row height (3.92 m) in order to make the P&R job easier.

3.2 Verification

A successful Design Rule Check (DRC) ensures that the layout conforms to the rules designed/required for faultless fabrication. However, it does not guarantee if it really represents the circuit you wish to fabricate. This is where a Layout Versus Schematic (LVS) check is used.

There are several vendors that provide tools to perform these checks. The licensing issues hindered the usage of other tools, hence Calibre Interactive by Mentor Graphics was used to perform DRC and LVS. The detailed procedure of setting up the tool and using it can be found in [9].

3.3 SPICE Netlist Generation

Once all the violations in the layouts are fixed, we are ready to perform parasitic extraction. The Post-Layout Simulation (PLS) tool-kit by STMicroelectronics is used to automate interconnect RC parasitics extraction from a GDSII database and a CDL netlist to a SPICE netlist at device (transistor) level [10]. The PLS graphical interface manages all the steps starting by the LVS verification of the design, the parasitic extraction, up to the conversion and packaging of the extracted netlist to netlists that can be simulated by the supported simulators.

The PLS tool can be invoked either from the Cadence Layout or Schematic editor. In the present research, we used PLS flow based on textual netlist import. To perform LVS verification, Calibre from Mentor is used while the interconnect parasitic extractor tool supported is Star-RCXT provided by Synopsys. The target simulator used in PLS flow was Spectre.

After having a successful PLS flow, we get a set of textual cellviews that are simulable extracted netlists, and a set of graphical cellviews that are usable as stopping views in the Analog Artist Simulation Environment. The SPICE netlist generated is used in library characterization process. The detailed process of characterization is discussed in Chapter 5.

4

Abstract Generation

Place and route tools do not require the full cell layout. They however, need geometric information about the cells. Another view of the standard cells called the abstract view contains all such information and needs to be generated. Abstract generator is a library modeling tool that lets user create abstracts for standard cells, macro blocks, and IOs from detailed layout information in Library Exchange Format (LEF), Design Exchange Format (DEF), and Graphics Design Station II (GDSII) Stream formats [11].

4.1 Goals of Abstract Generation

The abstract view provides information like:

- Cell name, site name, cell orientation.
- Cell PR boundary.
- Pin names, locations, pin metal layer, type and direction (input/output/input-output).
- Provides location of all metal track and vias in the layout (obstructions).

This information is passed to the P&R tool in the LEF format (Library Exchange Format). The LEF file contains all the cell descriptions and may contain technology information if needed. Fig. 4.1 below shows the abstract view of a simple inverter as an example.

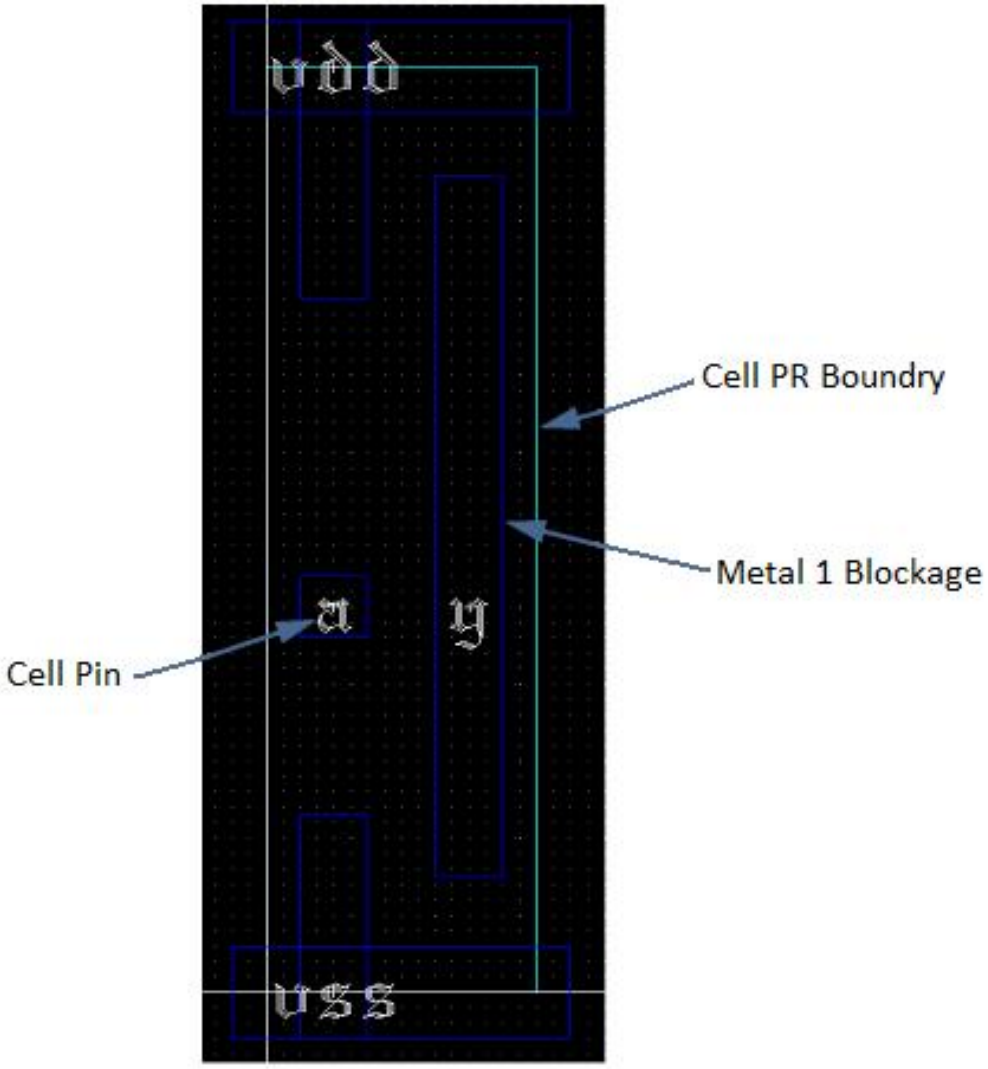


FIGURE 4.1: Abstract view of an Inverter

It is evident from the figure that abstract of a cell layout contains information only about metal and via layers. The abstracts are used in place of full layouts to improve the performance of P&R tools. After the P&R is complete, the abstracts are replaced back with the layouts.

4.2 Abstract Generator Environment

In order to generate abstract view of the cells, we need to import the GDS file containing our standard cell layouts. This can be easily achieved by first exporting the standard cell library to stream (GDS) format and then re-importing the GDS file into abstract generator.

By default five bins are available in the abstract generator to place cells with different processing requirements. These are core, IO, corner, block, and ignore bins. Each cell in a library is always contained in exactly one bin. All standard cells go under the Core bin. Since we did not construct any other type of cells; except for the one needed to form PPRT of the multiplier; so we confined ourselves to use only the core bin in present research.

There are three main steps and one optional step in generating abstracts.

- Generating the pins view.
- The extract view.
- The abstract view.
- Finally the verify step which is optional.

The resulting abstract view contains only the net and the blockage information. A LEF file is then generated, using the Abstract view of the standard cells. The steps mentioned above are discussed briefly in the following sub-sections.

4.2.1 Pins Step

While exporting standard cell layouts to the GDS file, all the information regarding pins is lost. So we need to re-instate that information before proceeding. The abstract generator derives pins from the text labels in the layout view and places the locations at the text origins. The Pins step maps text labels to the metal layers, designating certain metal blocks as pins.

4.2.2 Extract Step

The Extract step merges metal blocks under the same net into one single net. It also traces the connectivity between shapes and terminals, starting at the pin purpose shapes created in the Pins step. It also changes any *metal.pin* layer into *metal.net*. The abstract generator performs the following functions during the Extract step:

- Extracts each terminal net, one shape at a time.
- Constructs the correct database model for strong, weak, and must connect pins.
- Creates library process antenna information for standard cells.

4.2.3 Abstract Step

The Abstract step copies the pin (net) information from the Extract step, and generate blockages for the metal and via layers (or any other layer that has been specified). These blockages tell the P&R tool (namely SoC Encounter); which parts of the standard cell to avoid routing over with certain layers. The abstract generator performs the following functions during the Abstract step:

- Adjusts pin shapes.
- Performs blockage modeling.
- Creates sites. The site tells the placer where it can place the cells and all cells within the same bin are associated with the same site.
- Calculates overlap layers.
- Calculates and analyzes grid.

4.2.4 Verify Step

This is a not a mandatory step in abstract generation process and we can export the abstract in LEF format without performing this step from the *File* menu. The abstract generator performs the following functions during the Verify step:

- Checks terminals for any differences between logical and abstract views.
- Checks any pins and geometries off the manufacturing grid.
- Creates a small test design for each abstract and verifies whether it can be routed using either Silicon Ensemble or Cadence Encounter.
- Verifies the geometry of each abstract in either Silicon Ensemble or Encounter.

5

Custom Characterization

In order to form a custom cell library and to be able to read it into EDA tools, it requires geometry information and timing characteristics of the library cells based on their physical implementation (layout). The geometry information is captured by the abstract generation process while Cadence Encounter Library Characterizer (ELC) can be used to generate timing, power and noise models of library cells for different process corners. Since this thesis is the continuation of the previous research [5] carried out at Chalmers, so the scripts for setting up the tool were used from the same work and modified accordingly.

5.1 ELC Basics

Encounter Library Characterizer is a characterization tool that supports characterization of cells for timing, power, noise and statistical variation for different process corners. It accepts SPICE or Spectre descriptions of the cells to be characterized and characterizes them for timing, and optionally for power, noise and variation depending on the settings passed to it [12].

Encounter library characterizer characterizes input gate capacitance using both rising and falling input signals while generates an output driving model called Effective Current Source Model (ECSM). Three process corners, namely typical-typical (TT), fast-fast (FF) and slow-slow (SS), are used to generate the output models. The ECSM model describes the effective current for different combinations of input slew rates and output loading capacitances. The output from ELC can be exported in LIBerty library (LIB) format. The LIB files from the characterizer combined with the LEF files from the abstract generator define the cell library of the custom cells.

ELC also has other inputs that specify the characterization environment. These support files set tool configuration environment variables, database commands, simulation settings, incremental characterization settings, variation parameters and miscellaneous settings like the cell footprint information. These are discussed in detail in the following subsections. Complete details about ELC may be found in [12, 13].

The results of characterization can be captured in the Advanced Library Format (ALF), the standard Library compiler (lib) format, or both. The cell characterization is carried out for Half Adder and Full Adder circuits, so designed to be used in the semi-custom implementation of the rectangular PPRT of the HPM.

5.1.1 SPICE Input Files

Library characterizer requires following SPICE input files to setup the characterization environment.

- SPICE format subcircuit (SUBCKT) files of the cells to be characterized. Support for both the SPICE and the Spectre descriptions are provided. The subcircuit description includes all the transistors and local RC component circuits defined for each standard cell.
- A SPICE-format device model file, which specifies the device parameters for the technology. It is usually provided by the foundry and is generally available in both the SPICE and the Spectre formats. If more than one model files are used, they may be invoked easily using the .lib command in SPICE specifying the process corner to be used. The general format is:
  `.lib'<model1.sp>'PROCESS_CORNER`

5.1.2 Command File

This file specifies the database commands that are used to access the library database for the cell being characterized. A comprehensive list of the commands can be found in [13].

5.1.3 Configuration File

The configuration file (always called elccfg) is used to setup the session environment variables and settings. It include details like the format of the input subcircuit files (SPICE or Spectre), device names in the model file (NSVT and PSVT for 90-nm STMicroelectronics devices) and characterization settings. A complete list of the configuration variables can be found in [13].

5.1.4 Simulation Setup File

This file contains the details about the SPICE simulation that is setup as part of the characterization process. ELC runs SPICE simulations for the vectors it identifies and generates for the cell being characterized through Piece Wise Linear (PWL) waveforms. More details about this file can be found in [12].

5.2 Methodology

The flow shown in Fig. 5.1 produces the timing and power information of the custom library cell(s) under consideration. This along with the LEF output from abstract generator fully defines the custom library. Once the library and technology files for the cells are available the characterization flow is complete and the custom cells may be read into Encounter with the `.lib` and `.lef` files as inputs, for the use in semi-custom implementation of the HPM multiplier.

The flow in Fig. 5.1 doesn't capture the incorporation of custom cell library in Wired environment which is used as a frontend tool. However, it is evident from the outputs from abstract generator and ELC that once a custom cell library is formed then, rest of the flow follows same routine like any flow using foundry provided cell libraries.

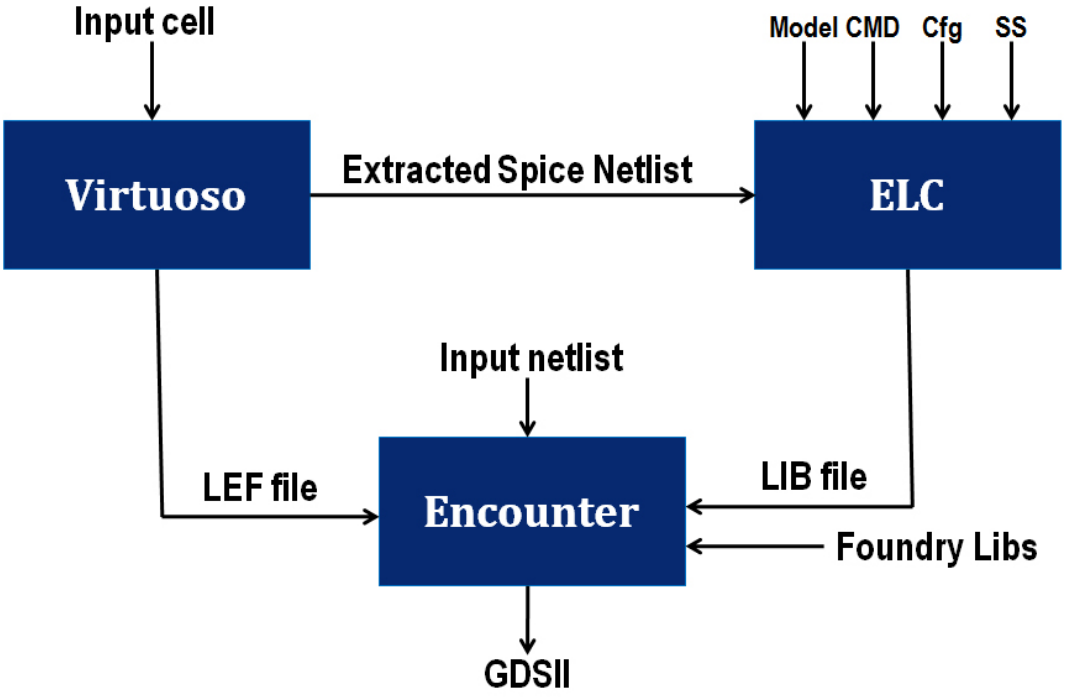


FIGURE 5.1: Custom characterization of cells using ELC

Part II
Layout Exploration

6

Methodology for Layout Exploration

In the study presented, an unconventional frontend design technique in Wired has been combined with the conventional backend flow associated with the digital circuits and applied to a library of cells custom created to test our hypothesis. The sign-off quality designs are not yet produced due the fact that all the efforts have been put to evolve a reliable methodology, where designers can employ custom library along with netlist from Wired into the conventional SoC Encounter flow to achieve significant gains in time. The work done as part of this project has opened up exciting opportunities in layout exploration. At this point fully routed designs have been produced. The scope of this methodology is still nascent and we hope that this work will enable accurate assessment of design decisions with reasonable effort. Fig. 6.1 shows the outline of this flow.

The sections following this, details the flow from a layout standpoint. It is assumed that complete HPM Wired descriptions are available as is the required RTL for the remaining blocks that are not implemented in Wired.

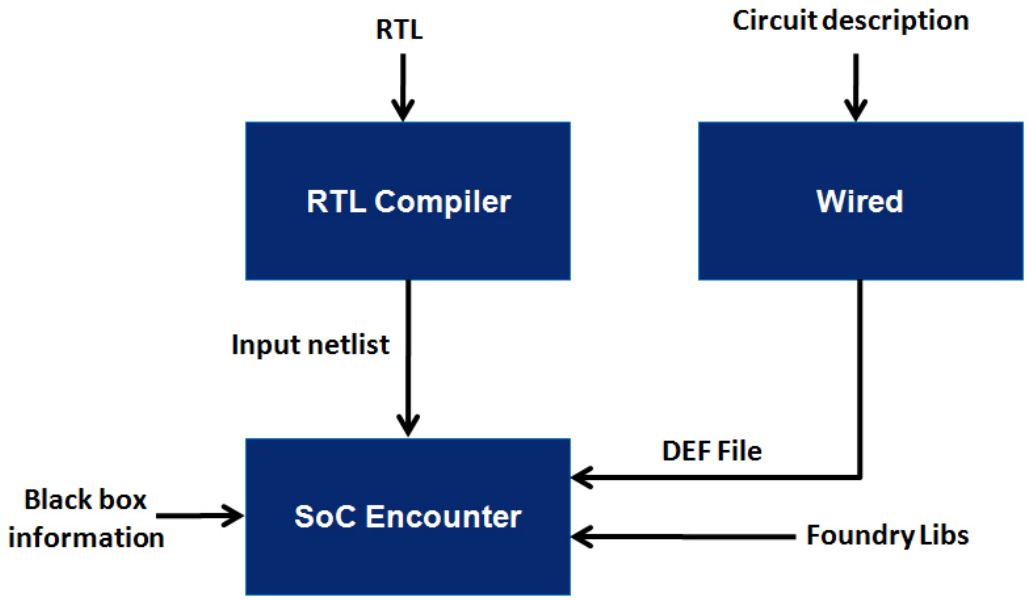


FIGURE 6.1: Flow chart for a Wired Encounter Flow Methodology

6.1 Physical Synthesis -RTL Compiler

The parts of the design where the designers anticipate benefits from constraining the placement are produced using Wired which is discussed in next section. Such parts in case of HPM are the PPRT and PPG [5]. The remaining portions, namely: the input and output registers, and the final adder remain as RTL implementations. These components are technology mapped to the 90-nm SVT library provided by STMicroelectronics, using Cadence RTL compiler, allowing for the existence of the PPRT and PPG as blackboxes. This gives rise to a block based or partitioned flow. The PPRT and PPG are incorporated into the design using SoC Encounter via the DEF file produced by Wired.

During the course of this investigation, since work from past research was brought together, so we rely on RTL netlists generated by Subramaniyan [5], but it is important to give an overview of how it was done in order to understand the underlying methodology.

The RTL for the HPM was generated from a PHP based generator maintained by Magnus Själander [14]. In order to accommodate the goals of this thesis, the RTL for the PPRT and PPG is replaced by the blackbox descriptions. This is implemented at the logic synthesis phase in RTL Compiler, where a technology mapped netlist of the HPM is modified to match the hierarchy and the port names with the one produced in Wired [5].

6.2 Wired as Frontend

A netlist consisting of the PPRT and the PPG was implemented using Wired for the purpose of this study. Wired is a hardware description language based on the functional programming language Haskell [15]. It elegantly describes the logic functions and cell placement of a design and is wire-aware, in the sense that basic timing information can be extracted. Wired adopts an applicative style, using labels to describe the logical functions. In theory, this means that we can dispense with any additional synthesis!

Wired also provides the designer with powerful tools like recursion and higher-order functions making it extremely simple to describe regular hardware structures like the multiplier reduction tree or prefix adders. Placement directives are simple descriptors to indicate the relative placement of the cell(s). These are also applicative and are part of the description. In order to allow interactive development Wired also provides a pictorial Postscript output. Finally, in order to interface with the P&R tools, the complete description can be written out in the Design Exchange Format (DEF). For a more detailed description refer to [16].

6.3 Backend Layout using SoC Encounter

The backend flow starts once a technology mapped netlist of the whole design and the DEF output from Wired describing the placement aware blocks of the design is available. SoC Encounter relies on number of commands, either based on Graphical User Interface (GUI) or command line prompt or even both. In adopting a partitioned flow, we make use of both of these types at different stages. The layout flow can be separated into four distinct steps, namely:

1. Floorplanning and Placement
2. Power Planning
3. Clock Tree Synthesis (CTS)
4. Routing

These are usually applied interchangeably, depending on the design, but in general follow the order listed above. In order to make the implementation flow clear, it is split into three subsections in light of choosing to implement multiplier designs as test candidates for the

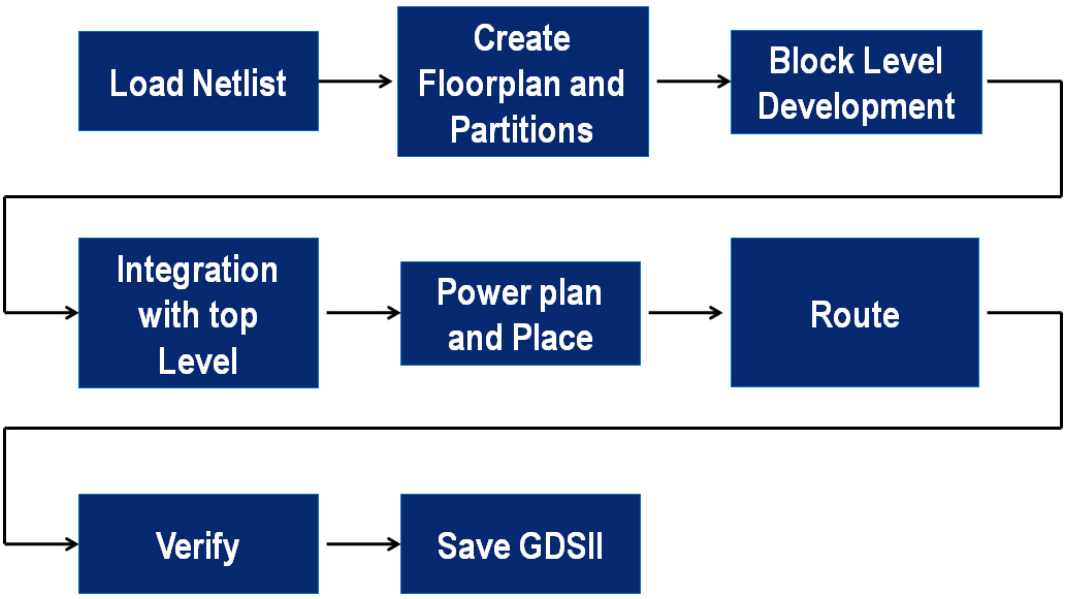


FIGURE 6.2: Flow chart detailing the Backend synthesis flow in SoC Encounter

methodology, and given the fact that a partitioned flow is to be used by necessity. Further, the layout steps listed above are highlighted as they appear in each implementation step. Fig. 6.2 shows how these steps are applied in general.

6.3.1 Setting up the Design

The first step is to setup the complete design. In this case, the input and output registers, and the final adder have distinct modules in the technology mapped netlist output from RTL Compiler, while the PPRT and PPG appear as empty modules with port connectivity to the rest of the design. The blackbox geometry is specified as a Library Exchange Format (LEF) file. This is important as, it decides the initial area for the complete design. While it does not have to be perfect, make sure a reasonable number is specified. It can be modified at a later time in the flow. The complete design, the timing libraries, technology files and blackbox LEF can now be imported into SoC Encounter with the switch to read empty modules as blackboxes set.

Floorplanning and Placement

With the design imported into the SoC Encounter environment, the initial chip floorplan with the design blocks placed outside should be visible. At this stage, discrete blocks like input and output registers and final adder are created and pre-placed. The blackbox (es) may be resized and/or re-specified if necessary. The required partitions are setup and pins are

assigned to them from the *Edit -> Pin Editor* menu. Also assign pins to the whole design using the same menu (fix the pins if necessary). Commit the partitions and carry out an initial non-timing driven placement to get an initial area budget. The partitions can be saved by using the *File -> Save -> Partition* menu. Exit the design using `freeDesign`. This command clears the design but keeps the Encounter session active. Listing the directories reveals that under the partition directories, the following partitions files have been saved: netlist, placement and Encounter configuration files.

6.3.2 Block Development

The previous step sets up the blackbox for further development. In this step we will complete the layout of the blackbox for integration into the top level later on.

Floorplanning and Placement

In order to read DEF netlists into Encounter, we must first have the standard cell libraries along with our custom cell library and technology files (LEF's) loaded. These are loaded using the `loadLibrary` and `loadleffile` commands respectively. These commands can be collectively put into a script file that can be sourced from within the Encounter session. Once this is done, the DEF file for the block which was obtained from Wired is loaded using the `loaddeffile` command. This command causes the Wired `<filename>.def` to be converted to `<filename>.def.v` for the use within the Encounter environment. Once the DEF file is loaded, we are free to perform any minor placement changes necessary, like creating fences or replacing components. The placement status of the PPRT cells is changed to Fixed while PPG are re-placed (non-timing driven in this case, as there are no synchronous elements), to get a better block floorplan. The status of the block is that it is now pre-placed but still has no routing information. In keeping with the philosophy of the flow so far, we have completed block development. Before proceeding further however, a few things need to be done. Firstly, in order to be able to convey the block details to the top level, save (replace) the netlist, that was originally saved while saving the partition. Also save the placement information as `<partition_name>.place.gz`. Secondly, as previously indicated the DEF file read into Encounter creates `<filename>.def.v`. Delete this file, so that no conflicts are caused when block assembly is carried out. Use `freeDesign` to unload the design after all the relevant files have been saved.

6.3.3 Block Assembly

Once all the blackboxes in the design have been developed, we can assemble the top level design. This is easily accomplished through the `updateBlock` command [17] from the design root directory. Before using this command check the top level partition configuration file (`<top_level_design_name>.conf`) to ensure that the blackbox LEF file is removed (It gets saved when the partitions are saved). Run `updateBlock` to merge all the block data into the top level. Running this command creates a flattened netlist of the top level design. The design flow now reverts to the conventional flow.

Floorplanning and Placement

The top level design can now undergo a final pass on Floorplanning and placement. The floorplan may be refined to get a better die area or aspect ratio. The timing constraints for the entire design can now be loaded and timing driven placement (full or incremental) can now be run. Care should be taken however, to ensure that all applicable placement constraints are in place. In the Encounter environment, this can be checked easily through the design browser.

Power Planning

Power planning involves setting up the power supply network to the whole chip. In case the design in complex, modular power domains may be setup in order to ease the load on the power network. The Encounter environment allows the user to setup power rings and stripes through a GUI with a number of options. Power routing may be completed through the use of sRoute command. There is also a GUI available for this under the *Route -> SRoute* menu. This command sets up alternating rows of supply and ground rails across the area of the die thereby enabling an abutted placement of standard cells.

Clock Tree Synthesis

Once the power planning step is complete, the clock tree may be routed globally. Encounter setups the clock specification through `*.ctsch files`. These clock specifications can either be manually created or Encounter can generate a default script that can be modified to suit the design needs. Save the clock nets of the design and proceed to route the global signals. The WRoute tool can be run to globally route only the clock and reset nets of the design by directing the routing tool to route only the named nets. The routing mode is set to global route with the timing driven option set. Proceed to detailed routing to complete the entire routing of the design Routing Once the clock tree has been synthesized all other nets

in the design may be routed. The WRoute tool can be invoked again, this time to run routing on all nets. A number of modes are available to the user to completely route the design. For the primary routing pass, choose Global and Detail Route with the timing driven option selected. Depending on the complexity, a number of correction passes may have to be made before a DRC compliant design is realized. Another routing tool called Nanoroute is also available to perform the same tasks of global and detailed routing. This tool however is a much more ”fine grained” tool than WRoute and performs better in optimizing the routing. It is invoked during timing optimization runs post CTS.

Part III
Multiplier Implementations

7

Results

The results presented in this section represents the work carried out in the course of this whole thesis. It is based on a hypothesis and some predictions from the previous work [5]. The idea implies that by having different pin orientations of the cells might result in some useful gain in terms of routing regularity thereby producing shorter routing wires and less number of vias. The aim of the research was to develop a methodology for carrying out the task defined above. The results obtained for the implementations of the test candidates for the underlying methodology proved to be very promising in terms of wire lengths and number of vias.

7.1 Test Case Implementation

The 90-nm GP library from ST is used for the physical synthesis of the rest of the blocks constituting 32-bit HPM multiplier. The two cases of rectangular PPRTs defined in section 1.2 of the thesis were implemented by following same constraints. Timing-driven placement was carried out with congestion optimization effort set to medium for both cases. Fig. 7.1 shows the placement of flipped cell case after power planning has been carried out. Since

the layout for normal cell case is not that visually different from Fig. 7.1 unless it is zoomed enough so that the pin names are visible, we elected not to have a screen shot of that. Timing-driven routing was done using NanoRoute with congestion effort set to 2.

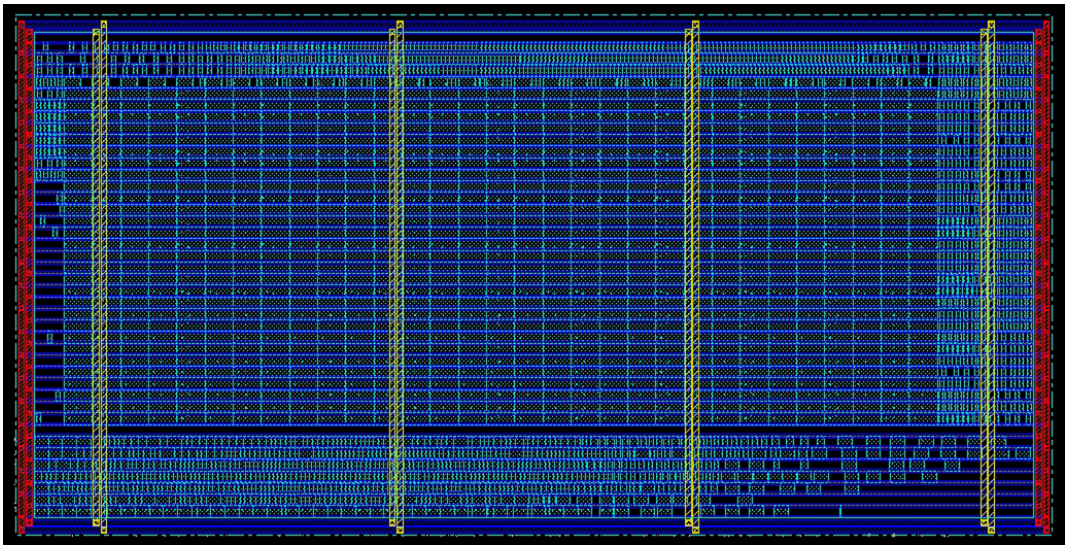
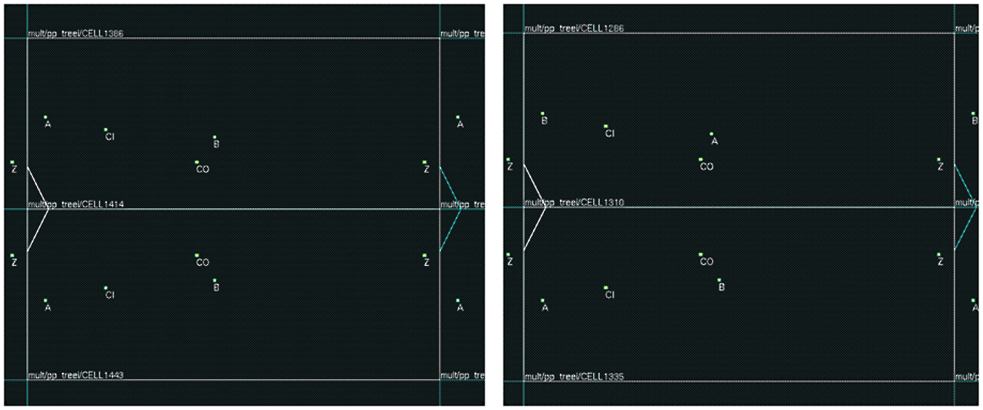


FIGURE 7.1: Power routed 32-bit rectangular HPM

Fig. 7.2 shows a blown-up view of the layout using flipped cells with customized pin locations. In this case, the pins are flipped about the x-axis with customized locations for the 'A' and 'B' inputs. As the results show, this minor change in placement turns out to be getting better values in terms of wire length and number of vias.



(a) PPRT having normal cells (b) PPRT having normal and flipped cells

FIGURE 7.2: Magnified version of PPRT cells showing different pin placement

7.2 Impact of Pin Placement

Table 7.1 shows the comparison results of the normal cell case and flipped cell case in terms of slack, wire length and number of vias. The slack figure for normal case is much better than the flipped case and it can be argued that the change in pin orientation didn't result in decent value in terms of slack. In order to get optimal slack values without losing substantial gain in terms of wire length and number of vias, one has to try out different pin orientations.

If we consider total wire length or average wire length values, it is pretty evident that the flipped case resulted in shorter wire length. Less wiring means impact of variability, in terms of absolute performance drop, ought to reduce. Less vias means higher yield. Form the table it is pretty evident that the flipped cell case has a lead on normal cell case in both regards. This is an important breakthrough as now we can mitigate the variability issue even on placement level.

	Slack (ns)	**Total wire length (μm)**	**Avg. wire length (μm)**	**No. of vias**
Normal	0.259	230,058	57.3	39,784
Flipped	0.185	208,938	52.0	24,352
ST Lib.	0.804	165,783	41.3	41,110

TABLE 7.1: **Comparison of wire length and number of vias for different implementations**

Table 7.1 also shows the test parameter values for the design case which employs only standard library cells. It is clear from the values that standard library cell case has better values in terms of slack and wire length from any of the case which uses custom library cells. It is because, the custom cells are not as compact and optimized area wise as compared to standard library cells, and the cell optimization requires more time investment. Since the focus of the thesis was to study impact of pin placement and not on having cells competing standard library cells, so we stick with the cell layout which look pretty decent. As a passing note, HA from custom library occupies the same area as the FA from ST library. Also the number of vias value turns out to be much worse for ST. Library case and this is also due to the same reason that non-optimized cells offer more space for over-the-cell routing thus requiring less jogging.

7.3 Conclusion and Future Work

The results presented above shows promising figures which leads to the fact that it is worth investigating the presented idea further and applying it to various other test cases and see the impact. From the perspective of the HPM for example, observations made in the results indicate the following:

- Pin orientation does have impact on routing. By supplying cell with many different pin placements, the overall wiring figures and number of vias tends to reduce. Both reductions in wire length and number of vias advocate regular routing and lesser jogging in metal layers. With shrinking process technologies, design variability is becoming worse, so any assistance in making the regular design tends to improve the variability.
- Although no power analysis was made in the present course of study, however the results suggest that by constraining the number of layers used and having shorter wire lengths, the power dissipation may be controlled as well.
- We need to have standard cell libraries providing various versions of same cells with different pin orientations if one cannot change the pin order itself.

There are other possibilities that need to be investigated further; for example: what, if the same idea is applied to other multiplier architectures like TDM (Three Dimensional Multiplier) or to arithmetic circuits exhibiting regular structures such as shifters and multiplexers? Shall there be any considerable gains in terms of routing regularity or not? For the rectangular HPM, only two different pin orientations were tried out in which only pin order of not so important pins (A and B) were changed while the remaining pins remained more or less same. This change caused gain in terms of wire length and number of vias but slack to decrease as well which is an undesired effect. It would be very interesting to try out different pin order for these pins as well and try to figure out optimal position.

In conclusion, the work carried out as part of this thesis presents design methodology to mitigate variability effects in cell based layouts. A semi-custom approach has been followed for HPM implementation which itself uses full-custom cells for the critical block i.e. PPRT. Since the present research was a continuation of research being made at department, it complements the Wired-Encounter methodology developed by Subramaniyan.

Bibliography

[1] L. Dadda. “Some Schemes for Parallel Adders”. *Alta Frequenza*, 34(5):349–356, May 1965.

[2] C. S. Wallace. “A Suggestion for a Fast Multiplier”. *IEEE Trans. on Electronic Computers*, 13:14–17, Feb. 1964.

[3] V. G. Oklobdzija, D. Villeger, and S. S. Liu. “A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach”. *IEEE Trans. on Computers*, 45(3):294–306, Mar. 1996.

[4] H. Eriksson, P. Larsson-Edefors, M. Sheeran, M. Själander, D. Johansson, and M. Schlin. “Multiplier Reduction Tree with Logarithmic Logic Depth and Regular Connectivity”. In *IEEE Intl Symp. on Circuits and Systems*, pages 5–8, 2006.

[5] K. Subramaniyan. “A Semi-Custom Approach to Exploiting Regularity in HPM Architectures”. Master’s thesis, Chalmers University of Technology, 2009.

[6] A. Qamar, K. Subramaniyan, and P. Larsson-Edefors. “Impact of Standard Cell Pin Placement on Routing Regularity of HPM Architectures”. Swedish System on Chip Conference, 2010.

[7] K. P. Subramaniyan, E. Axelsson, and P. Larsson-Edefors M. Sheeran. “Layout Exploration of Geometrically Accurate Arithmetic Circuits”. pages 795–798, Dec 2009.

[8] N. Weste and D. Harris. *“CMOS VLSI Design: A Circuits and Systems Perspective”*. Addison Wesley, 2004.

[9] *“Calibre Interactive User’s Manual,Software Version 2008.1”*. Mentor Graphics, 2008.

[10] *“Unicad 2 PLS user manual”*. STM, 2006.

[11] *"Virtuoso Abstract Generator User Guide, Product Version 6.1"*. Cadence Design Systems, 2006.

[12] *"Encounter Library Characterizer User Guide, Product Version 8.1.1"*. Cadence Design Systems, 2009.

[13] *"Encounter Library Characterizer Text Command Reference, Product Version 8.1.1"*. Cadence Design Systems, 2009.

[14] M. Själander. "Multiplier Generator", 2009. URL `http://www.sjalander.com/research/multiplier/`.

[15] M. Sheeran E. Axelsson, K. Subramaniyan and P. Larsson-Edefors. "Fast Layout Exploration Using the Wired System". Swedish System on Chip Conference, 2009.

[16] E. Axelsson. *"Functional Programming Enabling Flexible Hardware Design at Low Levels of Abstraction"*. PhD thesis, Chalmers University of Technology, 2008.

[17] *"Encounter Text Command Reference, Product Version 6.2"*. Cadence Design Systems, 2007.