DISSERTATION

DESIGN METHODOLOGY AND PRODUCTIVITY IMPROVEMENT IN HIGH SPEED VLSI CIRCUITS

Submitted By

KM Mozammel Hossain

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2017

Doctoral Committee:

Advisor: Thomas W Chen

Yashwant Malaiya
Sudeep Pasricha
Ali Pezeshki

ABSTRACT

DESIGN METHODOLOGY AND PRODUCTIVITY IMPROVEMENT IN HIGH SPEED VLSI CIRCUITS

Recent changes in the technology market demand faster turnaround of the design, and as a result, designers struggle to meet performance requirements under prohibitively expensive non-recurring engineering (NRE) costs. Increasing costs for the design, validation, and time to market are some of the most pressing issues for next generation microprocessor systems. Custom versus synthesis VLSI circuit design has been a lively debate for the last decade. However, the ever-increasing cost of designs in large-scale projects is becoming a bottleneck to the industry, thus, the wind is blowing strongly in favor of synthesis, especially when the modern synthesis engines are becoming more and more sophisticated in closing timing, completing routings, avoiding congestions and better interface with all the backend tools. Custom design advocates will argue that the last pico-second and milli-watts cannot be left on the table, but the reality is, they need to get over this mentality in favor of cost, time to market, and ability to adopt the last minute-change of the design.

As the industry moves into the next generation of microprocessor design, it faces growing complexity in device scaling, supporting algorithm, and time to market. The need for customization has never been greater to address specific needs for customers. Building a complex microprocessor system is complicated and time consuming. Thus, circuit designers need to face the reality and move forward for more automation in the design cycles. While the past generations of microprocessors had more custom circuit design to meet tighter cycle time

battle, more startups and IC companies are trying to design by moving towards common synthesizable design methodology, and in most cases, sacrificing the desired speed in favor of new functionality.

Needless to say, improving the synthesis methodology as an alternative to the custom circuit design is gaining high momentum in the industry because the custom design most often needs hand crafted schematics and layout design which are overwhelmingly time consuming requiring as high as 2-4 times development time to the synthesis. However, the key point is to stay within a similar area and power budget, and yet cut significant time to development cycle of the design. Study shows, with the help of advanced Electronic Design Automation (EDA) tool capabilities i.e. advanced algorithm for timing, power and congestion-mitigation, the synthesis-based-design is increasing by about 30% relative to its predecessor.

With the goal of mitigating the rising cost of designing custom macros and eliminating the limitation of the usages of smaller Random Logic Macros (RLM), in this dissertation, it is intended to work on a methodology and algorithm development that will enable the industry to synthesize custom and RLM macros into bigger macro, Multi-Million Logic Gate Synthesis (MMLGS), to improve the Physical Design (PD) resource efficiency. Additionally, MMLGS methodology most often requires embedded IP or custom components, which may require different power supply rail(s) and (or) sync-async clocking interface(s). One of the popular timing methodologies is to share positive slack across the latches to ease the timing burden from one cycle to another, thereby closing time-challenged and architecturally critical paths. Similarly, multi-power supplies are used in design to either dynamically control part of the circuits or statically isolate a portion of the logic to different power islands to control the total power consumptions of the chip. In a custom circuit design, designers carefully hand craft gates to

support these multi-clock and multi-power domains, satisfying the timing and methodology requirements. However, in a high-speed synthesis design environment, designers struggle to make sure multi-clock and multi-power interfaces are designed, placed, connected, and timed correctly. Identifying and applying the proper timing constraints such as “no cycle stealing” at synchronous and asynchronous (sync-async) domain interfaces in synthesis, unit, and chip timing, are of the essence. Even though some of these concepts are available now and were used in the past but they have very limited application to the custom design methodology only.

As the synthesis methodology for MMLGS is being developed, work in this dissertation will also include developing algorithms and design methodologies for multi-power and multi-clock domains to take advantage of slack sharing across all hierarchies of the chip design. The developed algorithm and methodology will: 1) improve physical design resource efficiency; 2) improve performance by sharing unused slacks efficiently at appropriate design hierarchies; 3) improve logic optimization by collapsing smaller macro boundaries; and 4) enable correct placement and connectivity of logic-gates in multi-power domain. Upon successful implementation of the design methodology approach for MMLGS with multi-power and multi-clock, these designs need to be functionally verified for different input stimuli at different frequencies. At the same time, the design must comply with all backend tools such as noise, electro-migration, timing, and of course all PD verification tools such as DRC, LVS, and YLD etc. to demonstrate electrical and physical feasibility of the design.

Overall, a synthesis based physical design methodology using soft hierarchy, interior pin placement, pre-placing critical logic, post-routing techniques, has been developed and proposed in this dissertation. The effectiveness of the proposed design methodology is illustrated using a very time and area challenged unit, the level-2 cache (L2 cache) unit. Additional synthesis based

physical design algorithms and methodologies have been developed for level translators at multi-voltage domains and slack sharing at sync-async interface paths at all level of PD hierarchies. Proposed design methodologies show how slack sharing can be an advantage to ease timing, and yet avoid potential meta-stability in the circuit. The developed methodology can save significant number of physical design resources because millions of gates can be packed in a synthesizable macro even in multi-clock and power domain instead of designing those macros individually in high-cost design flow. By adopting the proposed MMLGS methodology along with multi-voltage and clock approach in synthesis and timing methodology, development costs can be cut by about 50%, which is substantially significant PD resource savings in high-cost VLSI circuit design.

ACKNOWLEDGEMENTS

As much, and perhaps more than, this dissertation represents my personal abilities, it represents how lucky I am to be surrounded by incredible people who guided me, provided a shoulder to lean on, and examples to live by. First and foremost, I would like to thank Prof. Tom Chen, my advisor, for his guidance, encouragement, and suggestions during my research work with him. Prof. Chen provided me with all the tools necessary to complete my research within a short period of time. I am ever grateful to Prof. Chen for his commitment to excellence in research and the opportunity given to me to complete my dream! My heartfelt thanks go to all my committee members, Prof. Yashwant Malaiya, Dr. Sudeep Pasricha, and Dr. Ali Pezeshki for their help in shaping up my research with their invaluable critiques and suggestions.

IBM is a wonderful place to work, with many wonderful mind and people, many of whom have become my friends, colleagues, and mentors along the way. A great and special thank to my consulting advisor, Dr. Vikas Agarwal. Over the years, he enabled me to think and look beyond the scope of the research topic and provided direction with his thoughts and ideas. An enormous thank you and gratitude is also sent to Joshua Friedrich, my mentor, for his help and guidance from the day one here at IBM. I am not shy to say that I would not be where I am today without his help and the opportunities given to me. Furthermore, I am grateful to a good friend of mine, John Badar, for his help, honest feedback on my dissertation topics, and helping me to stay positive all the time. I wish to extend many thanks to my managers Sam Thomas, Mike Carlson, Gilbert Prince, and all my colleagues for their day-to-day help and providing me a shoulder to lean on when it is needed the most.

When I look at my life, I go back all the way to my school life and remember very important contributions that shaped up my life. My elementary school teacher, Abul Hossain, and high school teachers Shushanta K. Roy, Abdul Baten Selim, Abdul Gafur and Fazlul Haque, all played a crucial role and helped me to grow and think out of the box at very early stages of my life. I would like to extend my sincere gratitude to these wonderful teachers of my life.

I grew up with amazing brothers and sisters, all of whom provided me unparalleled love and affection that I cherish all the time. Mosharrof Hossain, Monzed Khan, Mokbul Hossain, and Masud Khan are not only my brothers, but also my dearest friends. Their wives and children are amazing family members who comforted me with their affection in my bad and good days. I am truly honored and grateful for such a wonderful family that is second to none. I would also like to thank my wife's parents, Mr. Shamsul Haque and Mrs. Rabeya Begum, who supported our decision to live on the other side of the world, even though I know how tough this was for them, and for making constant trips to visit and help us during our need.

I am in debt to my parents for their unconditional love, sacrifices, and prayers throughout my life. When I put my Ph.D. work on hold, I made my father upset the most. My father flatly told me, "I did not send you to the USA for job, I sent you for Doctoral Degree". It is amazing how his favorite quote, "Try and Try, Again Try, You Will Succeed," influenced me from my childhood until now. I am certain you, my dearest friend, mentor, and dad, looking from the heaven and resting in peace, knowing that I never gave up and fell short to fulfill your dream!

Finally, my greatest thanks are to my sons, Mushroor and Mainur, and my daughter, Samia, who joined us along the way and made our lives so wonderful. You all are the best gift of our life! And to top it all, Sharmin, my love, there are not enough words to express my love and gratitude to you—this Ph.D. is as much yours as it is mine!!

DEDICATION

To my late mother, beautiful wife Sharmin, amazing sons Mushroor and Mainur, wonderful daughter Samia, all my brothers and sisters, especially my brother Mokbul Hossain

and

dedicated with love to the memory of my mentor, best friend and father: Abdul Latif Khan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PROPOSED ALGORITHMS

Chapter 1

Introduction

In high-speed microprocessor design, the most concerning issue is product design and verification costs. One direction that industry has attempted, with the goal of mitigating the rising costs of per-application designs, is to move toward with synthesis instead of custom design, because in custom design, major re-engineering is required when moving from one technology/design point to the next. Thus, the focus is to find a design methodology to shorten the design cycle and lower development costs.

In the past 20 years, microprocessors technology has experienced improvements in circuit integration and microprocessor throughput. The technology has grown rapidly due to transistor speed, energy scaling and core micro architecture advances powered by Moore's law. In every generation (two years), transistor density has doubled as their dimensions have been reduced by 30% (shrinking their area 50%), and circuits have become 40% faster increasing the whole system performance [1]. However, due to the battery capacity and chip reliability (heat dissipation limits), power consumption has been one of the key limiting factors for performance scaling in the single-core microprocessor technology. In the past decade, multi-core microprocessors have become the major design trend. Limits in instruction level parallelism (ILP) and power dissipation constraints have triggered the high-performance microprocessor roadmap to enter the multi-core era, starting from the high-end server processors and moving to the low-end hand-held mobile device processors. A multi-core micro architecture provided an effective alternative to improve throughput performance of parallel programs while keeping power consumption under the control. To improve efficiency, single-thread performance

was sacrificed and instead multiple cores were joined on a single chip when more transistors became available [2]. The more threads accommodated in the application set, the more efficient the processors became [3-4]. Recently the typical pattern among multi-core CPU products is to keep the number of cores constant within a generation and double the number of transistors within each core [5]. By exploiting Moore's Law to replicate cores, multi-core architectures increased computational performance. However, there is no real benefit if the software has no parallelism [6]. Fig. 1.1 is an illustration of the microprocessor roadmap for industry leaders (IBM, Intel and AMD) from technology to technology over the past 12 years in order to cope with increasing demand for on and off chip performance including new functions within reasonable power and manufacturing cost.

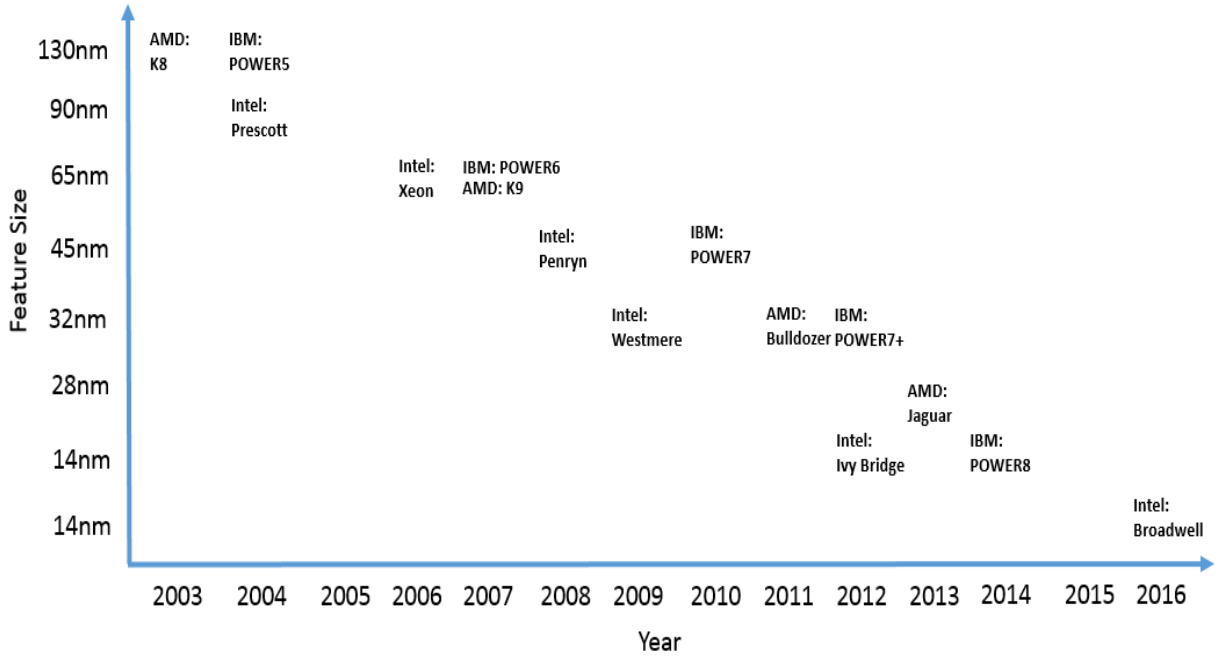
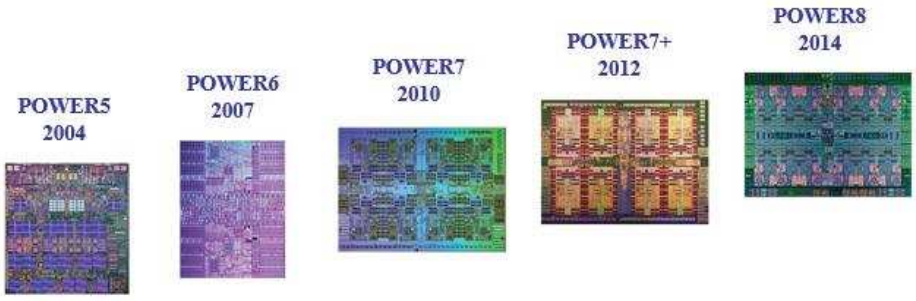


Fig. 1.1: Microprocessor road map of IBM, Intel and AMD [7-9].



Technology	130nm SOI	65nm SOI	45nm SOI eDRAM	32nm SOI eDRAM	22nm SOI eDRAM
Cores	2	2	8	8	12
Threads	SMT2	SMT2	SMT4	SMT4	SMT8
On-Chip Cache	1.9MB	8MB	2+32MB	2+80MB	6+96MB
Off-Chip Band Width	36MB	32MB	None	None	128MB
Sust. Mem.	15GB/s	30GB/s	100GB/s	100GB/s	230GB/s
Peak IO	6GB/s	20GB/s	40GB/s	40GB/s	64GB/s

Fig. 1.2: IBM's POWER microprocessor delivery roadmap [7].

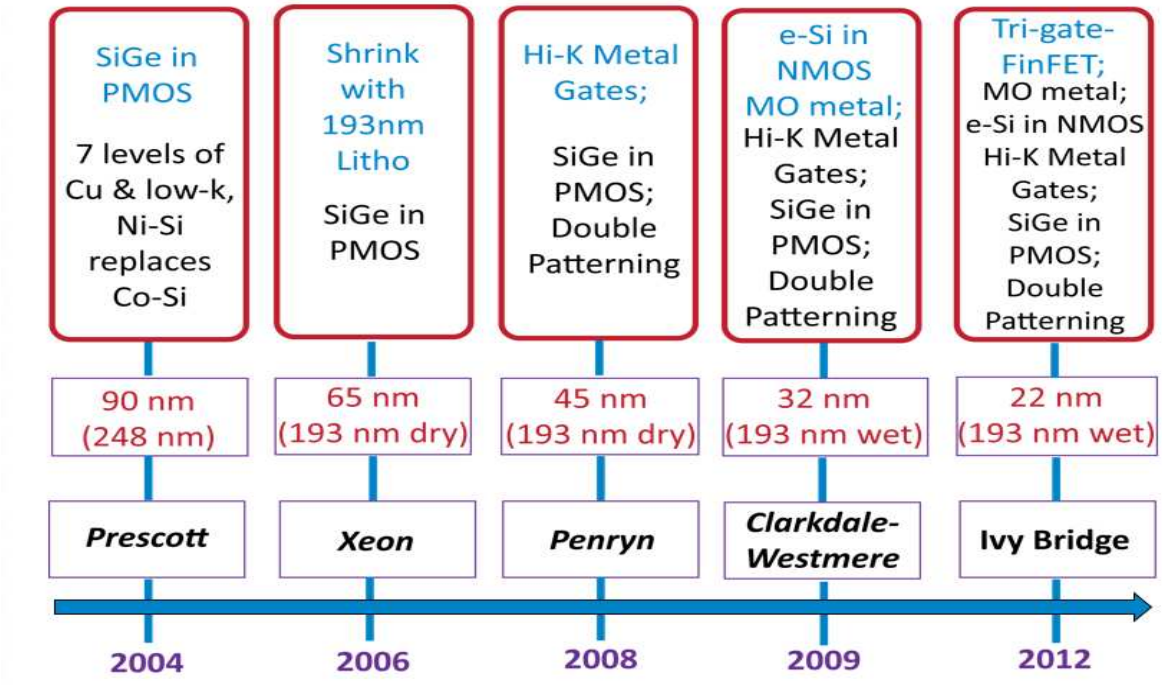


Fig. 1.3: Intel's technology roadmap for microprocessor design [8].

Fig. 1.2 shows IBM's delivery of high performing POWER microprocessors enabling multi-thread core, bigger on and off-chip cache and higher IO bandwidth [7]. Similarly, Fig. 1.3 shows the technology roadmap for Intel enabling high performing process technology which allows to pack billions of transistors on a single die within reasonable power envelop [8]. To support the ever-increasing demand for delivering high performance microprocessor in 2-3 years within a reasonable power and cost envelope, industry leaders are moving towards mostly automated synthesizable design with multi-clock and multi-power design methodology. In the current state-of-the-art design methodology for microprocessor, macros (generally within same power or clock domain) are built, timed and integrated at higher level of design hierarchy, called unit, to make the design constraints. However, these macro boundaries can be collapsed and turn-around-time (TAT) of current synthesis methodology can be improved even further for better physical design efficiency, sharing of unused slack and better logic optimization in multi-power and multi-clock domain.

In this dissertation, a Multi-Million Logic Gate Synthesis (MMLGS) design methodology is presented. The proposed design methodology differs from the methodologies used in the existing designs with four distinctive features:

- it allows to build a macro that is 2-3 times bigger in transistor count than what is currently done in the industry by using techniques such as soft-hierarchy, multi-VT insertion, interior pin and preplacing critical logic gates,
- it saves around 50% physical design resources (engineering man-month) due to higher degree of automation in MMLGS, otherwise traditional flow would require more circuit design, unit timing and integration resources,

- it allows designers to share unused slack safely in multi-clock domain to close timing critical paths effectively, and
- it enables designers to place and connect logic gates to proper power supply in multi-power domain design automatically and more optimally.

The proposed design methodology explains step-by-step of the physical design flow and shows how the development cycle and custom circuit design resources can be cut down by about 50% and yet maintain the same design quality. With the proposed design flow, the timing and integration resources can be eliminated in some cases, which are needed in today's design methodology to deliver high quality routed and timing closed unit to the chip. Synthesis-based design methodology can also improve power, area and timing efficiency by looking at the logic gates and drive strength across the macro boundaries. The presented data show how localized macro congestion can be resolved by sharing routing layer resources with a unit or chip. While the synthesis-based design methodology for MMLGS is proposed in this dissertation, it is also shown how techniques such as pre-routing, pre-placement, and a "soft hierarchy" (SH) can be used as aids to resolve critical timing and routing issues that are most often encountered [10].

Similarly, out-of-phase clock and multi-voltage are two major requirements as more and more functions are bundled on the chip to deliver high performing circuits within a reasonable power envelope. In high speed microprocessor design, multi-clock, multi-voltage usages, and slack sharing methodologies are very common in custom circuit design areas, but in automated synthesis design flow, these methodologies need to be analyzed, understood, and implemented in such a way that backend toolsets understand timing and implementation checks. Designers generally apply the tricks of stealing time from the next cycle if there is available slack. However, sharing slacks cannot be done between out-of-phase clocks because the likelihood of

metastability increases at synchronous and asynchronous boundaries, and can cause an unstable state in circuits. Similarly, signals crossing from one voltage domain to another must be interfaced through the level shifter buffer which appropriately shifts the signal levels. Design of a suitable level shifter is a challenging job. Application of multi-voltage can be classified as: 1) Static Voltage Scaling, 2) Multi-level Voltage Scaling, 3) Dynamic Voltage and Frequency Scaling, and 4) Adaptive Voltage Scaling. These requirements of multi-voltage should be taken into consideration during physical implementation of the electronic circuits. Recent trends in physical design methodology show a big shift toward MMLGS for design productivity and time to market. MMLGS design most often requires embedded IP or custom components, such as arrays, inside the macros that may require different supply rail or sync-async interfaces [11].

In this dissertation, timing algorithm and synthesis methodology for MMLGS have been developed and proposed for sync-async interface logic, allowing cycle stealing across all hierarchies of the chip design. Also, design interfaces for multi-voltage have been analyzed and methodologies have been proposed to support the algorithms in automated MMLGS design flow. The proposed design methodology provides new paths for closing timing while shortening the design time. Developed synthesis and timing flow allow designers to share unused slack safely in multi-clock domain and automatically place and connect logic gates to the proper power supply in multi-power domain design. Compared to the baseline experiment, the experimental results show that physical design development costs can be cut by about 50% using the developed design flow.

Chapter 2

Overview of Existing Synthesis Methodology

The technology market demands faster turnaround of IC design, and designers struggle to meet performance requirements as shown in Fig. 2.1. The microprocessor design industry has seen increasing costs for the design, validation, and time to market for the last decade. Past generations of microprocessors had more custom circuit design to meet the tighter cycle time battles. However, to meet the demand for shorter turnaround time, synthesizable design methodologies are preferred in most cases, sacrificing the desired speed of the chip in favor of new functionality and time to market.

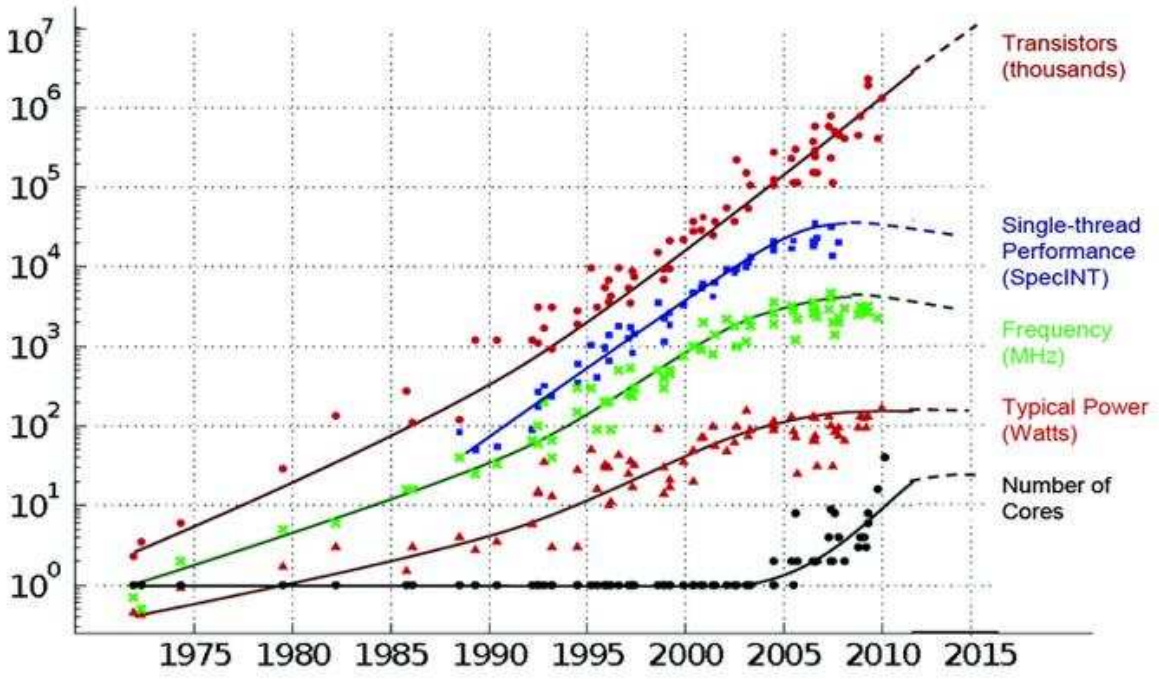


Fig. 2.1: Microprocessor trend data for 35 years [12].

To meet the design constraints such as area, timing and power, custom macros most often need hand crafted schematics and layout design which are overwhelmingly time consuming and thus requires as high as four times development time compared to synthesis flows. In general, custom circuit and layout designer must plan every detail including the physical placement of design up front. One of the problems with custom design approach is to incorporate any late design change which may sometimes cause a rip-up off previously built and timed macro. In a full or semi-custom design flow, bug fixes and late changes in the design most often become very expensive in terms of turnaround time. Unit like level-2 cache (L2 cache) would have more than 50 physical partitions and thus require more man power to comply with late logic change requests. To overcome these limitations, current industry trend shows a big shift towards synthesis based design flow. While this dissertation is based on work done in MMLGS and its requirements to deal with sync-async and multi-voltage interface, basic synthesis flow, tool and methodology will be discussed in this chapter to lay down a good foundation for the research topic of this dissertation.

2.1 Synthesis in Today's VLSI Design

Synthesis is very popular in the ASIC world due to its low efforts and faster turnaround time. The leaders in VLSI circuit design, such as IBM, Intel, and AMD are moving towards synthesizable design at a faster pace, where most of the designs are done in ASIC methodology and synthesis tools are being used extensively. Fig. 2.2 shows the relationship between design effort and the level of customization for different circuit design styles [13] that justifies the need for synthesizable design. The major advantages of synthesis-based methodology can be summarized as follows:

- Automatic logic to schematic translation

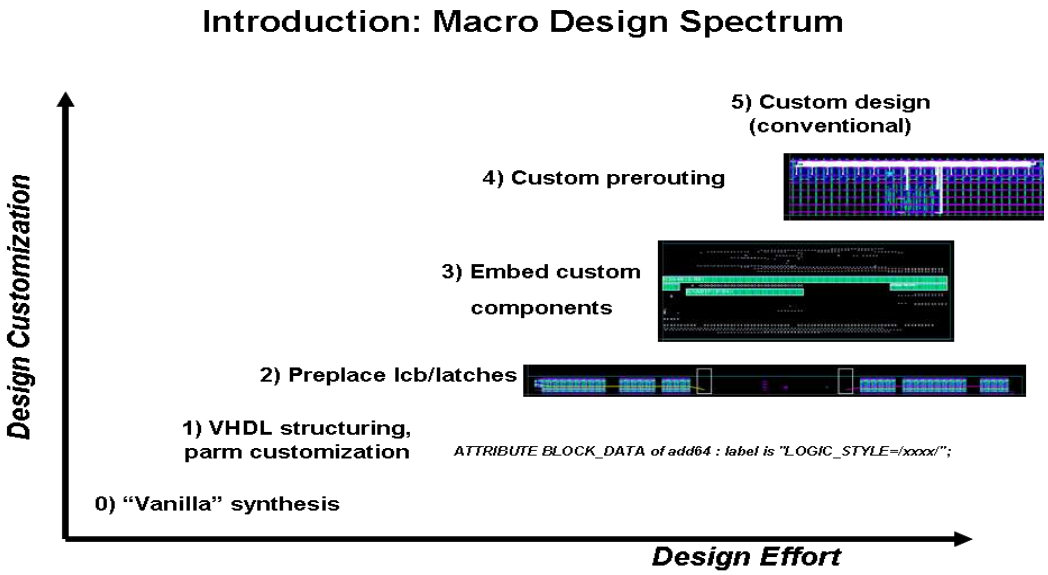


Fig. 2.2: Macro customization vs design effort [13].

- Reduced design cost because of faster turnaround
- Easy to incorporate last minute design change and
- No manual layout design is needed

Synthesis has been widely used in the past in developing random logic macros in high latency domain. However, with recent development in EDA tools such as congestion [14-15], timing [16], and noise [17-18] aware router, a fully automated solution provides adequate results with much shorter Turn Around Time (TAT) and improved Quality of Results (QoR). Thus, more and more complex units are being designed with more synthesized macro as technology industries move from one technology to the next. Fig. 2.3 shows how macro design methodology is changing from custom to hybrid (mixed of custom and synthesis) to RLM (Random Logic Macro) to LBS (Large Block Synthesis) methodology over the last six generations of microprocessor design for an IBM's L2 cache unit [13]. As the use of automated synthesis design flows gains popularity, better understanding of tools' behavior under different input

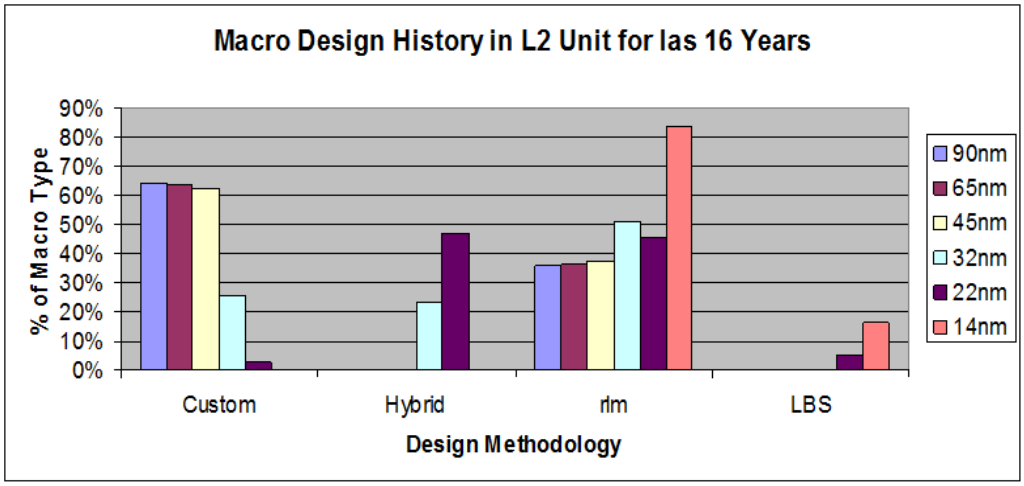


Fig. 2.3: IBM's L2 cache macro design methodology trend [13].

constraints is crucial to successful outcomes. The final design quality of synthesized macros using modern synthesis tools can be very sensitive to inputs and design constraints given by the designers. The quality of inputs and design constraints of synthesis tool need to be carefully examined to achieve desired outcomes. Similarly, to use this synthesis tool (including timing aware placement and routing and post-processing automated tools) wisely and intelligently, designers must understand the design flow, placement of critical logic, and in some cases, wiring to solve timing and congestions of the design. Thus, before diving deep into MMLGS, basic synthesis flow is discussed in this chapter with examples and illustrations.

2.2 Inputs to Synthesis Tools

To have synthesis tools to deliver intended results, it typically requires at least three user defined inputs and these are:

- HDL (Hardware Description Language) or functional logic
- Physical abstract information about the target macro, i.e. a physical bounding box

that has all the location of primary input and output of the design and routing constraints for the macro and

- Timing constraints for the macro including the clock names and phase information. These constraints are often referred to as assertions.

In addition to these primary requirements, tools most often need guidance from users in the form of control files, so that they can produce timing- and routing- closed physical design with expected results. In this dissertation, IBM's synthesis tool flow called PDSRTL, Placement Driven Synthesis (PDS), has been used as an example of modern synthesis tools to analyze experimental results. Some of the methodologies of PDSRTL have been cited here as general background information.

2.2.1 Register Transfer Level (RTL)

The synthesizable Register Transfer Level (RTL) description, often in either Verilog or VHDL hardware description language [19], can be optimized with respect to various constraints, including timing and/or area, using a synthesis tool, where a technology library file is being used to specify the components to be used by design. The RTL can contain parts of the design that are purely combinational or sequential, such as latches and flip-flops. In many cases, top-level RTL calls another component of the design as soft copy (i.e. another piece of logic) or hard copy (components that are built and timed separately) of some functions. Fig. 2.4 shows an example of high-level function. It is the task of physical design tools (synthesis or custom design) to translate these functions into transistor level for building the chip. Physical design engineers spend good amount of time to study the RTL to determine the data flow of the design and plan up-front to place the logic gates, metal usages and routing strategy of the design.

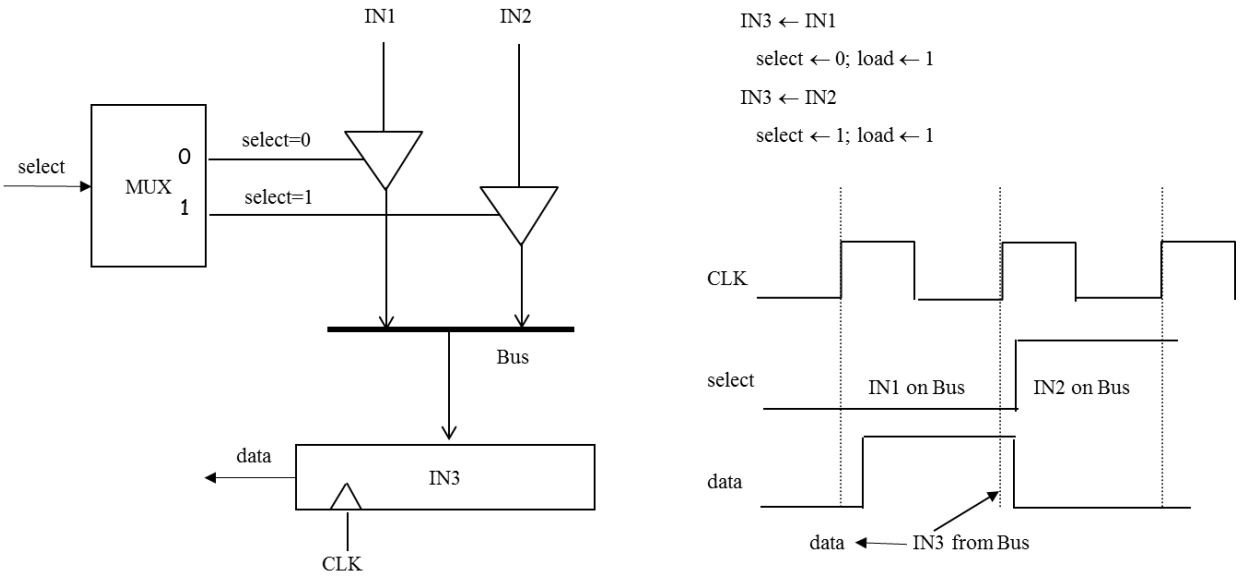


Fig. 2.4: An example of high level function.

2.2.2 Assertions

To guide the synthesis tools to close timing at input/output interface paths at macro boundary, timing and loading requirements are needed so that EDA tool work hard to meet the design constraints. Assertions, in terms of arrival time at input with correct timing phase (arrival time relative to functional clock), expected time of arrival at output relative to functional clocks and loading at macro output, are three minimum inputs to the synthesis tool. In today's high speed design, there might be many functional clock signals operating at different speed relative to each other and they are defined as phase. In synthesis, the assertion requirements are defined a bit more in details as follows:

Primary Input (PIS): To optimize the timing and gate placement, synthesis tools need to know the data arrival time (late and early mode to cover both slow and fast design corner), functional phase tag (data arrival time with respect to the functional clock) along with the slew (rise/fall time) of the primary inputs, so that design constraints are met during synthesis

optimization. The PIS file is generally a space separated constraint file, which is being used as an input during the synthesis flow. An example of PIS can be shown in table format in Table 2.1.

Primary Output Load (POS): The macro output requirements, in terms of arrival time at the output, slew (or rise/fall time) and capacitive load to drive, are required so that the timing and electrical requirements are met at higher level of the design hierarchy. POS value is given in terms of best and worst case capacitive load so that macro is designed to meet the timing for wide range of process corners. Similarly, POS value is also given in both lumped and effective load so that macro is designed to drive a wide range of load to drive at the next hierarchy of the design. The POS file is generally a space separated constraint file, which is being used as an input to the automated tools during the synthesis flow. An example of POS can be shown in table format in Table 2.2.

Table 2.1: Example of PIS assertions.

Input Pin Name	Phase	Late Arrival Time (rise)	Late Arrival Time (fall)	Slew (rise)	Slew (fall)	Early Arrival Time (rise)	Early arrival time (fall)	Slew (rise)	Slew (fall)
nclk	M-	125	0	50	50	125	0	50	50
data	M@L	100	100	80	80	80	80	60	60

Table 2.2: Example of POS assertions.

Output Pin Name	Worst Case Total Load (ff)	Best Case Total Load (ff)	Fan Out	Worst Case Effective Load (ff)	Best Case Effective Load (ff)
Reload_Bus	200	100	1	150	80
abist_en	150	80	1	100	60

Table 2.3: Example ETA assertions.

Input Pin Name	Phase	Late arrival Time (Rise)	Late Arrival Time (Fall)	Early Arrival Time (Rise)	Early Arrival Time (Late)
rd_data(0)	M@L	200	205	150	150
scan_out	S@L	400	410	200	200

Expected Time of Arrival (ETA): The ETA file is generally a space separated constraint file, which is being used as an input during the synthesis flow. An example of ETA can be shown in table format in Table 2.3. ETA consists of expected arrival time at the macro output (with respect to clock) along with clock phase information. Like PIS, ETA values are also given in terms of early and late arrival time of the signals so that macro is designed to function in both slow and fast corner of the process.

In addition to PIS, POS and ETA, synthesis engines can be guided for better results with additional information such as phase adjust and loading override files, where designers can override with updated phase information and input capacitance seen by tool for multi fan-in input of the design.

2.2.3 Macro Physical Abstract

Another necessary input to the synthesis tools is the macro bounding box with primary input/output (PI/PO), PIN locations, and metal blockages information. Synthesis tools read the physical placement of the macro PI/PO from the macro abstract and optimize the gate placement with proper drive strength as per the loading in POS file and at the same time, meeting the expected timing as defined in ETA file. When it comes to route the macro, router uses available routing resources as defined in the bounding box or abstract of the macro.

2.2.4 Control Files

Control file(s) might be thought of as additional help to synthesis but not mandatory. This control file(s) are called during the synthesis optimization process. However, for meeting the timing and quality of routings of any wires and placement of logical gates, these parameters should be considered as semi-requirements for the synthesis tools. For example:

- If the design is dominated by embedded IP, then synthesis will need IP placement information and timing rule to meet the timing and routing requirements.
- If the design is highly timing critical, the tool will need some guidance as how to use upgraded wires or use better VT (Threshold Voltage) gates to meet timing.
- If the design has a congestion problem, the tool will need to know how to spread logic to mitigate the wiring congestions.
- If certain portions of the logic needed to stay together, synthesis tool can be a guide as how to place the relevant logic gates together and so on.

2.3 Typical Synthesis Flow

Synthesis tools compile RTL design using two main phases: 1) Technology independent phase where the design is read in and simplified in combinational logic and 2) technology mapping where the design is mapped to available logic gates in a given physical library. However, before that there are few sub-steps such as optimizing clock logic and logic re-structuring in the flow. In merging clock logic step, synthesis collapses incoming clock logic into one Local Clock Buffer (LCB) per clock domain. Once latch placements are done, to make wiring and timing requirements for the design, appropriate LCBs are connected for each clock

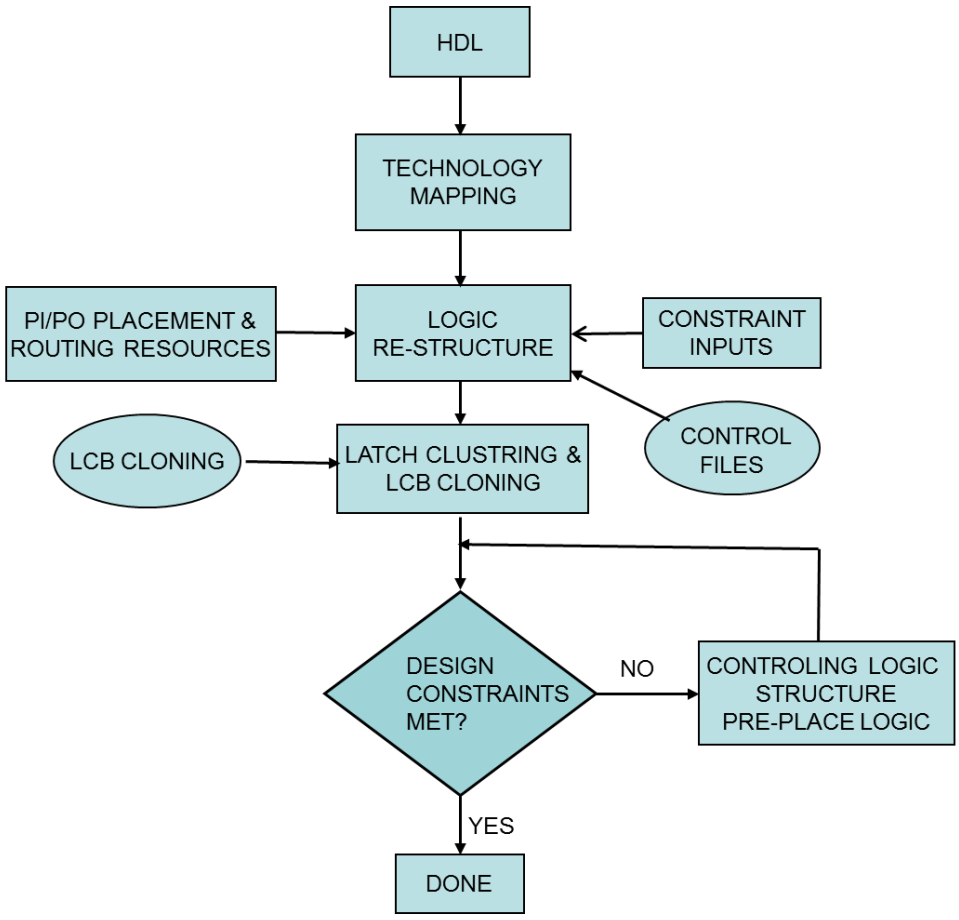


Fig. 2.5: A typical synthesis flow.

domain. LCB creates clock nets for master-slave and scan clock of the latches from mesh clock and other clock control signals as defined by methodology of each project. While inputs to synthesis are given, as discussed in previous sections, tools process this information in several steps to find an optimal solution for a given design. A typical synthesis flow is shown in Fig. 2.5. The flow contains many iterative steps and loops to satisfy the design constraints and timing requirements of a design. These steps include:

- Technology mapping
- Logic re-structuring

- Latch clustering and LCB cloning
- Controlling logic structure
- Pre-placing logic

2.3.1 Logic Re-structuring

Logic restructuring involves the following activities in synthesis:

- Optimizations are done on technology-independent netlist of the design
- Wire load is assumed to be zero
- Restructure logic to decrease network interconnections, circuit area, and remove logic redundancies and
- Algorithm used to minimize logic

Some examples of logic transformation and restructuring during synthesis have been discussed as here:

a) Transduction: Replace functions with more efficient ones. In the following example, Fig. 2.6.1, logic can be rearranged so that the delay is optimized. As oppose to connect signal "S" to the first level of gates, it can be forwarded to the second level by keeping same functionality to meet the timing.

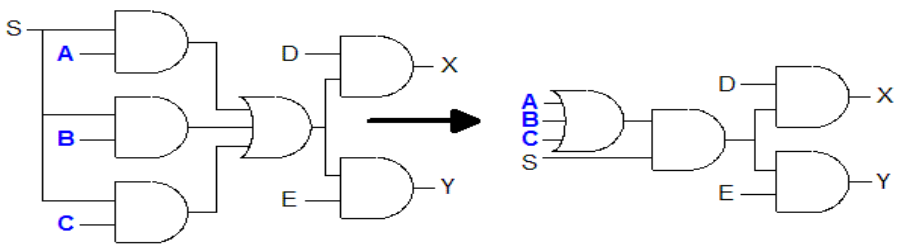


Fig. 2.6.1: Logic transduction (Source: EDA, IBM).

b) Cube Factoring: Extract a common gate from several gates as shown in Fig. 2.6.2, where a common function for input "C" and "D" is determined and then fed into the subsequent logic.

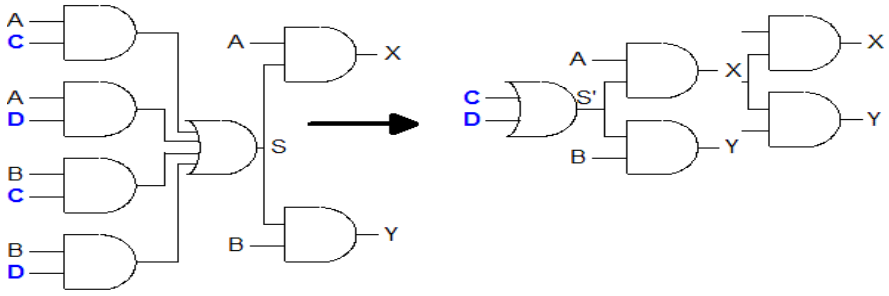


Fig. 2.6.2: Logic cube factoring (Source: EDA, IBM).

c) Create common factors: Creating common factor, Fig. 2.6.3, and then feed it to rest of the logic helps closing timing.

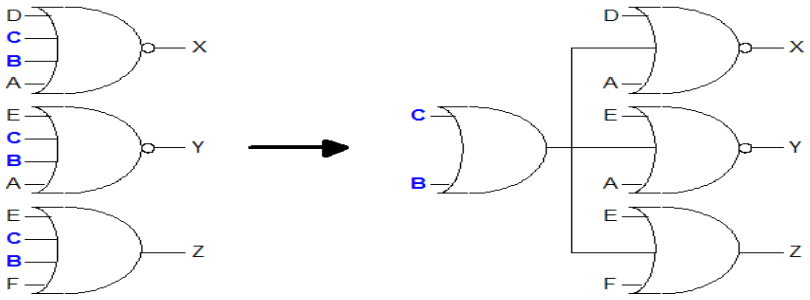


Fig. 2.6.3: Common factoring (Source: EDA, IBM).

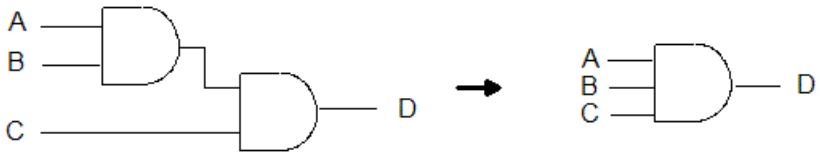


Fig. 2.6.4: Logic merging (Source: EDA, IBM).

d) Merging logic: Sometimes merging logic gates helps closing timing. It also depends on how critical is the timing for the input(s). In Fig. 2.6.4 both NAND2 gates can be merged to create NAND3 to improve the overall timing. However, if signal "C" is critical, this type of merging will hurt the timing on "C" because NAND3 is a slower logic-gate than NAND2.

2.3.2 Technology Mapping

Given a technology independent structural description, a target technology, and a set of design constraints, technology mapping is the process of implementing the structural descriptions in the physical domain at the same level of abstraction, where all design constraints are fulfilled. In this step of synthesis, logic gates are mapped to a specified design technology library for the project. For example: "REG" is defined as state storing element "latch" in RTL, however, during this step, tool will look for equivalent functional logic gates such as "nlat", "eslat" etc. in the adopted project. During the technology independent phase of synthesis, the Boolean equations representing the logic network are subject to logic optimization for minimizing the number of literals, which have been shown to correlate well with total cell area [20]. Fig. 2.7 gives an example how physical library is mapped to technology independent physical library gates. Optimal gate placement is a crucial part in synthesis. A timing aware placement and optimization algorithm has the following steps:

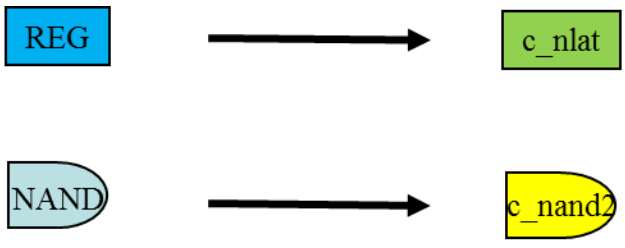


Fig. 2.7: Technology mapping.

a) Initial Placement: Based on the Steiner modeling of the net [21], synthesis use placement algorithm for initial placement of the logic gates. This step is basically based on the wire length between the logic gates. Once initial placement is done, synthesis can be guided with some parameter for a targeted logic-gate-placement-density in certain area of the design. Similarly, if the design is very congested, a set of parameter can be used to spread out the logic gates. To avoid the high pin density of certain gates, synthesis parameter could be very handy to avoid local congestions. Global placement is one of the most important steps in a physical synthesis flow. The last few years have witnessed a renaissance in the research on wire length-driven global placement, largely due to the availability of challenging benchmarks derived from real industrial design [22-23].

b) Placement Optimization: Placement-aware synthesis attempts to use a placement tool [24-25] to perform a partial physical design to compute more accurate interconnect loading estimates on a net-by-net basis. After the initial placement, synthesis algorithms go through several iterations of the placement to optimize the timing. In this phase, the tool determines which gate-movement would result in the most benefit to resolve timing violations. In Fig. 2.8, the dotted box is a new placement of the gate that is driving three blue boxes on the right, yielding minimum movement to solve the timing pressure.

There are several parameters being used in synthesis in order to meet timing and improve the routability of a given design. Here are some examples of most frequently used parameters, which can be used to guide placement of the logic gates:

- target_density: Controls the density of gate placement to avoid congestion.
- place_xfactor: Spread out complex logic gates to avoid congestion.

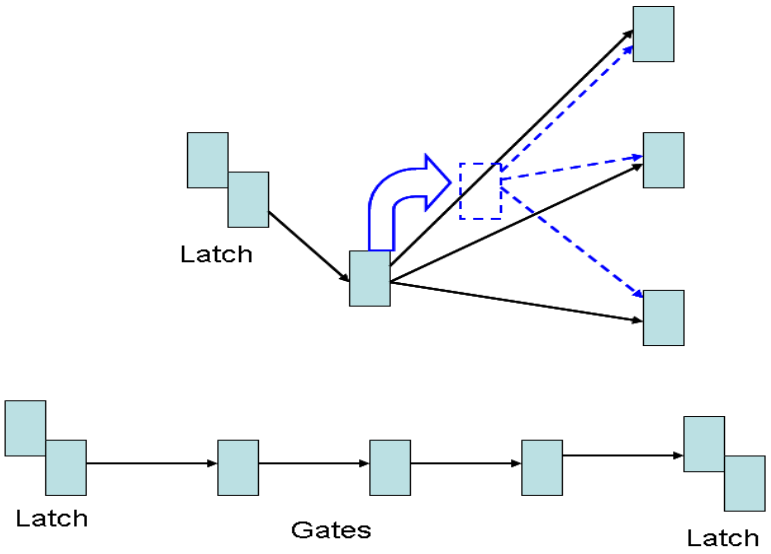


Fig. 2.8: Gate placement optimization.

- blockage_file: Specify areas where gate placement is not allowed.

Traditionally, optimizing the wire length has been the key objective during global placement, with routability being addressed via post-placement refinement techniques. There have been attempts to alleviate congestion via routability-driven global placement, but they rely on rudimentary, and most often, inaccurate techniques like pin-density or probabilistic congestion estimation [27-30].

In initial placement optimization, netweights (assign number to a net to define criticality) and attractions (define which gates should be closed to each other) may be applied and are not meant to be extensive. At this time, no placement legalization is performed to save run time. Placement optimization is done with netweights and attraction based on the timing of the nets. Examples of netweights and attractions concepts are shown in Fig. 2.9. Nets with higher weight will be given higher priority to place the associated gates closer to the source gate to close the timing for critical nets. There are several options for attraction method that can be used in

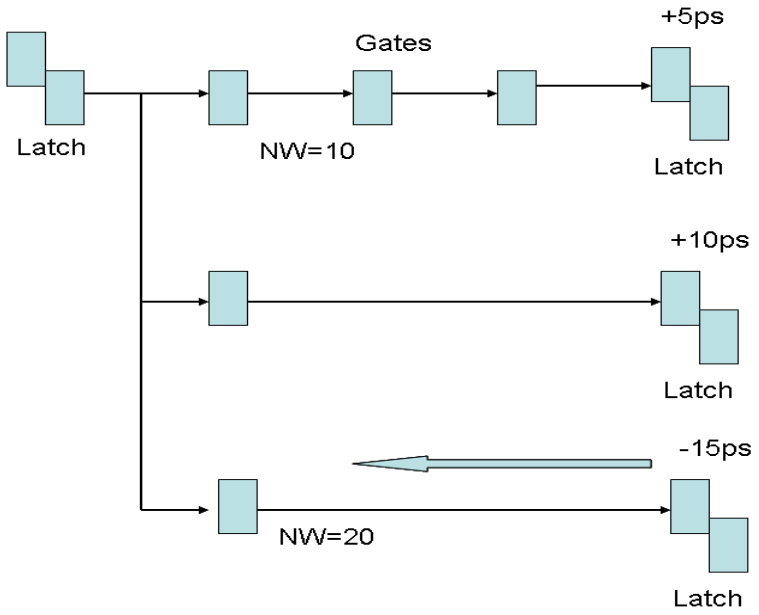


Fig. 2.9: Slack optimization.

synthesis to solve the timing failure. For example, synthesis can find out the worst slack for a sink gate and can apply attraction to the instance name or pin name of the sink. Similarly, net weight can be used on PI/PO of a gate to place them close to each other. User can control these placements by using a parameter in run directory. Examples:

net_weight_file – Can be used to give user-driven net weight file to place gates closer

attractions_file - Can be used to specify user-driven attractions file

c) Latch and LCB Placement: Local Clock Buffer (LCB) is used to create master-slave and scan clocking for the latches, where LCB is connected to a main mesh clock, along with other clock control signals. These control signals are being used to either shape-up clock pulse or mode of operation. In any design, it is common that latches could be driven by multiple clock domains and clock gating functions. And thus, synthesis must acknowledge physically aware clustering of latches and clock domains, where, latches cluster around the driving LCB to meet

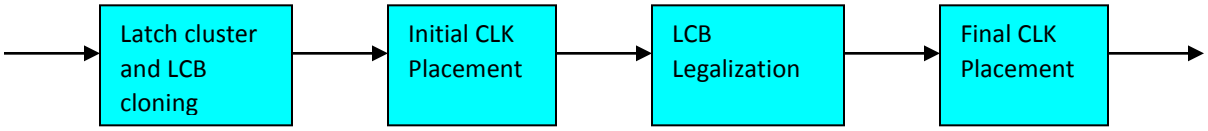


Fig. 2.10: Latch and LCB placement flow.

the tight clock skew guidelines of the projects. However, the placements of these latches and LCBs are very iterative placement process. Clock placement and routing optimization steps can be pictured in Fig. 2.10.

The latch and LCB placement algorithm starts with one LCB per clock domain, followed by cloning and power-up of LCBs with optimal latch connection to meet clock loading, delay and other design spec. An example of LCB and latch placement with several clock-gating domains is shown in the following picture, Fig. 2.11, where each color represents a different clock domain. Once synthesis goes through the initial LCB placement, latch clustering, LCB legalization and final clock placement, the above design ends-up looking like what is shown in Fig. 2.12.

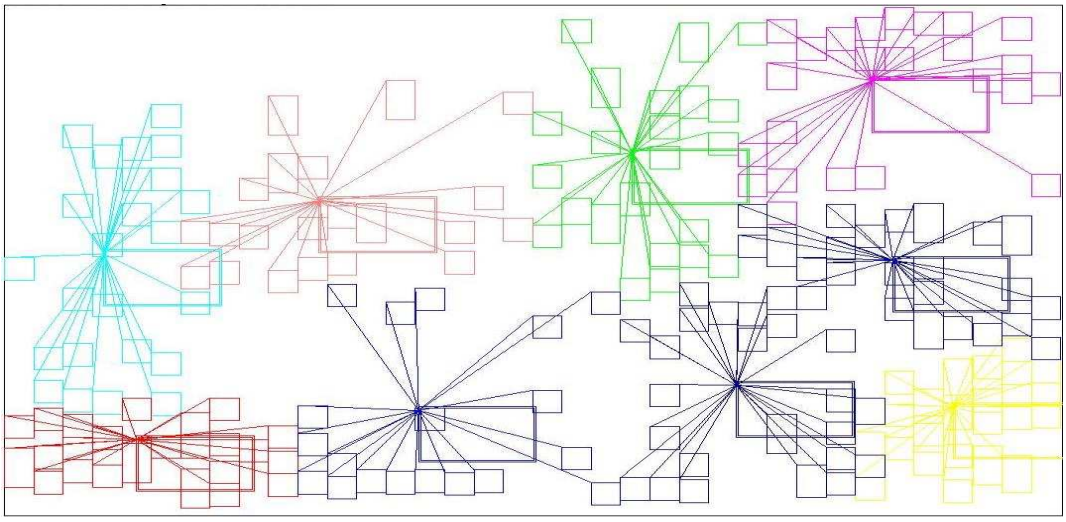


Fig. 2.11: Initial latch and LCB placement (Source: EDA, IBM).

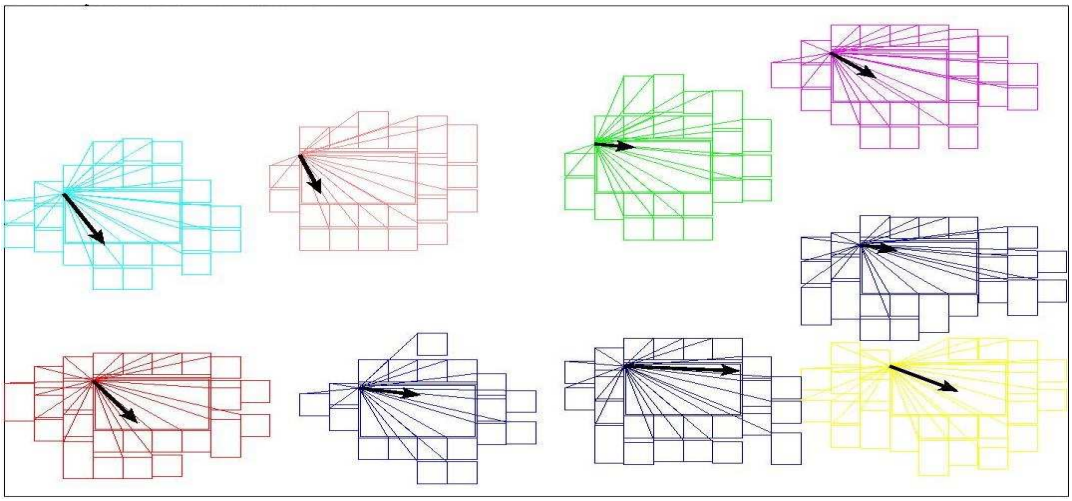


Fig. 2.12: Final latch and LCB placement (Source: EDA, IBM).

d) Design Violation Clean Up: At this stage, PDS will look at the connectivity and correct any design violation. These violations typically involve gate topologies, fan-in/fan-out restrictions, restriction on latch-input and LCB, etc., defined by the methodology of the project. For example: 1) Driver to the latch input cannot be driven by illegal gates, 2) Driver to the latch input cannot have more than one sink i.e. only sink is latch and 3) Macro output cannot have a feedback loop i.e. cannot have internal logic to drive. At the end of this phase of the design, the tool would re-do netweights and attraction steps to incorporate additional logic to place.

e) Final Placement: At this step, all the critical pieces of logic are very much locked-up. These logic gates include LCB, Latches, driving gates of latches etc. Similarly, as explained in the previous section, net-weight/attractions are also locked down. At this point and forward, physical placement refinement is done to make timing and power spec of the design. Refinement will perform physically aware optimizations but will not drastically change the placement solution. Part of this refinement is to correct electrical and timing corrections. The tool would

swap a gate with bigger gates and all sub-sequent gates to fix the slew or slack violation. Similarly, slower gates will be replaced by faster gates to meet the timing. The starting point of this stage is a placed netlist with a clocking solution locked down. In short, refinement will perform physically aware optimizations but will not drastically change the placement solution, i.e., minor moves/legalizations will occur.

2.3.3 Latch Clustering and LCB Cloning

To meet the timing, loads on critical logic are separated by cloning both latch and logic gates. A sample of such cloning is illustrated in the following example, where latches are coded to drive different output logic, when input is the same for both latches. However, synthesis tools need to be enabled with parameter so that PDS can clone latches, and similarly, equivalent latches must be defined in VHDL (or Verilog) before going to the synthesis process. Parameters must be enabled via synthesis parameter. A functional latch is described in details in Section 4.2 of Chapter 4. Following are the examples, Fig. 2.13, of two equivalent latches (cloned) in the VHDL, where one input net is connected to both latches but those latches are driving different outputs.

```
pds_use_cloned_latches: true                    #parameter in constraint file

    unit_control: entity latches.nlat

    port map    (c1             => master_clk,

                 c2             => slave_clk,

                 scan_clk       => scan_clk,

                 scan_in        => funcscan_in(0),
```

```
                scan_out        => func_scan_out(0),
                data_in         => unit_data_in(0),
                data_out        => unit_data_out(0))
unit_control_colone: entity latches.nlat
port map        (c1             => master_clk,
                c2              => slave_clk,
                scan_clk        => scan_clk,
                scan_in         => func_scan_out(0),
                scan_out        => func_scan_out(1),
                data_in                 => unit_data_in(0),
                data_out        => unit_data_out(1),
                )
```

Fig. 2.13: Latch clustering and LCB cloning.

2.3.4 Control Logic Structure

To control synthesis flow, sometimes designers can hard code the logic gates with appropriate logic style and drive-strength with some attributes in VHDL. There are some advantages and disadvantages for such coding because in the event of hard coding, synthesis will not optimize gates to meet the slew/slack requirements of the design. Similarly, poor coding or lack of understanding of the optimization can cause sub-optimal design. This coding is used as a

hint to the tool, not a requirement. Designers could use “Logic Style Direct” or “No modification” in VHDL source. Here are some examples of VHDL Logic Style Direct & NO_MODIFICATION.

ATTRIBUTE BLOCK_DATA of example_dir_mod0: label IS "LOGIC_STYLE=/DIRECT/";

ATTRIBUTE NO_MODIFICATION of example_dir_mod0: signal IS "TRUE";

stall_blk1a0: example_dir_mod0 <= NOT (main_eec OR main_stall);

stall_blk1b0: example_dir_mod1 <= NOT (main_stall OR ecc_data);

Like “Logic Style Direct” or “No_modification”, synthesis understand the “structure preserving logic” initiated by the designers. Advantage of structure preserve is: consistent logic and synthesis determine the drive strength. However, the designers need to include realistic logic in the design. Designers can also preserve the output drivers as coded in VHDL. The following is an example, Fig. 2.14, of how to hardcode a NAND gate in VHDL with desired drive. These gates belong to synthesis physical design library.

```
use_cw_nand2_rd0_l: entity stdcell.cw_nand2
    PORT MAP ( vdd => vdd,          #power
               vss => gnd,          #ground
               a => nand_input1,    #Input 1
               b => nand_input2,    #Input2
               y => nand_output     #Output );
```

Fig. 2.14: Logic structure control.

2.3.5 Pre-Placing Logic

Synthesis does very good job working on complex, Boolean equations of non-structured control type logic macro. To save area and close timing, sometimes synthesis needs "hand holding" i.e. placement information, for the timing critical and data flow oriented logic. In addition to hard IP and custom components, synthesis most often needs placement and logic topology information for critical logic, such as encoder, decoder, mux select etc. to converge to the desired solution. This pre-placement is generally a one-time job that requires understanding of the data flow of the unit. A sample of placement file that is being called during synthesis is given in following example:

begin_place

place <inst_name> xloc yloc <rot> movetype=fixed

end_place

where, <inst_name>: name of instance in VHDL;

xloc, yloc: x-coordinate, y-coordinate of the instance in floor-plan;

<rot>: rotation of the instance in floor-plan;

movetype: instance can be moved in synthesis or not.

2.4 Review of Quality of Data

There is a common saying in the synthesis world: "Garbage in, garbage out". Synthesis engines heavily depends on the quality of the inputs that it gets and optimizes logic structure, and places them to close timing. Otherwise, optimization tools will spend time unnecessarily. These essential inputs for synthesis tools are discussed here. Good quality assertions with PIS, POS and

ETA are essential otherwise synthesis will work on the wrong paths. Synthesis has a direct impact on the quality of the logic that is entered. If the slack is really negative, either assertions are way off, or the logic still needs major work. Designers should not have too much logic between latches or make sure that not too much logic fighting for the same physical area or IOs are not congested to access. Otherwise, this may lead to local congestion.

Macro pin placement should be done from unit prospective to avoid extra timing penalty and congestion. Also, input pins feeding a critical cone should be placed together, so that extra buffering is avoided. Duplication of PINs should be considered for large fanout nets. Similarly, PO pins fed by common critical cone(s) should have close placement. Sometimes designers can use macro internal pins instead of edge-pin so that unit can use better performing wires to connect input and output of the logic gates.

2.5 Synthesis Output

Each step of synthesis produces a different type of output based on the tool that is being used. In this dissertation, experiments are done in IBM's tool flow and thus some of the steps are discussed as follows:

Technology Mapped Netlist: The first step that produces technology mapped netlist from RTL. At this step, no optimization and no placement is done. The tool basically maps logical gates into available physical design gates in the library as discussed in Section 2.3.2.

Initial Optimization: At this step, the tool still has a technology-mapped netlist with very little placement and some optimization. This optimization is based on wire length and placement of gates. Buffer re-powering, removal, and re-placing are done at this step. Net weight and attraction are being done.

Final Optimization: This placement is after clock-optimization and has very little change in placement of the gates. Timing, electrical and power optimizations are done at this stage. Legalization of placement is done in Final Optimization.

Refine Steps: It is the last step in synthesis and the tool tries to deliver optimized design with the least electrical violation and optimized power. Major optimizations include:

- Electrical violation correction
- Efficient fanout trees
- Repowering, buffering, logic restructuring, pin swapping, cloning etc.
- Vt recovery (Remove leaky gates for better power design)

2.5.1 Major Output Files

Log Files: This file has all the run details including errors and warnings while the job runs. This is a very helpful file to debug any problem that may occur during synthesis steps.

History: This file keep track of summary of data for all runs in same directory. These data include: #of latches/LCB, macro utilization (logic gate density in macro), latch to latch slacks, run time, type of gate (logic gates with different kind of threshold voltage for leakage power) usages, CPU usages, total logic gate width in the design etc.

Plots*:* A routing and timing closed macro is the end goal for the designers. The provision of writing out the macro's internal logic gate and IP placement, vertical and horizontal wiring congestion, slack distributions are plotted for designers to give an overall picture of the design. Based on the plots, designers can make improvements to the quality of the design by mitigating noise and timing violations of the design using available methodology in synthesis.

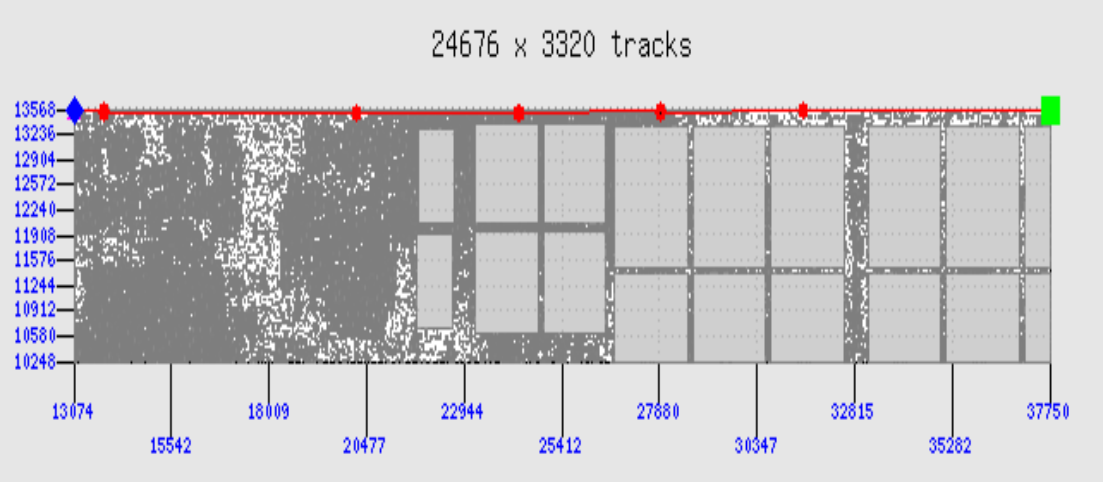


Fig. 2.15: Example of worse slack plot in synthesis.

Distributions of Gate Array (GA) cell: GA cells are used to fix bugs when rest of the design is frozen. In synthesis, these information is written out as plots and thus gives a good pictorial view to designers so that designers could take a look and decide which gates to use to fix functional and timing bugs. An example of such a plot, worst slack, is presented here in Fig. 2.15, where the source and sink latch are colored in blue and green respectively. Designer can look at the placement of these part of the logic and consider several options, as discussed in Section 2.3 to close timing on this macro.

Timing Reports: Tool writes out several timing reports (also known as endpoint report) for user to look at. These includes as follows:

- Comprehensive timing report: This comprehensive timing file consists of timing information on all nets and gates in the design for all timing phases.
- Endpoint Report: This output file of synthesis is shorter and user friendly version of comprehensive report. It includes both latch to latch and primary input/output paths containing the delay information for the gates and wires on the paths.

- Latch to Latch Endpoint Report: Out of all files that synthesis writes out, this file is the most useful to the designers. It allows designer to focus on macro internal latch to latch paths while the PI/PO paths are still in flux at the primary stage of the project. Matter of fact it is very typical that at the early stage of project, designer would relax the PI/PO paths so that internal paths get higher attention to fix the timing violations. Similarly, designers also look at another latch to latch endpoint file with "zero wire length" where there is no wire delay in the design and thus one can quickly determine whether this logic paths can be closed in synthesis or not or need logic fixes. An example of Endpoint file is shown in Fig. 2.16, where gate, wire delay and rise/fall time is broken down per logic gate.

Phase Name	Rise/ Fall(ps)	Slack (ps)	Delay (ps)	Arr. Time (ps)	Slew (ps)	Load (ff)	Sink Gate
Setup MA-	F	250.00	0.00	47.00	0.00		LCB
MA@L	R	79.89	32.15	210.70	48.05	1.39	LATCH
MA@L	R	79.89	6.07W	204.63	46.54	2.00	INV
MA@L	F	79.89	22.85	181.78	11.68	0.80	INV
MA@L	F	79.89	0.27W	181.51	11.58	1.92	NAND2
MA@L	R	79.89	9.60	171.91	17.58	0.80	NAND2
MA@L	R	79.89	0.05W	171.86	17.56	0.88	XOR2
MA@L	F	79.89	15.11	156.75	21.72	5.93	XOR2
MA@L	F	79.89	3.29W	153.46	19.68	17.62	INV
MA@L	R	79.89	12.54	140.92	27.29	3.87	INV
MA@L	R	79.89	6.83W	134.10	22.65	10.37	NAND2
MA@L	F	79.89	12.23	121.87	19.36	2.33	NAND2
MA@L	F	79.89	0.19W	121.68	19.31	2.54	NOR2
MA@L	F	79.89	29.40	92.28	56.12	0.45	NOR2
MA@L	F	79.89	23.34W	68.93	24.62	40.07	INV
MA@L	R	79.89	12.08	56.85	28.20	5.40	INV
MA@L	R	79.89	0.84W	56.02	27.99	5.99	NAND2
MA@L	F	79.89	15.37	40.65	11.55	1.00	NAND2
MA@L	F	79.89	0.05W	40.59	11.52	1.08	LAT
MA-	F	40.59	0.00	47.00	0.00		LAT

Slack: Cycle Time + LCB delay – path Delay = 250 + 40.59 -210 = 79.89ps

Fig. 2.16: Example of timing path in synthesis.

Netlist: It is the ultimate output from the synthesis tool that has all connectivity, placement and mapped to available physical gates of the physical design library. This text based netlist file is converted to layout cells once imported into cadence library and at this point physical designer can open the placed gates in GUI to analyze any timing fails and possible placement in the design with some parameter or customization. Another usage of this cadence based netlist view is: designer can edit this netlist for possible logic change (ECO: Engineering Change Order) because they can visually check the placement of other gates and come-up with better solution to optimize routing, metal usages and timing for the design.

2.6 Early Mode Padding (EMPAD)

Hold timing violations are the fast path violations, and so it is important that the timing represents the fastest environment in which the chip will operate (best case voltage, temp, etc.). EMPAD is a part of placement driven synthesis where optimization is needed to fix hold timing violation in the design. Hold violation is a violation when data arrives faster than the clock at the sink latch. These are the 'must' fix in the design and involve to slow down the fast paths. In Fig. 2.17, design has back to back latches connected mostly to stage logic so that correct balance of latency exists in the design. There is not enough logic gate between the left latch packs, where the set-up and hold slacks are shown as 45.0 and -8.0ps (hold violation) respectively.

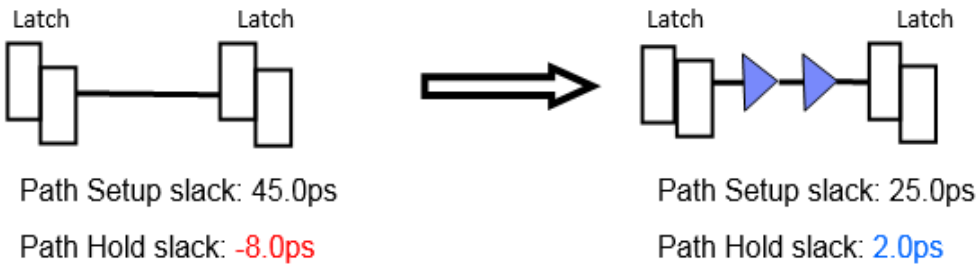


Fig. 2.17: Example of early mode padding (EMPAD).

In order to fix this hold violation, an inverter-pack (two gates with blue color) has been added on the logical path and thus the set-up and hold slacks became 25.0ps and 2.0ps, which satisfy both the late and early mode timing check for the design. Also, to fix EM failures, gates on the timing paths can be sized down to slow down the data arrival time. However, the designer(s) must be very careful in downsizing gates so that noise does not become a problem because smaller gates are more susceptible to noise. Similarly, it is also highly recommended to place EMPAD gates at the most downstream logic to minimize the usages of gates to save power and area.

2.7 Routing

Logic design team can avoid routing problems and achieve greater success by adopting a physical- and congestion aware synthesis methodology that reduces iterations between front- and back end tools, which positively impacts the die size and schedule time. Given the delay impact of wires in today's process geometries, modern timing-aware routers will try as much as possible to detour those routes that are non-timing critical, but this is not always possible. And given the prominence of physical wire delays in today's geometries, it is likely that a non-critical path becomes critical after its route is detoured. Further compounding the problem, physical tools tend to buffer those detoured routes in an attempt to speed them up, and this can create further congestion [31]. In the physical design stage, not only global routing, but some new placement algorithms begin to consider the congestion minimization as one of the most important objectives of optimization besides area and timing [32-38].

The routing tools involved in these steps are: power routing, clock routing and signal routing. However, as data prep, the router will read the synthesized data as netlist, where all the connectivity is already defined, along with macro-bounding box with wire allocation and

constraint files. In addition to the synthesized netlist, the router needs to know what layers of metals it can use to route the macro. Abstract of the macro has PI/PO placement information and cover cell has all blockage information that is needed during the routing steps. A constraints file is used as an aid to the router, where any specific such as a control file for power routing layers, is defined. A flow diagram for routing a synthesized macro with inputs is shown in Fig. 2.18, and sequence of the tool executions are shown in Fig. 2.19. Routing jobs in synthesis are mainly divided into two major steps: 1) Global Routing and 2) Detail Routing where following tasks are performed [39]:

1. **Global Routing**
    - The logic gate placements are preserved from synthesis
    - Identify routing resources to be used
    - Identify layers (and tracks) to be used
    - Assign particular net to these resources and
    - Also, used in floor-planning and placement
2. **Detail Routing**
    - The logic gate placements are preserved from synthesis
    - Define pin-to-pin connections
    - Must understand most or all design rules and congestion issues of the design
    - May use a compactor to optimize result
    - Necessary in all applications

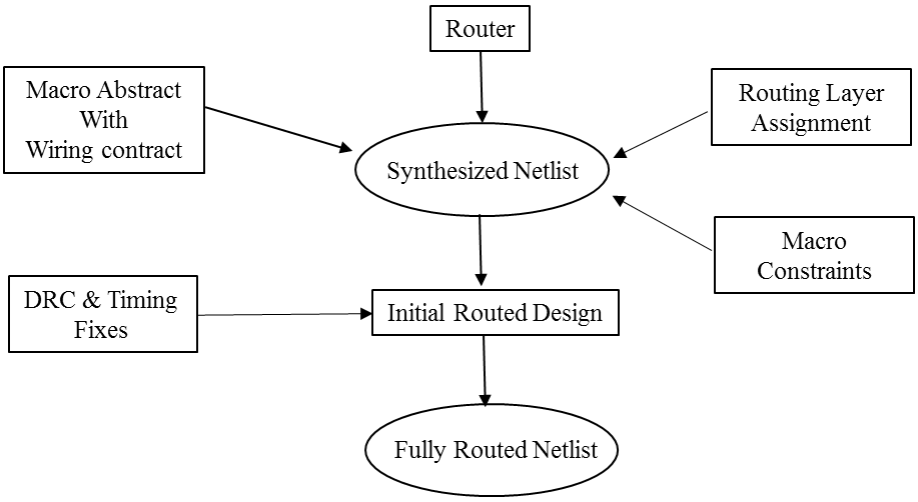


Fig. 2.18: Input and output of routing in synthesis.

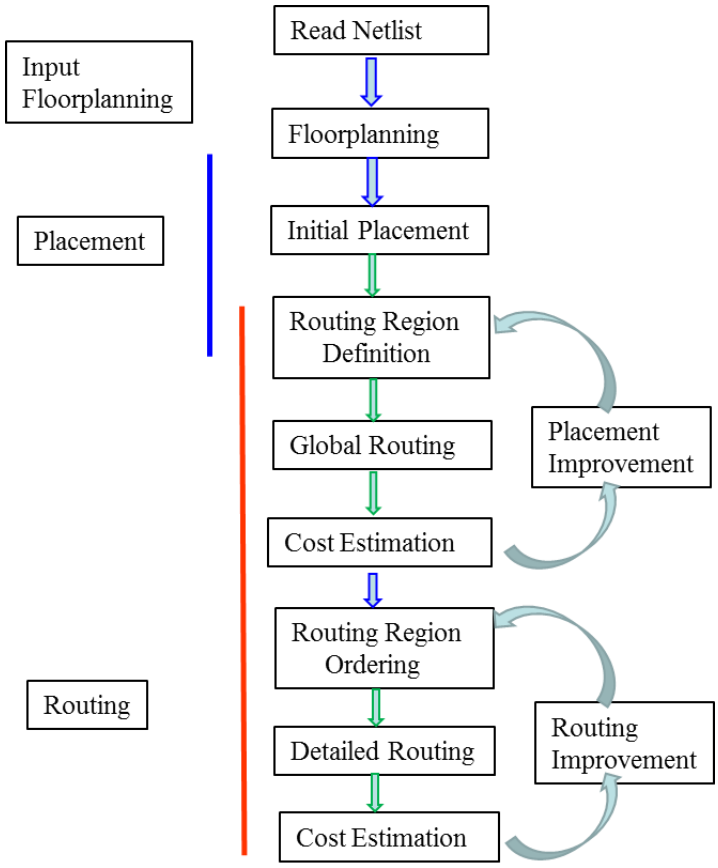


Fig. 2.19: Routing flow (Courtesy: IBM EDA).

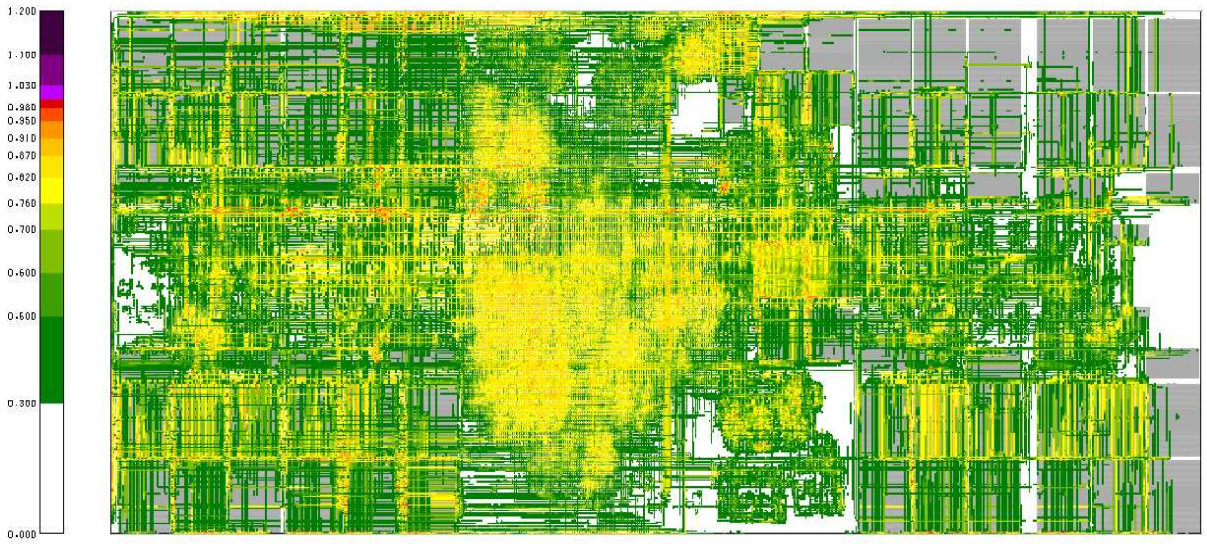


Fig. 2.20: Routed design.

To route any design in most effective way, it is very important to look at hot spots that design may have due to the congestion. The design may get routed but in the long run designers must be careful for back-end tool fails such as noise, electromigration, electrical rule fails, and timing fails due to noise. Designers should also think about having some extra space for future changes and bug fixes. There are some home-grown tools that can be used to find out hot spots in the routing that can led to space out logic gates or allow higher layers of metals if available. In Fig. 2.20, it is shown how congested the design is by taking a picture of metal routings and coloring red where design is highly congested. Yellow indicates caution, while green indicates no substantial problem to route.

2.8 Post Routing Optimizations

Once design is routed, another post processing tool can be used to fix the design violation left by the router in a minimally disturbing way. One can perform the optimization after detail routing to fix a post-routing timing "jump" [40]. Some usages of these tools are discussed here:

- Electrical Quality
  - Rise/Fall time fail fix
  - Max loading fix
- Timing Quality
  - Improve both late and early mode timing fails
- Improve power by replacing leaky gates with power efficient gates
- Remove unnecessary routings to fix some design rule violations

Post processing tools work on already routed designs for better optimization with incremental routings and placements to fix electrical and timing fails. However, it works on the design that is already placed and has detailed wiring done. It fixes scenic nets and tries to make it straight. Given a set of scenic nets, post-processing tool rips-up and re-routes these nets. Once re-routing is done, it is much straighter than the original routings as captured in Fig. 2.21. Table 2.4, on the other hand compares both early and late mode slacks in addition to slew and maximum cap fails before and after running the post-processing tools once the design is routed.

Table 2.4: Post processing tool to fix timing violation.

	Beginning	Ending
Late Mode Slack	-15	-5
Number of Negative Paths	200	88
Early mode Slack	+ive	+ive
Number of Negative Paths	0	0
Slope Violation	20	2
Load violations	15	0

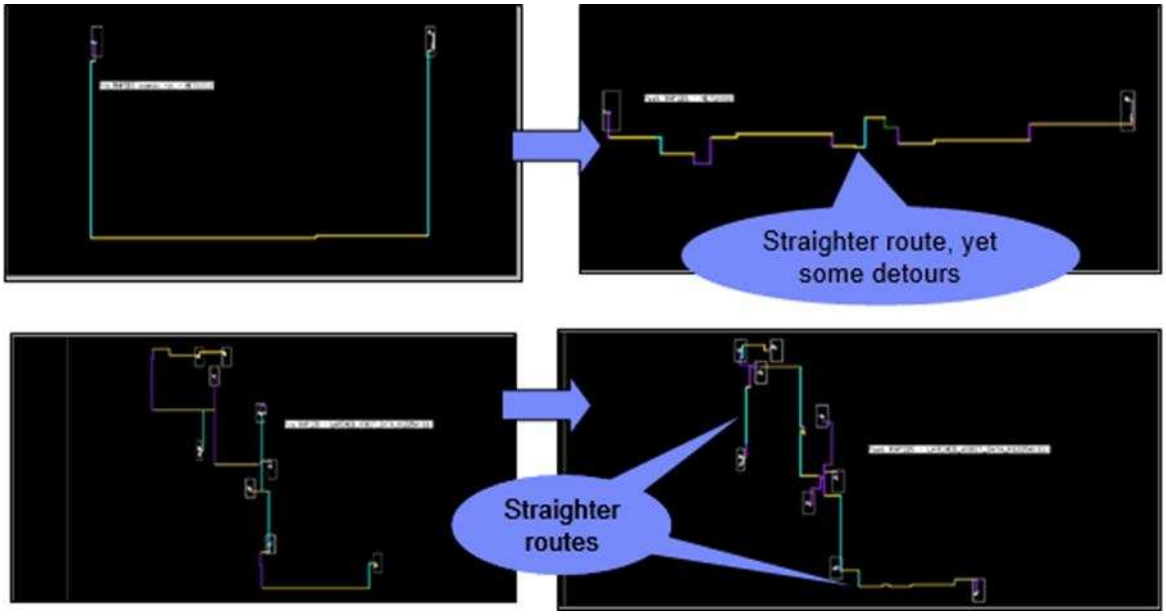


Fig. 2.21: Bad routing and fix after post-processing (Courtesy: IBM EDA).

2.9 Summary

In this chapter, industry standard synthesis flow and design methodologies are discussed. Today's synthesis engines do a very good job in closing timing, electrical and noise constraints for low-frequency and random logic macros. However, traditional synthesis flow does not do a good job for structured, high-frequency, multi-power, multi-clock and timing-critical multi-million logic gates which need to be packed in a tight floorplan. In an automated synthesis design methodology, these design constraints must be analyzed, understood and implemented in such a way that backend toolsets understand the timing and implementation checks, especially in supporting MMLGS to optimize and gain high level of design productivity to improve TAT.

Chapter 3

Drawbacks of the Existing Synthesis Flows

The synthesis methodology reviewed in Chapter 2 has been the key to success for decades, where latency goals were not too difficult to achieve due to aggressive technology scaling and the relaxed latency demands in ASICs. Therefore, it has been a key vehicle for physical design and implementation mostly in ASIC areas, where area and latency requirements are less important than other goals, such as time to market. Logic synthesis is recognized as an integral part of the design process, leading to an evolution in methodology from capture-and-simulate to describe-and-synthesize. The new methodology's advantage is that it allows us to describe a design in a purely behavioral form, de-void of implementation details, and then to synthesize the design structure with CAD tools. Designers can apply the describe-and-synthesize methodology on several levels of abstraction. On the gate level, they can synthesize functional and control unit logic by means of combinational logic synthesis. They also can synthesize controllers from finite-state machine diagrams by means of sequential synthesis. On register-transfer level (RTL), they can describe the behavior of ASICs with programs, algorithms, flow-charts, dataflow graphs, instruction sets or generalized FSMs in which each state performs arbitrarily complex computations. Then they can synthesize these ASICs by means of high-level (or behavioral) synthesis techniques [41].

Fig. 3.1 shows the current usage of synthesis based physical design, where most of the control macros are designed as smaller RLM or SUPER RLM, especially in a low frequency

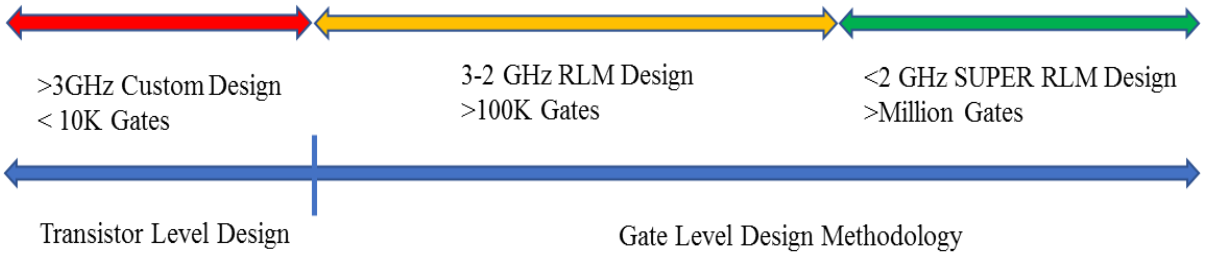


Fig. 3.1: Design methodology for synthesized macro.

domain and go through gate level design methodology. The logic gate counts in this synthesis based design is in the range of 1 to 2 million functional gates.

Intel's Ivytown [42-43] has 15 core designs with fairly good usages of MMLGS. As they reveal the product, here are the summary of their large block:

- Continued the trend for Multi-Million Logic Gate Synthesis
- 2 million synthesizable gates in static timing
- Enhanced synthesis with 2x design complexity and
- "Special" methodology for signal integrity and circuit quality.

Ivytown chip floor-plan with Intel's 15-core chip design [44] is captured in Fig. 3.2, where MMLGS usage is drawn in white dotted rectangle. However, compare to L2 cache unit for the experiment used in this dissertation, it is a far smaller and high latency unit, because the L2 cache unit has around 20 million synthesizable gates and 40% of the logic is on same frequency as the core and the other 60% is on 2:1 frequency to the core.

Multi-Power-Supply based design is a very popular way to save chip power, especially at the array design interface, where part of the input/output signal of the design are on higher power

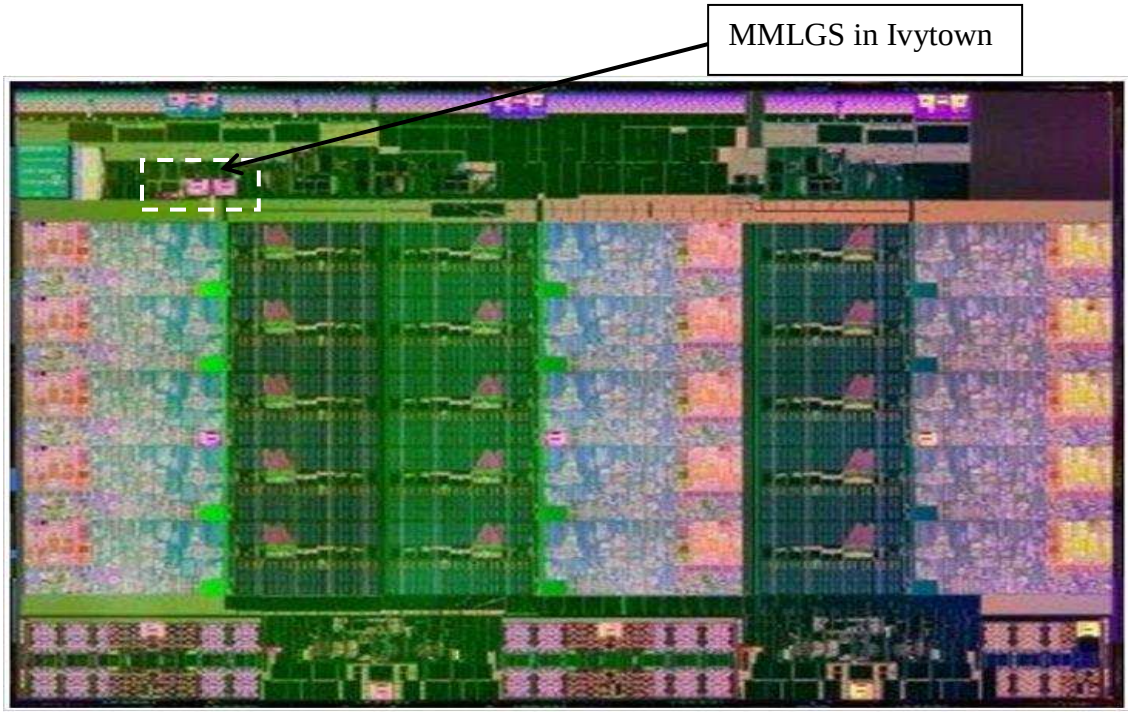


Fig. 3.2: MMLGS usages in the Industry (Intel's Ivytown Microprocessor).

supply than rest of the macro to maintain the desired performance. Similarly, multi-clock based design methodology: 1) enables certain part of the chip to function at different frequency to meet the performance requirement; and 2) reduces the power of the chip by dynamically shutting the clock down when circuits are not in use. When the synthesis methodology for MMLGS is considered, designers must consider the multi-voltage and multi-clocks design for functional and design productivity requirements and still satisfy the power-performance tradeoff for the design. The following subsections explain why current industry standard design methodology is inefficient in many ways and provide motivations for MMLGS and the necessity of employing multi-voltage and out-of-phase clocks in the design. These design methodologies are also discussed in details in chapter 5.

3.1 Physical Design Resource Limitation

Even though today's synthesis engines do a very good job in closing timing while meeting electrical and noise constraints, most often synthesis methodology is limited to random logic, i.e. it does not do a good job for structured logic that needed to be packed in a tight floor-plan. Even for control macros, designers spend quite bit of time to work on macro size, pin placement, routing assumptions, etc. Thus, to divide and conquer, units are broken into pieces to define the macro boundaries. Typically, a lead designer works with a group of designers and divides the work so that turnaround time is optimized. Since each designer works on their own piece of design, several designers, typically 6-8 engineers, work in parallel to each other to design and deliver the macros for unit integration and timing work. With increased complexity, the number of engineers needed for completing a functional unit increases dramatically with the traditional design flow.

3.2 Turn Around Time (TAT)

The basic advantage of synthesis over custom design is a very quick turnaround time. In a traditional physical design flow for both synthesized and custom units, logic and circuit designers along with unit integrator go over the macro portioning based on unit data flow. In a typical design flow a lead designer will then allocate macros within the team, spending quite bit of time analyzing macro boundary, floor-planning, metal usages, and work-load balance. The Physical Design (PD) team will then design macros, close time, clean all PD check violations and deliver those to the unit independent of each other. Sometimes, even though macro internal timing is closed, the primary input and output paths will be still broken and thus designers will need to exchange a good amount of physical design efforts for unit level work. For example: macro designers generally design macros independent of each other and thus may come-up with

unoptimized logic at PI/PO. If there is a timing fail between two macros at the unit, both macro owners must work together and optimize both designs in terms of logic-gates, driving-strength and wire-usages of the design. They may have to go through couple of iterations of physical design to get the it acceptable by the unit timing and integration. Thus, required TAT is much longer than the market dictates.

3.3 Timing Optimization

In smaller macro portioning, primary output drivers generally drive long wires, unit level buffer(s), and then input to the sink macros. Both source and sink macros are designed and built independent of each other, thus there might be inefficiencies at both macro boundaries. It is practically impossible to get rid of these inefficiencies and thus, a timing penalty is paid at these boundaries. Since these macros are physically separated and synthesis cannot see the logic at the same time as one macro, the timing engine cannot optimize the macro better. Fig. 3.3 is a simple example of unoptimized design where unnecessary inverters, logic and buffers can be avoided in MMLGS to improve the overall timing of the design. In real design, any number and types of gates can be optimized for better optimization of the design.

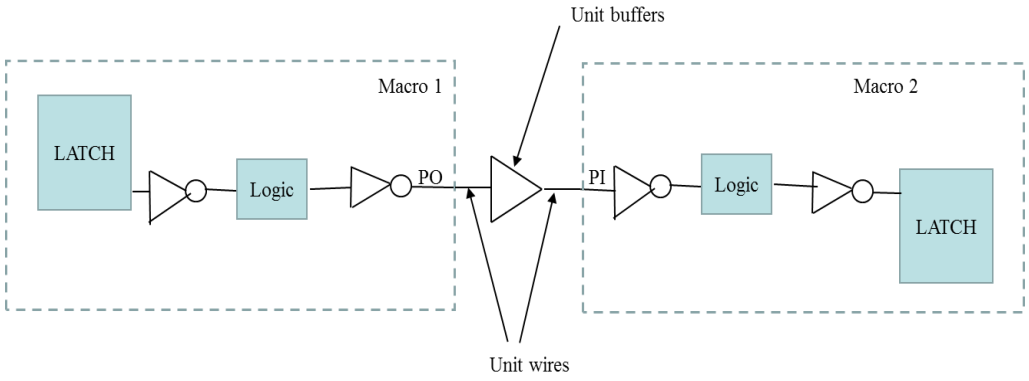


Fig. 3.3: Unoptimized macros at unit.

3.4 Power and Area Optimization

Both custom and synthesis flow at lower level macros are constrained by logic partition and timing windows. Each macro is tuned individually and may not necessarily be good for the unit. Most often, macros are designed and developed by a group of people and lack of understanding of each other's macro or timing constraint could lead to sub-optimal design. For example, unnecessary drive power or redundant logic at the macro boundary could end up wasting power and timing arc. Like timing optimization, area optimization is another important atask for synthesis. If physical gates are placed nearby, synthesis can eliminate extra buffering or redundant logic and does not need to make driving gates bigger, and thus can use smaller gates that have a ripple effect, as illustrated in Fig.3.3. One of the effects is area optimization. Thus, if the design has a conventional physical partition, there will be wastage of physical areas of the macro. However, if the macros are combined using proposed MMLGS methodology as sown in Fig. 3.4, design can be improved for timing, power and logic optimization compared to unoptimized design shown in Fig.3.3. Both Figs. 3.3 and 3.4 are functionally equivalent to each other.

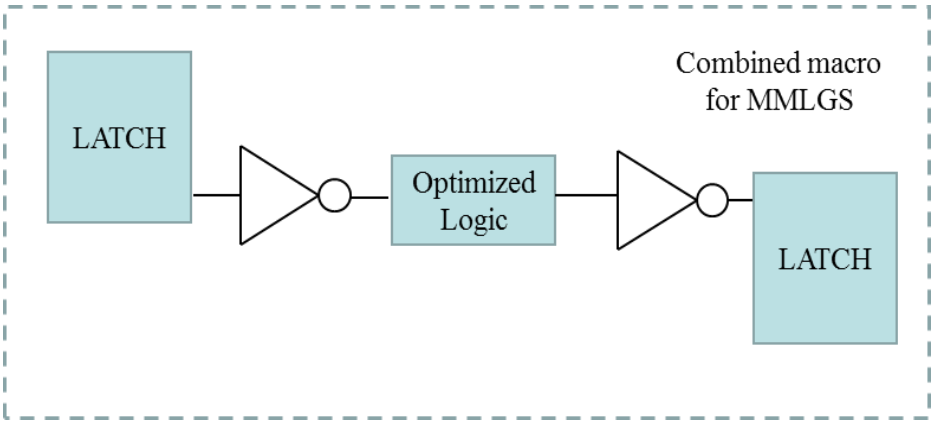


Fig. 3.4: Optimized macro with proposed MMLGS methodology.

3.5 Sync-Async Interface in Synthesis

To meet the design cycle time, designers use a technique call cycle stealing or slack sharing [45-46] i.e. instead of limiting design in one cycle, it is common to look ahead across multiple cycles and make sure timing can be met without changing the micro architecture of the macro. Fig. 3.5 shows an example where logic to the left clock (CLK) has a slack of -15ps, whereas logic to the right clock (CLK) has a slack of +20ps. Instead of looking one cycle at a time, if designers look at both cycles together then it can be found that there is a +5ps on the path and thus designers can delay the launching clock by 17ps, Fig. 3.6, to meet the timing for both cycles. It is worthwhile to note here that designers must pay attention to the early mode timing at these interfaces as well when slack stealing is in place.

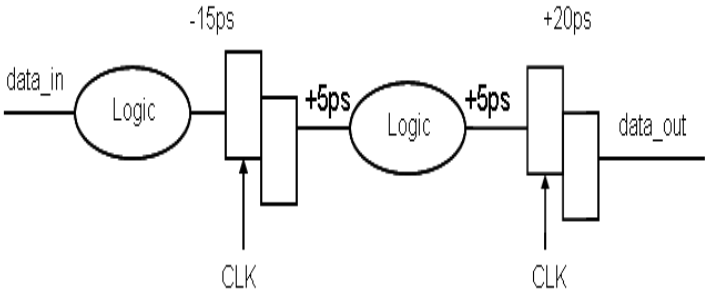


Fig. 3.5: Example of timing slack in 2 cycle.

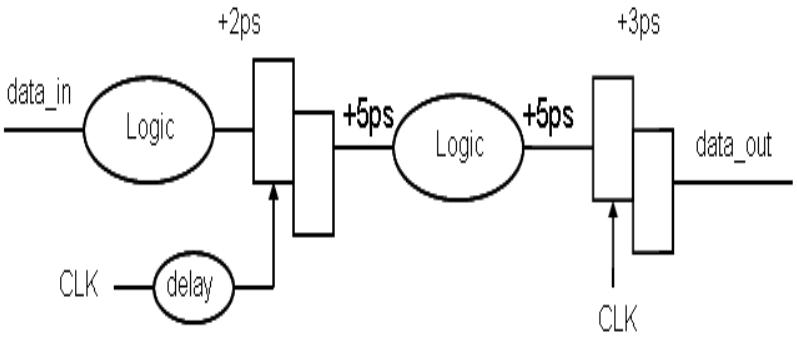


Fig. 3.6: Example of slack sharing.

Cycle stealing across cycles has been very effective and has been in use for years to relieve timing pressure in synchronous digital circuit design. However, if a timing path crosses out-of-phase multi-clock domains, slack cannot be shared between cycles due to the increased chance for metastability. In an automated synthesis design: i) a methodology is needed so that multi-clock domain interfaces are identified and designed correctly; ii) a timing methodology is needed so that slack sharing is not performed at sync-async interfaces during macro, unit or chip level timing; and iii) at the same time, the rest of the design can use slack sharing safely.

Available industry standard clock scheduling schemes [47-49] explain how to borrow slack across cycles for synchronous circuits. However, these methodologies do not offer a clear solution in automatic synthesis methodology especially when: i) the feedback path snips to the sink latch that is transparent during slack sharing; and ii) they do not explain how to address sync-async interface timing during slack sharing.

3.6 Multi-Power Domain in Synthesis

In a fully or semi-custom design flow, voltage translators are placed at the interface (input/output) logic that has gates on different power supply domain. However, in fully vanilla synthesis design methodology, generic toolsets do not support the physical placement and connectivity of the input and output logic that is on multi-voltage domain. Recent trends in IC design methodology shows big shift toward MMLGS methodology that most often requires embedded IP or custom component, such as arrays inside the macros. These embedded IP may require multi-voltage support and thus complicates generic synthesis flow and ends up partitioning and connecting logic gate(s) to wrong power rail for input and output gates. Therefore, in automated synthesis areas, these methodologies need to be analyzed, understood and implemented in such a way that backend toolsets understand the timing and implementation

checks. Supporting level translators for multi voltage domain in synthesis methodology, especially in MMLGS that has embedded array (IP), is a necessity for power efficient design in today's high speed high performing design technology.

In a full or semi-custom design flow, Level Translators (LT) are placed at the interfaces of the input/output (IO) logics that have gates on different supply domain. In synthesis, standard cells in a power domain may use only the primary power and ground, the power management cells like level shifter, isolation cells, and retention registers need a second power supply [50]. This second voltage domain complicates generic synthesis flow and end up connecting to wrong power rails for the input and output logic gates of LT unless properly attributed in VHDL and proper algorithm in synthesis methodology is followed. Both Synopsys and Cadence have offerings as Unified Power Format (UPF) and Common Power Format (CPF) respectively that are available in synthesis methodology [51-53]. However, study finds these CPF and UPF methodologies with very limited articles that are available to the public domain. Study shows that these available methods use power-ground constraints file [54-55] as an input to the synthesis flow. Thus, a methodology is needed which is an automatic enhancement to the synthesis engine, especially in MMLGS domain, along with HDL attributes and without any constraint file so that synthesis engines understand the power/ground attributes, placements and connectivity of the level translators.

3.7. Summary

Some of the limitations of synthesis based design methodology for high-speed, multi-clock and multi-power domains have been discussed in this chapter. As discussed in previous sections, available industry standard methodology is inefficient in timing, power and area optimizations and requires more PD resources. These problems can be resolved efficiently by

developing the synthesis-based design methodology for MMLGS, in both multi-clock and multi-power domain, as discussed in later chapter of this dissertation. The proposed techniques for MMLGS methodology can take advantage of breaking hierarchies for all non-array macros to optimize the logic gates and their placement to close highly timing challenged paths. The MMLGS methodology enables the downsizing of the PD resources for both custom and synthesizable macros and may eliminate the need for unit integration and timer resources in some cases.

Chapter 4

Circuit Design of Common Library for Research Experiment

In Chapter 2, industry standard synthesis methodology and flow have been discussed along with logic and physical design optimization. Both routing and post routing processes have been discussed there as well. In Chapter 3, the limitations of design methodologies have been discussed and foundations have been laid out for future development. Before developing new methodologies and algorithms in the following subsequent chapters, the basic building block of the physical design libraries are discussed in this chapter because several macros and units will be using these library components for the test cases used in the experiments.

4.1 Library Cells

To save chip area and power, efficient PD methodology can be used to choose desired size of the logic gates, which provides fine device size granularity. For this dissertation, IBM's RodRunner programable cell-based PD gate library is used where layout generation is automatic and physical design error free. For more details about RodRunner library, please refer to Appendix A1. In addition to the RodRunner library, a few other libraries for latches, Local Clock Buffer (LCB), buffers etc. [56] are also available for illustration purposes. Characterization and static timing analysis with a single supply for the entire chip can be done at a single performance point. The libraries are characterized for this point and the tools perform the analysis in a straight forward manner. With multiple blocks running at different voltages, and with libraries that may not be characterized at the exact voltage that being used, timing analysis becomes much more complex [57].

4.2 Latch Design

Scanable master-slave flip-flops (MSFF), Fig. 4.1, which operates in pulse mode, is used in the proposed design methodology for the test case. In normal mode, each of these MSFFs are controlled by two opposite phases, slightly skewed clocks, C1 and C2 that drive master latch (L1) and slave latch (L2), respectively. In normal mode operation scan_clk is held low to isolate scan data. To save the dynamic power, latch is operated in pulse mode, where C2 toggles and C1 is held high as shown in Fig. 4.2. The latch design methodology allows designers to operate the latch in several modes including delaying the mesh clock at the capturing latch which enables designers to steal time from next cycle. Local Clock Buffer (LCB) control signals can enable shifting the cycle boundaries and tuning frequencies in the lab for frequency limiting paths.

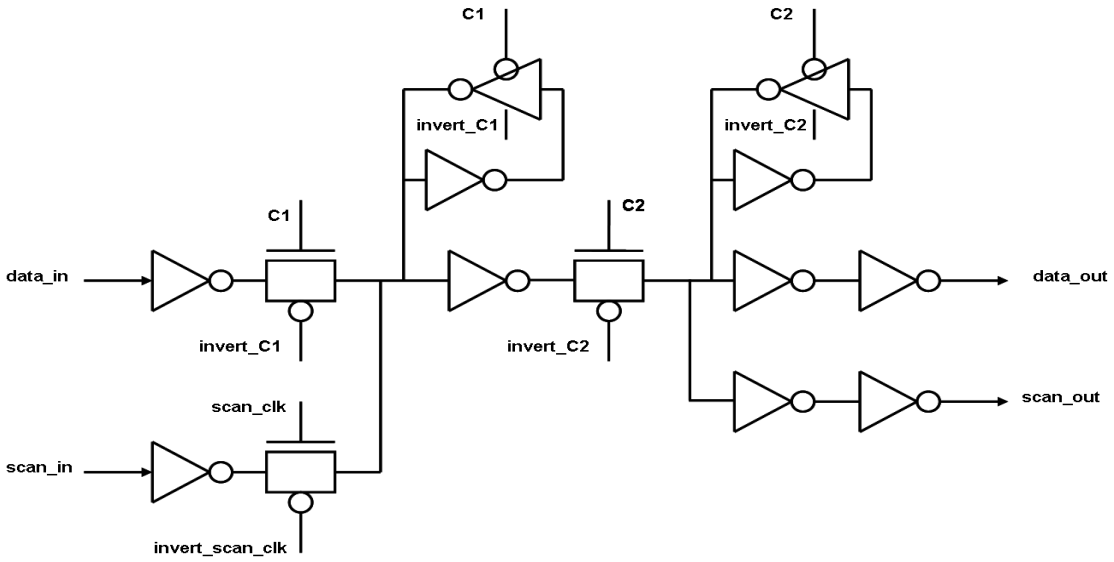


Fig. 4.1: Master-slave latch

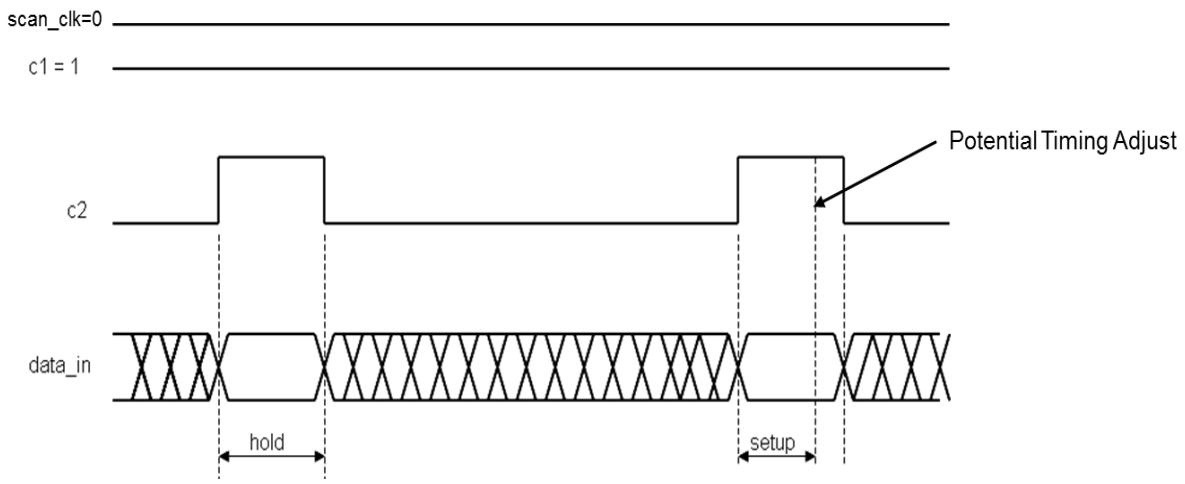


Fig. 4.2: Timing diagram of master-slave latch

Fig. 4.2 shows data set-up and hold requirements, where data is not allowed to change the state, in the timing window of clock being pulsed, in order to reliably transfer data from the input to the output of the latch. Fig. 4.2 also represents how the pulse mode timing is done in synthesis and unit/chip timing methodology enabling cycle stealing to give designers relief to close critical timing path at all levels of design hierarchies.

4.3 Array Design

Traditionally, SRAM arrays are designed using full custom approaches. Every transistor is tailored for its specific application. Typically, the effort is on the order of one to two Person Year (PY) per array. While ever-increasing chip functionality drives increases in the number of unique arrays per chip, pressure on cost, and thus resources, does not allow for the staffing of development teams to keep pace [57]. As a rule of thumb, development of a compilable system should be considered if the number of unique arrays exceeds five. The design point for arrays, the growable SRAM, is chosen to be an alternative for small-size register files for the L2DIR (macro for L2 Directory access) and L2LRU (macro for the “Least Recently Used” algorithm).

This SRAM cell allows independent read and write ports while offering higher area density of the design. The primary frequency target for L2 cache applications is typically half the core speed, but certain configurations of arrays also meet the cycle time requirements of the core and are used in high frequency clocks. A fully custom SRAM array is chosen for the L2 cache for better performance and area saving. The application space described above drives the number and type of options that are offered. To cover a good share of a typical register file use, the growable SRAM system offers one write port and up to two read ports. These ports are independent and can be hooked up to different clock domains. This flexibility also extends to voltage domains. A growable arrays can be used in chip areas where only a single supply voltage is distributed. In areas where Vcs (power supply for arrays) is present, a custom SRAM array can take advantage of that and can operate with increased performance. To increase yield on small processing nodes, the growable arrays system can build arrays that can be repaired post production if they are affected by manufacturing defects.

4.4 Sync-Async Latch Pack

A group of latches and corresponding logic which are always clocked by a common clock or clocks, have a common frequency and fixed phase relationship. On the other hand, two clock domains are operating asynchronously with respect to each other if their respective clocks do not have a fixed phase and frequency relationship. Asynchronous signals coming to a synchronous system must be synchronized with the rest of the system. This is usually done by feeding the asynchronous signal to a latch or flip-flop referred to as ‘Synchronizer’. If a clock edge from a synchronous circuit changes too close in time to data arriving from an asynchronous circuit, the circuit may enter a metastable state [58]. This can lead the system into an illegal or incorrect state as shown in Fig. 4.3, causing the system to fail. Such an event in SPICE has been created

and presented in Fig. 4.4, where the cycle- and set-up time to latch are 340ps and 30ps respectively with a stimulus for the input of MSFF, discussed in Chapter 4.2, that would change arrival time every 1 ps at the latch input. As the simulation suggest, the set-up time to the latch is found to be around 30ps and once the input starts to violate the set-up time, the latch starts to go into a metastable condition.

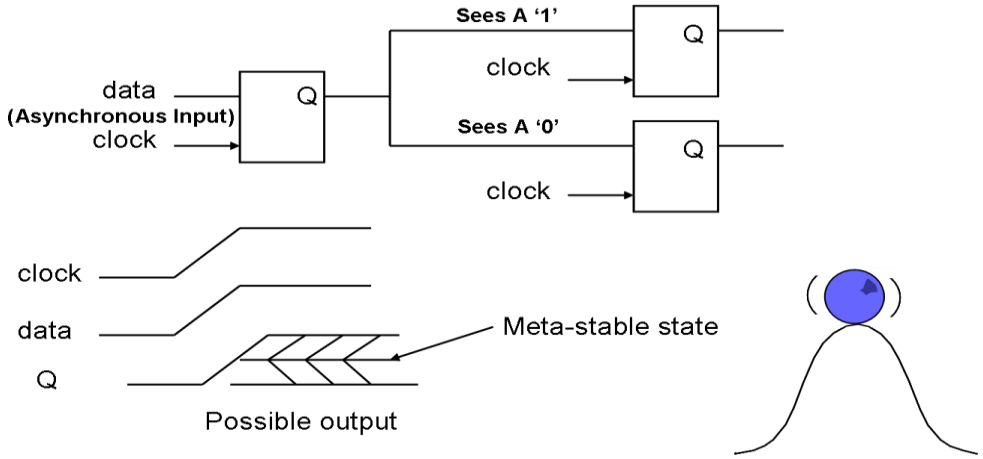


Fig. 4.3: Metastability example

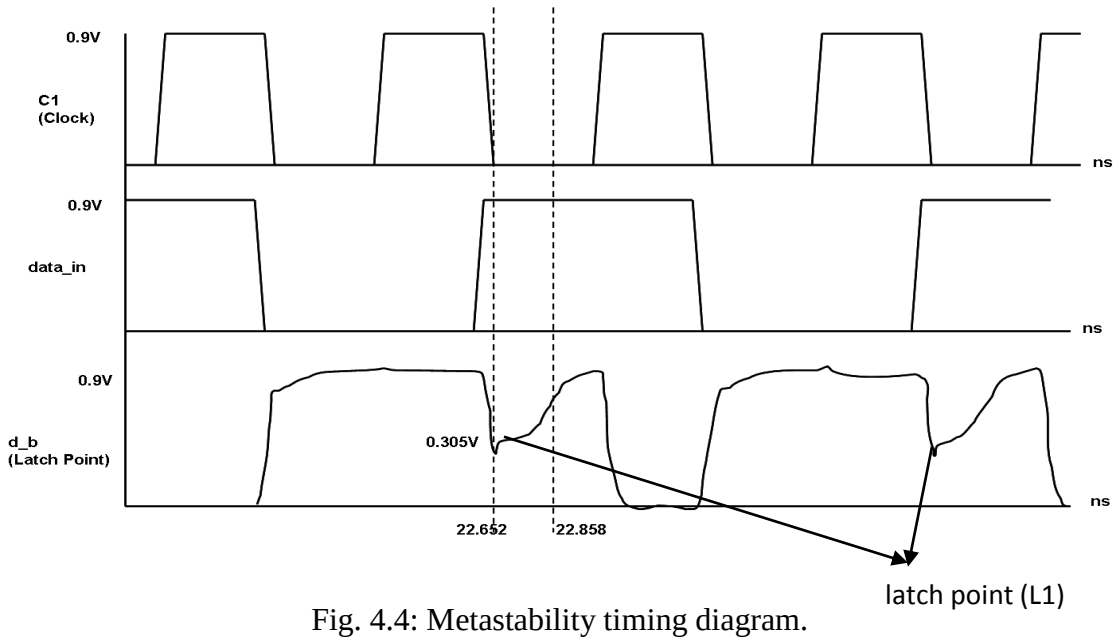


Fig. 4.4: Metastability timing diagram.

In Fig. 4.4, it is shown that the latch point (L1) is in metastable state i.e. unknown state for about 206ps. During this period of unknown state, any electronic circuit would mal-function and theoretically stays in unstable condition for an unknown period of time causing a permanent system hang. However, this circuit does come back to a known state in about 206ps because of process, voltage and temperature (PVT) changes in active silicon area around the latch point, which may not happen all the time.

To avoid metastability, and support multi-clock design, the simplest approach is to double-latch asynchronous signals being sampled by a synchronous module to create sync-async latch. In sync-async latch pack, the first latch is clocked by transmitting clock domain and the later latch is clocked by receiving clock domain as shown in Fig. 4.5. Here transmitting latch is clocked by "NCLK" clock and receiving sync-async latch pack is clocked by "NASYNC" clock. This latch pack is another standard common library cell which is designed in such a way that would reduce the chance of either latch going into metastable condition and may have stricter timing requirements between them to reduce chance of metastability propagating from one latch to the next.

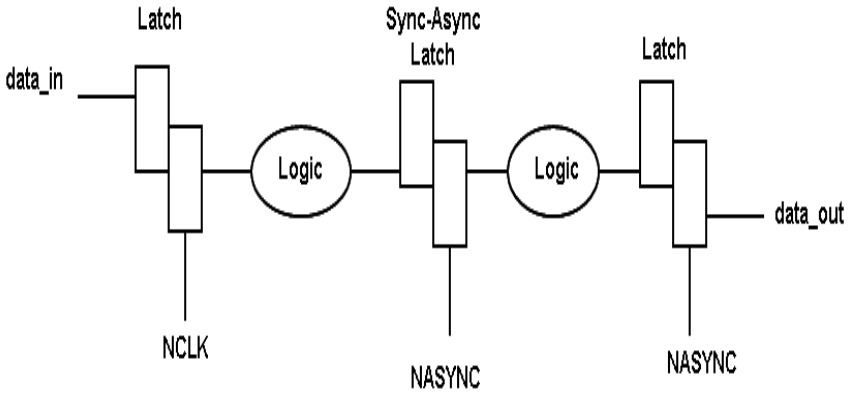


Fig. 4.5: Sync-async latch pack.

Chapter 5

Proposed MMLGS and Timing Methodology

5.1 Introduction

Fig. 5.1 is an example of a typical PD flow in high-speed and high-performing VLSI circuits, where each macro is individually built and timed to build a unit and finally the full chip. In a typical microprocessor, due to the area and timing pressure of the design, critical arrays and highly-structured-timing-critical-macros are designed using full and semi-custom approaches, where every transistor is tailored for its specific application. On the other hand, all random logic macros, to perform certain control functions, are built using automated tool flow such as synthesis. To meet the design constraints such as area, timing and power, data flow macros most often need hand-crafted schematics and layout design. The custom design approach requires a

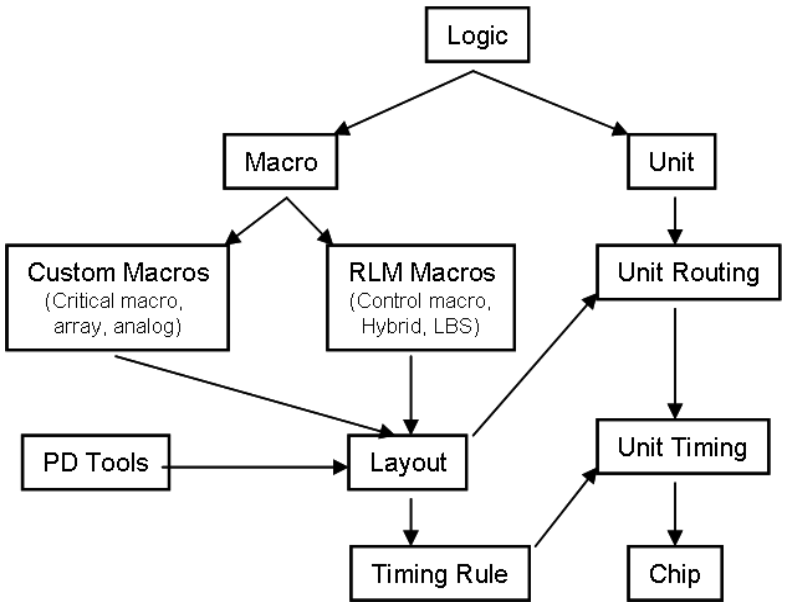


Fig. 5.1: High level physical design flow in VLSI circuits.

good amount of hand holding and manual intervention and thus requires as much as four times the development time of the synthesis flow. In general, a custom designer must plan every detail, including physical placement of the design, up front. The problem with a custom design approach is incorporating any late design change may occasionally cause a major rip-up of a previously built and timed macro and thus, become very expensive in terms of TAT.

Compared to custom macros, synthesis-based design is primarily for non-structured control logic and is automatic tool based with faster TAT. Late changes in the design can easily be incorporated in an automated way, keeping the rest of the design mostly unaffected. Today's synthesis engines do a very good job in closing design constraints for low-frequency and smaller random-logic macros. However, synthesis does not do a good job for structured, high-frequency and timing-critical logic that needs to be packed in a tight floorplan. In a typical synthesis flow, each smaller macro is individually tuned to make timing and PD rule checks clean for the unit level work, still requiring a fair number of PD resources for macro closures. This methodology still needs same unit integration and timing support as the custom macro design flow.

5.2 Multi-Million Logic Gate Synthesis (MMLGS) Methodology

Both the custom and the traditional synthesis flow at lower level macros are constrained by logic partitions and timing windows. In both design flows, each macro is tuned individually, which may not necessarily be a good solution for overall unit design. Most often, macros are designed and developed by a group of people who lack understanding of each other's macros or timing constraints, which can lead to a sub-optimal design. For example, redundant logic or unnecessary drive power at macro boundaries could end up missing timing arcs and wasting power as explained in chapter 3. Bug fixes and late changes in an individual macro design most often become very expensive in terms of turnaround time and physical design resources.

Multi-Million Logic Gate Synthesis, MMLGS in short, is the way to the future physical design for design in both high speed microprocessors and low speed ASIC design. Compared to traditional synthesis flow, the proposed techniques for MMLGS methodology can take advantage of breaking hierarchies for all non-array macros to optimize logic gates and their placement to close highly timing challenged paths. MMLGS methodology enables downsizing of the PD resources for both custom and synthesizable macros and eliminates the need for unit integration and timer resources.

As more and more functions are packed in a design and dealt with faster turnaround time, the synthesis methodology and tool sets are challenged with multi-millions of gates in multi-clock and multi-voltage areas. In the microprocessor design, the use of multi-clock, multi-voltage and slack sharing is very common in custom circuit design techniques. However, in an automated synthesis design methodology, these techniques should be analyzed, understood and implemented in such a way that backend toolsets understand the timing and implementation checks. Sharing slacks cannot be performed between out-of-phase cycles because the likelihood of metastability increases at the synchronous and asynchronous (sync-async) boundary. Similarly, a signal crossing multiple power-supply domains must be interfaced through a level translator (LT), which appropriately shifts the signal level. The requirements of the multi-voltage and sync-async interface should be taken into consideration during all levels of physical implementation and timing for the electronic circuits. The demand for multi-clock, multi-power in synthesis methodology is ever increasing, especially in multi-million gate designs, MMLGS, to support multi-million logic gates for design productivity. The MMLGS designs most often require embedded IP or custom components, which may require different power supply rail(s) and (or) sync-async clocking interface(s). Thus, the development of MMLGS methodology

should be fully supported for multi-power and multi-clock enabling and disabling slack sharing where appropriate in an automated way. The proposed MMLGS methodology shows around 50% design productivity improvement over traditional digital design methodology.

5.2.1 Enhancement to the Existing Synthesis for MMLGS

Placement-aware synthesis is an essential component to close routing and timing of the design. However, most often, designers will find that wiring and timing constraints alone are not enough to meet timing and area constraints for a design, especially in MMLGS methodology. Thus, the industry standard synthesis tools need to be enhanced with proposed modified flow as shown in Fig. 5.2, where several key methodologies and algorithms, such as soft-hierarchy, Multi-VT insertion, internal pin and pre-placing critical logic have been added. These key enhancements are discussed in details in the following sub-sections.

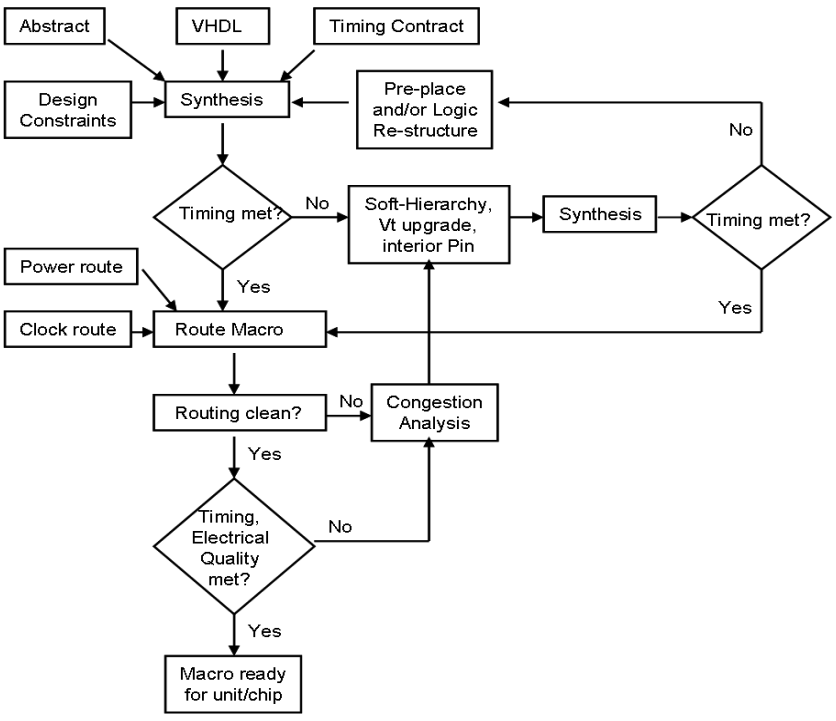


Fig. 5.2: Enhanced synthesis flow for MMLGS.

5.2.1.1 Soft-Hierarchy Methodology & Algorithm

Frequently, synthesis tools require guidance from the designers for relative placement of the gates. In this dissertation, a methodology called Soft Hierarchy (SH) is incorporated to guide the synthesis tools as a hint that logic within a soft hierarchy be placed relatively close to each other. SH is a method for physically synthesizing a design of an integrated circuit that includes compiling a logical description of the design into a flattened netlist, extracting a soft hierarchy from the flattened netlist, wherein the soft hierarchy defines a boundary on a die across which cells of the integrated circuit are permitted to move, and placing a cell of the integrated circuit on the die in accordance with the soft hierarchy [59]. In addition to the SH algorithm, user parameter can be used to guide synthesis for a user-specified placement of the logic gates that are involved in soft hierarchy. A pseudo algorithm explaining SH is developed and presented here in "Algorithm 5.1". In this algorithm, a synthesis tool traces logic gates forward and backward along the fan-out and fan-in nets to identify the gates that should be placed together.

Algorithm 5.1: pseudo code for soft-hierarchy during synthesis

```
1.  START
2:    read netlist of the macro and mark all gates unassigned
3:    group logic gates (n group) with Same Hierarchical Identifier (SHI)
4.     pick group #n and place gates with SHI into logic module(s)
5:       select first gate in logic module and mark as assigned
6:        trace forward along fan-out nets to get instance name(s) & place them in fan-out bucket
7:       trace backward along fan-in nets to get instance name(s) & place them in fan-in bucket
8.       Merge fan-out and fan-in one bucket with first assigned gate
9.       tag all gates in merged bucket as assigned
10.     select next group (n-1) and repeat 4-9
11.    if group=0 -> done assigning macros for soft hierarchy
12.    place gates from each group in same physical proximity
13.  END
```

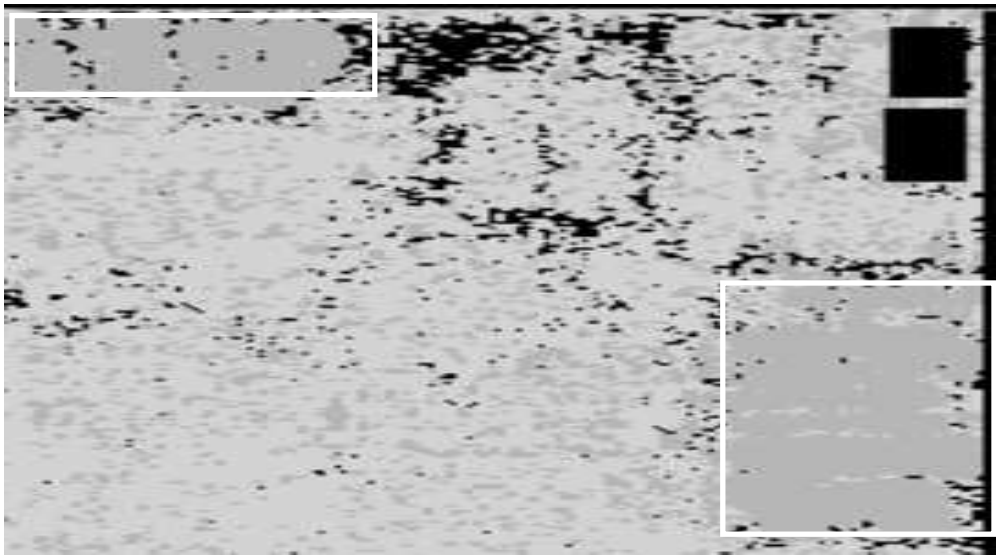


Fig. 5.3: Soft hierarchy example in synthesis flow.

Fig. 5.3 shows a classical example of the application of SH to meet timing, where logic gates in the two white boxes are placed apart but these gates are expected to be placed together. In a typical design flow, the designer would try out 1st pass synthesis without any pre-placement or SH to close timing. Once the job is done, and if timing is not met, the designer uses a GUI to highlight the placement of critical gates that are supposed to be together for better timing, power and congestion. If synthesis tools place them apart for any reason, designers can use simple tcl code, as shown in the example (5.1), to create a bounding box enabling "Algorithm 5.1" in synthesis so that gates in the two white boxes can be placed together. Designers do not need to wait for 1st pass synthesis if they have knowledge of logical placement of critical logic in design.

Example of soft-hierarchy parameter that can be used during synthesis flow:

name=<inst_name> prefix= <name> xlow=< > ylow=< > width= height= *(5.1)*

where,

<inst_name>: user specified name to recognize gates

prefix: is the name of logic gates used in HDL

xlow, ylow= left lower coordinates of the bound box for SH in micron

width, height: width and height of bounding box of SH in micron

Example:

name=rlctl prefix=l2rlctl xlow=50 ylow=20 width=30 height=30

5.2.1.2 Multi-VT Insertions & Algorithm

One of the goals is to deliver a timing-closed design within budgeted power that includes both leakage and switching power. As the timing-critical paths have been analyzed, it is found that most of these failed paths have too much logic between latches. To fix these paths, there are few choices i.e., add faster gates (Vt upgrade), re-architect the logic, delay the clock for capturing latch, upgrade the wire, and pre-place logic gates so that wiring is minimum. While these options are explored, synthesis parameter are also used so that synthesis can use faster gates if needed to close timing. These parameters enable the synthesis algorithm to upgrade *Vt* to a user-specified limit and override the project-specified limit. A pseudo algorithm "Algorithm 5.2" for upgrading *Vt*, has been developed and presented here. In this example, two levels of *Vt* have been defined for the simplicity of the algorithm. However, the same algorithm can be extended for more *Vt* types if allowed by the technology. Definition of gates for different *Vt* levels (As an example):

LVT: Lower threshold voltage. Very leaky device but faster

HVT: High threshold voltage. Least leaky device but slower

Algorithm 5.2: pseudo code to upgrade *Vt*

```
1. START
2.    identify all timing failed paths
3.    identify user's specified Vt distribution (user_vt) # Ex: user can have 20% LVT devices
4.    find current Vt distribution in the design (current_vt) # Ex: current LVT usage is 0%
                                                              # And all devices are on HVT
5.    pick-up 1st path and identify all gates on it
6.      while {current_vt < user_vt}{
7.         swap 1st complex gates to LVT
8.         timing closed? if yes, go to 16
9.         If no, swap Vt for next complex gate and go to 8
10.          If no more complex gate to swap then
11.             swap smallest inverter to LVT
12.              timing closed? if yes go to 16
13.              if no, go to 11 & repeat 11-12 for next sized inverter
14.                  timing closed if yes go to 16
15.                  if no, report current slack for the path and got to 16
16.    select next path and repeat 6-15
17.        if no more path to fix timing then go to 20
18.                                 }
19.     Exit no Vt upgrade available # Already reached limit
20. END
```

5.2.1.3 Internal Pin, Pre-Placing Critical Logic and Algorithm

Common design methodology is to use a bounding box for the macro i.e., abstract with input and output pins around the edges having appropriate metal usages to contact with the chip. In general, the macro gets a lower level of wires to route to the edge. However, synthesis can come up with a sub-optimal solution or a chain of buffers on the paths, which may create timing fails at the macro or unit boundary. The data of this experiment has been analyzed and it is found that boundary paths can easily be improved by guiding synthesis to place the timing-critical Primary Input (PI) and Primary Output (PO) pins right on top of the driving or receiving logic

gates. Although this methodology reduces a good portion of failed paths in the design, this can be viewed as a source of two potential problems. First, it can create a local congestion i.e., synthesis may end up placing critical logic in the same vicinity, and the router may end up fighting for routing channels. Second, the unit router must access the deeper part of the macro, and thus needs a higher layer of metal and can create a routing resource conflict.

A pseudo algorithm, "Algorithm 5.3", is presented here to explain how "interior pin" methodology is incorporated to close timing in this MMLGS test case.

Algorithm 5.3: pseudo code for interior pin

```
1. START
2.     PI/PO names for interior pin placement exists?
       # User provide a file with desired PI/PO for interior pin
3.     if yes {
4.             add lowest net weight value to PI/PO nets
5.             interior_pin=yes
6.            }
7.     do initial placement of the gates
8.     if interior_pin=yes {
9.       place PI/PO PIN over the receiving/driving gates
10.      Assign pins to coarse grid for routability
11.      Legalize pin location (fine grid, blockage aware)
12.                      }
13.   optimize gate placement to close timing
14.   optimize drive strengths
15.   continue with normal synthesis optimization flow
16. END
```

Synthesis does very good job working on complex, Boolean equations of non-structured control type logic macro. To save area and close timing, synthesis tools need "hand holding" i.e., placement information for timing-critical and data-flow oriented logic. In MMLGS, embedded

IPs and custom or array macros are pre-placed; otherwise, synthesis would simply not be able to complete the placement of the gates. Even in smaller MMLGS with 1 or 2 embedded IPs, pre-placing these IPs according to their data-flow is highly recommended for better quality of results. In addition to hard IPs and custom components, synthesis most often needs placement and logic topology information for critical logic such as encoder, decoder, and mux select to converge to a desired solution. This pre-placement is generally a one-time job that requires understanding the data flow of the macro. A sample placement file that is being called during synthesis is shown in Fig. 5.4.

begin_place

place <inst_name> xloc yloc <rot> movetype=fixed

end_place

where

<inst_name>: name of instance in VHDL

xloc: x-coordinate of the instance in floor-plan

yloc: y-coordinate of the instance in floor-plan

<rot>: rotation of the instance in floor-plan

movetype: instance can be moved in synthesis or not

Fig. 5.4: A sample placement file.

5.2.2 Proposed Multi-Power Design Methodology for MMLGS

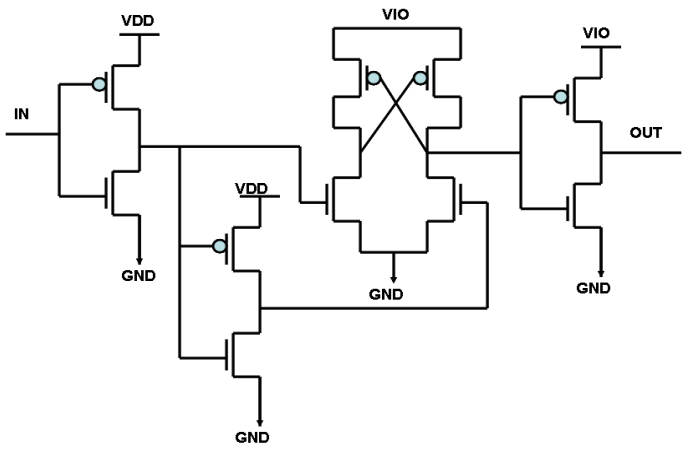


Fig. 5.5: Example of a generic “Level Translator” (LT).

Signals that are going to a high-power domain require a special circuitry called “level translator” (LT), to minimize static power loss or reliability of circuit operation because when a signal is propagating from low to high voltage domain, the downstream circuit could stay ON if the difference in power supply is greater than or equal to the threshold of the gate. There are number of ways LT can be designed. However, for methodology development, a generic version shown in Fig. 5.5 is used as an example, where input “IN” and output “OUT” are on “VDD” and “VIO” power supply respectively with common ground pin as “GND”. The top part of middle cross-couple logic is on VIO and lower part is on VDD power supply domain. In an automated design flow: 1) a tool is needed to check this power domain crossing and 2) make sure level translators have been instantiated with correct connectivity. In addition, the tool must understand not only the voltage translator function but also placement of the gate, timing and wiring of the path so that correct gates are connected to correct power supplies. However, the standard synthesis tool flow has the limitation in performing this basic function as explained in Fig. 5.6, where multiple voltage is used in the design.

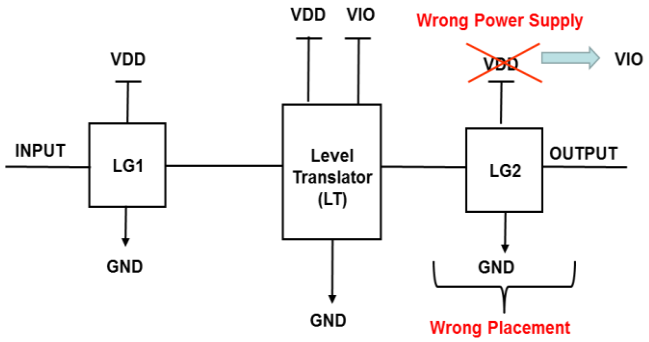


Fig. 5.6: “Level Translator” (LT) is expected to drive gate with “VIO”.

In Fig. 5.6, the circuit connectivity shows how a generic synthesis tool is unable to handle the connectivity and placement of multi-voltage gates in the design. In this example, the logic gate “LG1” with power supply VDD is driving LT that has a VDD and VIO power supply. LT is driving the logic gate “LG2” and thus “LG2” is expected to have VIO power supply; however, the generic synthesis tool incorrectly place “LG2” in VDD domain area and connects it to VDD power supply instead.

5.2.2.1 Parameters in Synthesis for Multi-Power Design

To help the synthesis tool to understand power supply nets, designers need to add proper “attributes” in VHDL so that synthesis can apply appropriate power names and constraints during placement, optimization and connectivity. Examples of such “attributes” are shown as follows in example (5.2) and (5.3) for power pins.

attribute pin_power_domain of <net_name> : signal is "vio"; (5.2)

attribute pin_power_domain of <net_name> : signal is "vdd"; (5.3)

Example:

attribute pin_power_domain of tc_pcie_wr_scan_diag_dc: signal is "vio";

attribute pin_power_domain of tc_pcie_wr_scan_dis_dc_b: signal is "vdd";

When it comes to placing an embedded IP, LT or any special circuits, the synthesis tool needs “hand holding” to get placement information. In order to properly place these level translator macros and IP, synthesis needs pre-placement for better quality of routings and timing. This pre-placement is generally a one-time job that requires understanding the pin placement and macro micro architecture. In this design experiment the level translator that is driven by “vdd” logic and output supply of level translator is on “vio”. This level translator is driving array logic which is also an embedded custom component (IP) with “vio” as power supply in this design. An example of placement file for LT or embedded IP that is being called during synthesis is given below in Fig. 5.7.

begin_place

place <inst_name> xloc yloc <rot> movetype=fixed

end_place

where

<inst_name>: name of instance in VHDL

xloc: x-coordinate of the instance in floor-plan

yloc: y-coordinate of the instance in floor-plan

<rot>: rotation of the instance in floor-plan

movetype: instance can be moved in synthesis or not

Example:

begin_place

place xlvlshifter_200 20664 10688 Y 180 movetype=fixed

place xlvlshifter_300 28152 8592 N 0 movetype=fixed

end_place

Fig. 5.7: Example of placement file for LT.

To compensate the delay due to addition of level translator (LT), router can use higher layers of metal with a cost function to meet timing as shown below in example (5.4):

<Flow>: <wire_code> <time gain> <routing layers> (5.4)

synthesis_layer_traits : W20S10L15 3 3 M2 X3

The layer directive constraint forces the timing critical nets route on the higher metal layers since the wires on the higher layers have smaller delay in advance technology nodes [60].

In example 5.4, W20S10L15 means double wide and single spaced M1-M5 metal layers can be used if tool finds a gain of 3ps on the net. The last argument indicates available routing layers.

Fig. 5.8 illustrates an example of the desired placement and connectivity of logic gates (LG) in mulit-power synthesis methodology. The LG and the embedded components are expected to be physically placed according to their power-supply attributes added in HDL. The LTs are expected to be placed and connected properly as the interface circuit between the signals crossing different power domains. As an example shown in Fig. 5.8, the placement regions have been divided mainly into two regions. The components on VDD power-supply are

placed on the left side and the components on VIO power-supply are placed on the right side of the macro. Based on the physical design requirements, this scheme can be easily extrapolated to multi-voltage design.

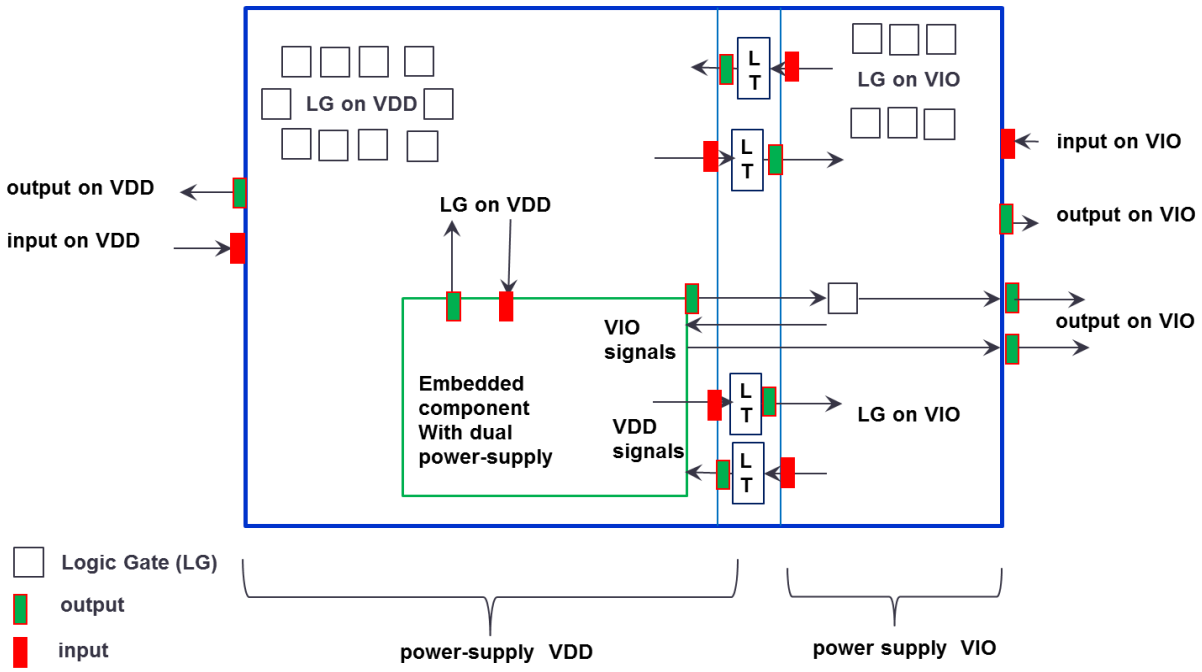


Fig. 5.8: Expected placement and connectivity of component in multi-power synthesis.

5.2.2.2 Proposed Algorithm for Multi-Power Synthesis

To recognize placement, timing constraints and appropriate connectivity for LT, including proper power and ground routings as described in Fig. 5.8 an algorithm is developed [61] for synthesis, presented as pseudo code in "**Algorithm 5.4**". This algorithm makes sure any signals which are referenced to the alternate power supply are hidden from buffer/gates which would be connected to the primary power supply as intended.

While the proposed "**Algorithm 5.4**" is used in design flow, Fig. 5.11, synthesis parameter also being incorporated so that synthesis can use faster gates, re-structure logic gates if

needed to close timing on these critical nets. This parameter enable synthesis algorithm to upgrade *Vt* (faster gates due to lower threshold voltage) to a user specified limit and override the project specified limit, with an expense of allowable static power in the circuit.

Algorithm 5.4: pseudo code to connect multi-power for Level Translator (LT), pre-place embedded IP or other component including LT if requested by user and apply wire trait if path is timing critical during automated synthesis methodology.

```
1: Start find_multi_power_domain                    ## This code is called during synthesis
2:      Define primary power supply   = vdd         ## Macro power supply
3:      Define secondary power supply= vio
                              ## embedded IP/Level Translator (LT)/logic gate with vio supply
4:      Primary Input = PI            ## Primary input to the macro
5:      Primary Output = PO           ## Primary output to the macro
6:      if pre placement file exist {
7:              pre-place all instances during synthesis as requested by the user
8:                                  }
9:      read macro VHDL {
10:             get power supply names: VHDL has attribute pin_power_domain
                                    ## should have both vdd and vio as power supply
11.             read all macro pins and get  attributes ## collect power attributes for synthesis
12.             read all embedded IP pins and get  attributes
                                    ## this includes level translators and embedded IP
13                      }
14:          if PI attribute = vdd {  ## Input on primary power supply, vdd
15:                     all downstream logic connected to PI inherit vdd as power supply
16:                             if PI or logic connects to embedded IP {
                                    ## IP=arrays or Level Translator (LT)
17:                                 check PI attribute with embedded IP's PIN attribute
18:                                   if both attributes match {hide secondary power vio}
19:                                    else {
20:                                        severe errors and exit from synthesis
21:                                        check logic for power connectivity
                                                ## formal verification tool (gyzer)
```

```
22:                                          }            ## checks power connectivity
23:                                                                  }
24:                             proceed with normal synthesis flow
                ## Synthesis will use power attribute to correctly place logic gates in VDD region
25:                             }
26:          else {    ## Input on secondary power supply, vio
                    all downstream logic connected to PI inherit vio as power supply
                             if PI or logic connects to embedded IP {
                                                  ## IP=arrays or Level Translator(LT)
29:                                  check PI attribute with embedded IP's PIN attribute
30:                                  if both attributes match {hide primary power supply vdd}
31:                                  else {
32:                                         severe errors and exit from synthesis
33:                                         check logic for proper power connectivity
34:                                       }
35:                                                                      }
36:                    proceed with normal synthesis flow}
                ## Synthesis will use power attribute to correctly place logic gates in VIO region
37:                 }
38:          if PO attribute = vdd {                  ## Output on primary power supply, vdd
39:                     check driving logic or IP's PO attributes
40:                             If both attributes match {hide secondary power vio}
41:                             else {
42:                                    Severe errors and exit from synthesis
43:                                    Check logic for proper connectivity
44:                                  }
45:                             proceed with normal synthesis flow
46:                                 }
47:          else {                                          ## Primary Output with vio supply
48:                    check driving logic or IP's PO attributes
49:                             If both attributes match {hide primary power supply vdd}
50:                             else {
51:                                    severe errors and exit from synthesis
52:                                    check logic for proper connectivity
53:                                  }
54:                             proceed with normal synthesis flow
```

```
55:               }
56:     read timing report of the macro
57:             calculate slack on the PI/PO nets of the LT
                                        ## To determine whether LT is on critical path or not
58:                 if {slack < 5ps} { ##5ps is an example here.
                                        ##It is really project default or user specified slack
                                        ##target on critical nets, which varies technology to
                                        ##technology and based on circuit design topology.
59:                     add this net to critical net bucket
                                        ##critical_net_bucket consists of timing critical nets
                                        ##which are generated during synthesis's timing run.
60:                      for each net in critical_net_buckets {
61:                         apply wire trait algorithm and parameter as requested by user
62:                             if still needed apply Vt upgrade if design is within power limit
63:                                                                 }
64:                                     }
65: end find_multi_power_domain
```

5.2.3 Proposed Timing Methodology for Sync-Async Interface

As explained in Chapter 4, to save power, latches operate in pulse mode, where only slave clock (C2) toggles and both master clock (C1) and scan clock (scan_clk) are tied high. In timing methodology, there are several timing methodologies and techniques available for industry leaders to use during all levels of chip hierarchy to take advantage of slack sharing. In this dissertation, IBM's existing methodology called "Rise Edge Adjust of Leading clock" or "REAL" in short for latch and LCB is expanded to develop the macro and unit/chip timing methodology for sync-async interface. In Section 4.3, it is explained how latches work in pulse mode with a timing diagram. In "Auto REAL Adjust" or "ARA" mode of timing, the leading edge of the clock virtually moves out to allow data to arrive late to give timing relief. With the application of ARA, overall slack can be improved, i.e. the data launch in next cycle gets delayed by same amount, which gives designers freedom to steal cycle by looking at two or more

consecutive cycles. Appropriate cutoffs are placed in the flow so that only a set-maximum-slack is borrowed from a downstream path, leaving a minimum positive slack in the logic.

ARA methodology works well in the areas of synchronous design. However, ARA cannot be done at: i) where there is an inverted feedback path to the latch as a state saving mechanism as shown in Fig. 5.9, and ii) at sync-async interface as shown in Fig. 5.10. For the feedback paths, in addition to turning off ARA, there must be a "hold" test in "late-mode" timing because late arrival "data_in" can overwrite the inverted feedback data. Thus, an algorithm is proposed here to add to the existing synchronous timing methodology to deal with paths with an inverted feedback loop.

In Section 4.4, the need for sync-async latch pack is explained where clock crosses the asynchronous boundary. Timing methodology at this interface must be clearly understood so that the design is not at risk for metastability condition. Thus, the methodology of "Auto REAL Adjust" cannot be applied as shown in Fig. 5.10, at sync-async interface logic. This problem leads to develop macro synthesis and automatic timing methodology at sync-async interface for all levels of the chip hierarchies i.e. MMLGS, unit and chip.

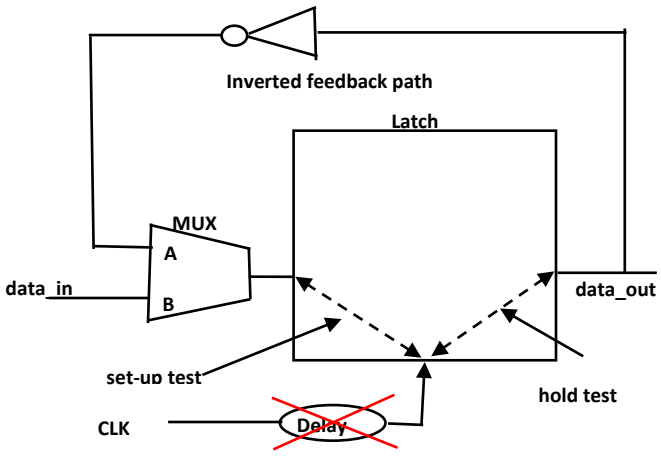


Fig. 5.9: Latch with feedback path.

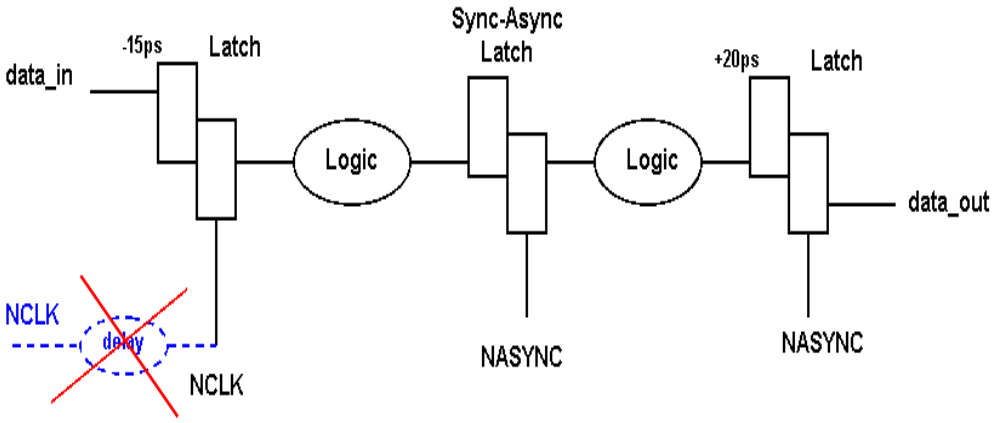


Fig. 5.10: No auto REAL adjust at sync-async interface.

An algorithm is proposed here, which would find the names of clocks the latches are connected to and to examine the phase relationships between them. If they are out of sync based on the clock phase definition of the project, then both synthesis and timing tool would add these latches to "Auto REAL Adjust" (ARA) exclusion list and slack will not be shared between them. For the proposed methodology and algorithm development, IBM's home grown tool called "geyzer" under the umbrella of formal verification tool "simarama" [62] was used. This tool checks for the existence of synchronous-asynchronous clock domain and makes sure that sync-async latch packs are used at this domain crossing. Once this interface is identified, the proposed synthesis and timing methodology do not allow any "auto REAL adjust" at this interface.

To support the timing and synthesis methodology at both synchronous and sync-async interfaces, the following "**Algorithm 5.5**" (pseudo code) is proposed in macro synthesis, unit and chip timing environment. The application of "Algorithm 5.5", Fig. 5.11, allows us to apply appropriate credits to the timing failed paths at any level of physical design hierarchy. Proposed methodology examines the slack sharing criterions i.e. 1) make sure right clocking interfaces, 2) non-inverting feedback loop between the input and output and 3) has enough slack to share. Once these criterions are met, appropriate slack relief will be applied at all level of the physical

design hierarchies during macro synthesis and timing for both synchronous and sync-async interfaces.

Algorithm 5.5: pseudo code for proposed timing methodology to apply "Auto REAL adjusts"

```
##pseudo code to identify latch path to apply Auto "REAL" adjust"
 1:  start chip_timing_run
 2:   start unit_timing_run
 3:         Start macro_timing_run
 4:             slack_sharing_algorithm             {
 5:                 for each latch in the design:
       ## Tool read design netlist to identify if a candidate for real adjust
 6:    step1:
 7:                      if {inverted feedback loop exists at latch input}{
 8.                        perform late mode hold check at latch and
                           add latch name to timing tool's REAL adjust exclude list
9.                                                                        }
10:                     else {
11:                       read out_of_clock_phase output file to recognize the existence of
                          SyncAsync latch
               #geyzer, a part of IBM's simarama formal verification tool [62], is used in this
               #exercise that creates out_of_clock_phase output file (SyncAsync Latch) after
               #reading logic code.
12:                    info <= split sync_async_latch file
13:                   foreach info {
14:                     add latch to timing tool's REAL adjust exclude list
                        ## Cannot share slack across latches
15:                                 }
16:                       }
17:                   else {
18:                       meets slack criteria on input (n) and output (n+1) of latch
                             ## Has enough positive slacks to share between two cycles.
19:    input needs slack relief (negative slack) and output side (positive slack) is > cutoff
       slack to provide relief?
20:    step2: Determine real adjust value (X ps) at point n borrowing up to the real adjust limit
       ## X ps (pico second) varies from design to design so that both late and
       early mode timings are met
       ## Depending on all design constraints and technology variability
```

```
21:    step3: Credit setup test at n by X ps
22:    step4: Delay clock launch segment to n+1 by X ps to reflect the borrowed amount
       ## Slack borrowing by X ps from next cycle(n+1)
23:    step5: Recalculate and report slacks at n and n+1
24:            re-iterate if requested
25:                          }
26:                                                 }
27:    create macro_timing_rule
28:    end macro_timing rule
29:      load unit_netlist         {
30:    if {unit_level_latch exist} {apply slack_sharing_algorithm}
31:       create unit_timing_rule }
32:    end unit_timing_rule
33:    load macro_timing_rule
34:    load unit_timing_rule
35:    run chip_timing
36:  end chip_timing_run
```

Fig. 5.11 represents the proposed design flow dealing with sync-async interface and multi-power domain in MMLGS, unit and chip level timing as outlined in Sections 5.2.2 and 5.2.3. The proposed methodology steps for multi-voltage and sync-async interface have been added as dotted box to the existing design flow. For multi-power domain synthesis, Algorithm 5.4 has been added to the traditional flow. Similarly, for multi-clock design methodology, a list of sync-async latches is passed to synthesis and all timing steps so that slack sharing at sync-async interface does not take place while rest of the design can take timing relief as outlined in Algorithm 5.5.

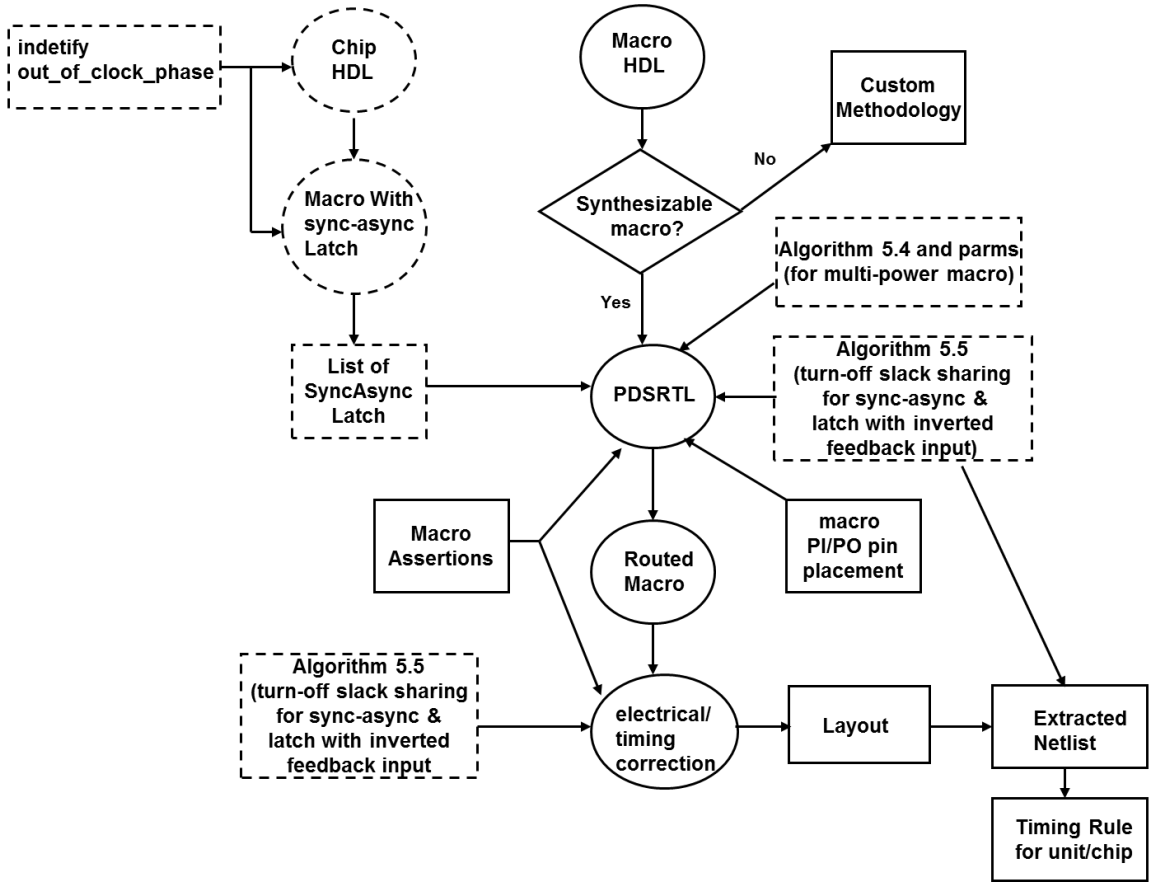


Fig. 5.11: Proposed design flow at sync-async interface and multi-power domain paths.

5.3 Congestion Analysis in MMLGS

In practice, different flavors of congestions can occur during physical synthesis optimization. To guide the tools to mitigate the congestion, congestion models and metrics are necessary. Congestion creates more routing and timing challenges than logic designers can anticipate due to the tendency of sharing of same physical proximity. Even though congestion problems have been considered as physical construction problem, the logic coding style can contribute a fair amount of challenges in routing a macro. Since most of the physical designers are not familiar with logic, fixing congestion thus take a high toll on physical implementation and can become critical path to chip design schedule. Some detail placement techniques, such as

dynamic programming could be used as well for better timing and congestion. The spreading techniques [63-64] are used to mitigate the global and detail routing congestion. Congestion could come from: a) Global interconnect such as IO placement of the chip, wiring channel allocation for global wiring to go over macros, b) Local congestion such as bad macro pin placements and limited routing resources for macros, and c) Logic induced congestion such as logic structure and cell selection. A common example is an extremely wide multiplexor. An example of local congestion is shown in Fig. 5.12, where input pins of the embedded hard IP are blocked by the clock reservation. Since clock PIN reservations are fixed by methodology, in order to fix the local congestion, the IP placement is moved around for a better PIN accessibility to the IP. However, congestion could happen at the higher metal layers when nets are blindly promoted to achieve best timing results [65].

There are few ways to address all these congestion problems by carefully coding and placing macro pins, and allocating routing resources. The spreading techniques are used to mitigate global and detail routing congestion for both vertical and horizontal wiring resources. For more details about congestion analysis, please refer to Appendix A2, 2.1 and 2.2. Modern synthesis tool has come a long way to perform congestion aware routing in short time by using cost function when create logic structures for placement algorithm.

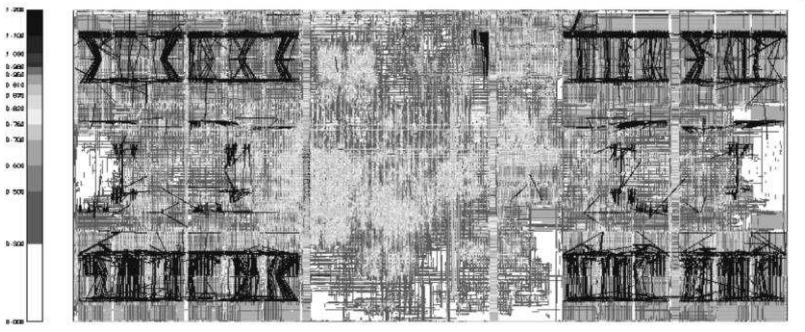


Fig. 5.12: Local congestion problem in synthesis.

With the latest fast global router and benchmarks containing information to perform both placement and global routing, it is now possible to perform routability-driven global placement using accurate congestion estimation techniques. Some work in this area [66-74] have shown promising results. In this dissertation, analysis of the routing quality of the post routed unit with following data is done to make sure: a) design will be robust and pass all backend tools, b) still have enough routing and physical placement allocation for future changes.

In RLM (Random Logic Macro) design methodology, the layout is expected to be clean by construction. This means the routed design must pass all physical design verification (PDV) checks such as DRC (Design Rule Checker), LVS (Layout vs Schematics), YIELD (defect less chip) plus all electrical rule checks. However, it is not uncommon that due to the routing constraints and congestion, the router would leave the design with a handful number of failures that need manual clean-ups.

5.4 Noise Analysis in MMLGS

In general, noise is not a problem in smaller macros, especially RLM because of the random nature of the routings. However, in a larger design, MMLGS as an example, circuit malfunctions can occur anywhere when composite noise is greater than the noise margin of the design because of the long run of wires side-by-side, especially in congested areas. Thus, noise simulations are performed for every net of the design using a gate-level noise simulation tool, which analyzes *R(f)L(f)C* data of a synthesized circuit for evaluation. To fix noise failures in the design, an automated parameter-based solution is much more effective, where the designer can restrict routing layers, shield nets, space out neighbors, buffer nets, if possible, to change the timing window of the aggressors, so that composite noise amplitude is below threshold of the noise margin.

5.5 Electromigration in MMLGS

Unidirectional current flow can lead to interconnect (wires and vias) degrade by causing an open in the circuit that is termed as Electromigration (EM) and there by shortening the time to failure(TTF), which is model as follows [75]:

$$TTF \alpha \exp(c/T)/J_{dc}^2 \quad (5.5)$$

Where T is temperature and Jdc is current density.

As equation (5.5) suggests, EM gets worse very quickly at higher temperature, especially around power and ground supply. To prevent this scenario, a power router is run to make sure design has adequate wide strapping and VIAS (contacts between wires) for both power and ground routing. To ensure EM reliability, global signal nets such as clock nets needed to be looked at in MMLGS macros. The bidirectional current in global signal line causes self-heating of interconnect because of the electrical power dissipation due to resistance of interconnect [76], which can be determined by following equation (5.6).

$$Ieff=SQRT((switching\ factor/tcycle*fi^2\ (t)d(t))) \quad (5.6)$$

where *i(t)* is instantaneous current and *tcycle* is cycle time.

In order to limit Ieff for a robust design methodology, the following individual options can be combined:

- Use higher layer metals for longer nets
- Add more vias on wider nets
- Divide and buffer nets to reduce loads

5.6 Bug Fixes and Design Change in MMLGS

While late bug fixes and new functional changes are common and expected, a fully automated way to incorporate these Engineering Change Orders (ECO) is an important factor to reduce turnaround time. Later in the design cycle, a fully automated way to perform these ECOs not only saves time to implement but also allows designers to work on multiple changes simultaneously. In a fully synthesized methodology, designers will have more flexibility to add or change logic without ripping up the existing design. It is very common that highly complex unit will see anywhere between 5-10 bugs to fix every week and thus will require a few engineers to work in parallel, creating dependencies on each other for timing and backend design closures. However, in the proposed design methodology, one designer can incorporate all changes in MMLGS, as high as ~500 logic gates, in one automated ECO process, giving the tool an opportunity to optimize more efficiently across logical boundaries. It is recommended that these ECO gates to be ~500 logical gates because beyond that, the design may start to see local routing and timing degradation given that the ECO tool works only in areas where the design has been changed, keeping the rest of the design fairly unchanged. Should the designer need to add more gates, a full synthesis is recommended instead of incremental. In a regular or "vanilla" synthesis flow, where each macro is designed independently, a big ECO of approximately 400-500 logical gates may involve multiple macros. Thus, the designers must incorporate the changes independently and may result in a sub-optimal solution, requiring more time to implement. Proposed MMLGS methodology can reduce TAT significantly to incorporate this late change by combining many ECOs together. The runtime of a big ECO depends on how the existing design is, regarding congestion, timing, and routing resources. On the average, a timing-aware ECO in the MMLGS can take approximately 3 to 4 hours runtime for just the synthesis part.

Chapter 6

Experimental Results

6.1 Multi-Million Logic Gate Synthesis (MMLGS) Methodology

6.1.1 Experiment Setup Using the L2 Cache Unit in IBM's Power8

Fig. 6.1 shows the floor-plan of IBM's Power8 microprocessor design, the picture on the right is a part of the die which shows the inclusion of L2 cache unit that was used as a test case for this research work. This MMLGS has 512 Kilo-Byte of cache that interface with "core" providing faster access rate. The L2 cache unit has interfaces with both the core and the L3 cache. Part of the L2 cache unit that interfaces with the core, operates at 1:1 (4.5 GHz) clock frequency, i.e., the same speed as the core, supporting high-performance core logic design.

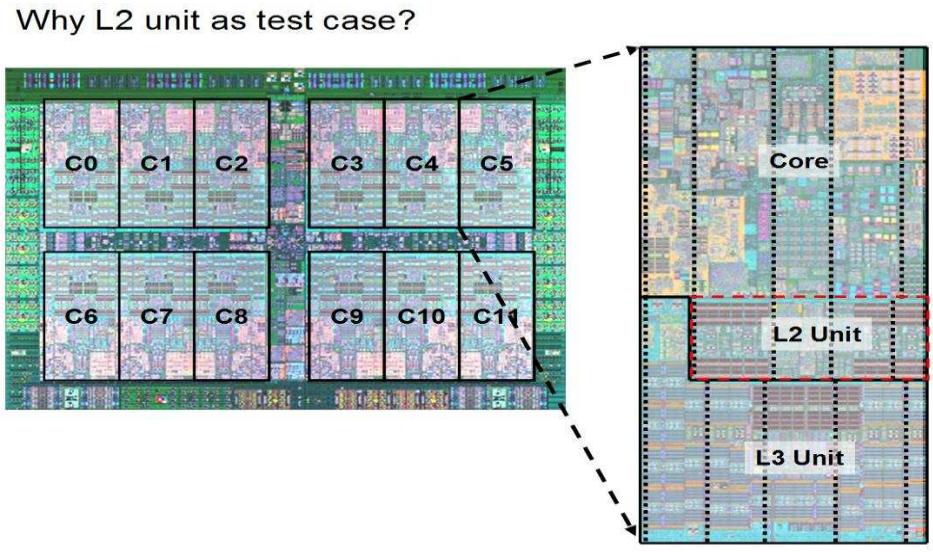


Fig. 6.1: IBM's P8 Microprocessor.

The rest of the L2 unit's clock operates synchronously at 2:1 (2.25 GHz) interfacing cache and directory support; i.e., half of the core frequency. The L2 cache unit represents an interesting test case for developing a synthesis methodology for the following reasons:

- It is an area-challenged unit.
- It has both 1:1 and 2:1 clocking methodology.
- The paths on 1:1 clocking are timing challenged.
- It requires dual-voltage routing and clock gating.
- It uses a combination of data flow and control macros.
- It is a large unit to challenge tool methodology and data management.

As benchmark data for the L2 test case unit, a timing-closed and routed design that has a hierarchical bottom-up design, i.e., each individual macro is built and timing clean, has been studied, Fig. 6.2. This benchmark data is a part of IBM's next-generation microprocessor, Power8^{TM}, which is already in production. Some of these macros were built as custom design, and some were fully synthesized with individual macro boundaries. The PD resources based on the benchmark design have been tabulated at the end of this chapter to compare with the proposed design methodology. Once the benchmark data was set, a "vanilla" synthesis flow was tried on a minimally placed L2 cache unit to set the bar for baseline experiments and shown how the current industry-standard synthesis flow is broken in placing timing-critical gates, routing and closing timing of the design. After that, proposed design methodology was followed to produce data to compare with each other. The same routing layers, wiring resources and timing constraints were used in all subsequent experiments to compare the results apples-to-apples.

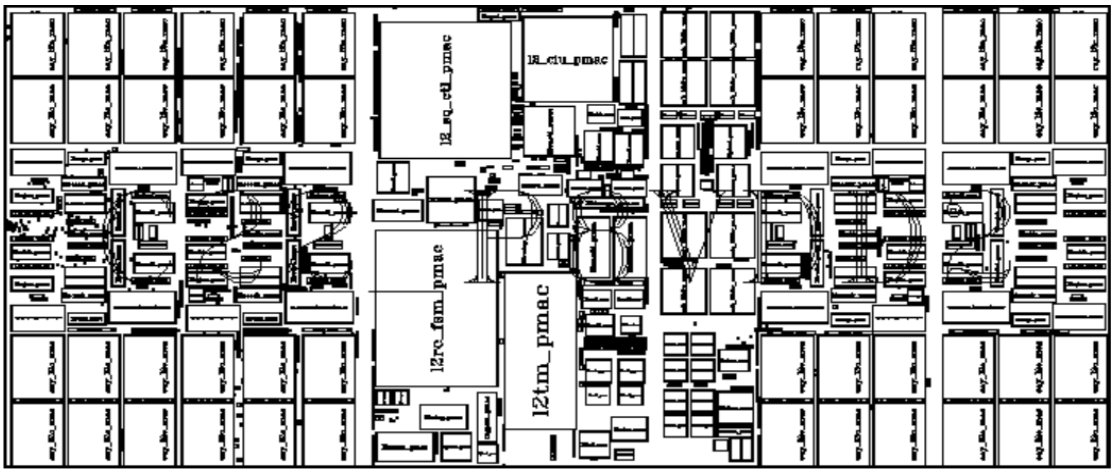


Fig. 6.2: Floor-plan of test case: L2 cache unit.

6.1.2 Placement of Macros

The macro placement follows the uArch and data flow of the unit, keeping critical paths in mind. To better guide synthesis for better wiring, congestion and timing, a well-planned unit floor-plan is an essential component so that pin placement, metal usages and data flow of the macro can be determined earlier in the design flow. In this test case, most of the critical paths existed between the Load Store unit of the core and L2 cache to support demand load requests. However, with a synthesis-based L2 cache design, L2 internal paths turned out to be as critical as the external paths. Thus, closing the timing for the L2 internal paths became one of the challenges in this design methodology. The array-based macros such as cache and LRU were pre-placed to align the data flow of the chip and the unit. These array macros were designed either as full custom or growable arrays using the methodologies explained in Section 4.3. To develop MMLGS methodology, "black-box" methodology for these arrays, where synthesis uses the array's timing rule and placement information, was used to optimize the rest of the design during the physical design flow.

6.1.3 Power Routing

The power grid in the unit is generally dictated by the power grid of the chip and how units are being architected to support power budget of the chip. However, in today's high-speed and high-performing chip, it is very common that a unit like the L2 cache demands two power grids, one for regular logic (Vdd) and other for cache design (Vcs). Vcs, slightly higher than Vdd, is required for SRAM cells for performance and stability at low voltage operation. Centering SRAM cell Vt higher and keeping Vcs sufficiently high mitigate these problems while still allowing logic circuits to scale further for ac power reduction. Thus, to support multiple voltage domains, synthesis methodology was used to support both power routings. Based on certain areas of the floor-plan design, a dual-voltage domain existed and the power supply grid was designed to support internal voltage (IR) drop requirements for the chip.

6.1.4 Clock Routing

Multistage clock buffer trees between the PLL and sector buffers of L2 cache unit were designed as symmetrically as possible to maintain the low skew at the input to the sector buffer. Clock routing is generally done as a pre-route for the unit. In a smaller macro, the clock PINs are top down inside the macro and synthesis treats those PINs as a global pin and thus clock router is not being run at macro level. However, for a big design, like L2, a clock router was called once the macro's layout was generated with pre-reserved routing channel only for clock nets to connect the mesh clock of LCB with clock buffers. Global clock buffer columns were pre-placed and reserved for clock buffers as shown in Fig. 6.2, three vertical columns, where synthesis was prohibited to place any functional-logic-gate i.e. these columns were reserved only for clock-sector-buffers and clock-routings to maintain the desired clock skew of such a high- preforming and timing challenged unit.

A set of well-defined synthesis constraints (parameter) is a pre-requisite to close timing and routing of any design. It is very important to write the logic in such a way that it has good logical boundaries to close timing and fits in the macro's physical size with reasonable utilization for downstream Physical Design (PD) rule checks. A tool aware of power and timing optimization engines could cut down the design cycle time and manual intervention of fixes. PDSrtl [77], a modified version of Placement Driven Synthesis (PDS), is used to build and route the L2 unit with user's input such as assertions, logic and PHYSCELL (Text version of physical abstract) as outlined in Section 2.2 of Chapter 2. PDSrtl has a list of extensive features to design the macros, which include late and early timing, power, electrical and physical design rule check feedback. It also provides flexibility for the designers to add some user parameter to pre-place part of the critical logic, latches, Local Clock Buffers (LCBs), etc. As explained in Section 4.2, LCB enables MS latches to operate in pulse mode in all phases of physical design.

In synthesis timing methodology, cycle stealing enables timing relief for the designers to close critical timing paths, and in most cases, saves power. While closing timing, for both early and late mode within the project- defined slew (rise/fall time) limit, is a very difficult task, this design methodology produced excellent results with synthesis by delivering a routed unit that was a production quality design. In conjunction with current design methodology, after the design was routed, another post processing tool was ran on the routed unit to fix some electrical and timing violations such as slew and slack as a part of automated design flow to make the design ready for physical design verification, extraction and timing-rule generation. To close the design with developed MMLGS methodology, focus was given to the timing, area, power and congestion of the unit as discussed in next sections.

6.1.5 Timing Results

A timing contract, to describe boundary condition such as arrival and departure time, slope and loading at the input and output, was used to allow both unit and chip level design to be analyzed and iterated independently and thereby reducing the turnaround time. One way of addressing timing closure of hierarchical block-based designs is to allocate timing budgets to all the subsystem components, including the global interconnects [78-83]. Timing optimization techniques, such as buffering, gate sizing, cell movement, Vt swapping, cloning, logic decomposition, inverter merging, connection reordering, local logic remapping, can then be applied to fix timing problems. Even though the timing out of 'Rapids' (IBM's home grown tool which fixes post routing electrical and violations if there is any) was considered a first good pass for the designers but as a golden rule both early and late case timing corners were analyzed with a gate level sign-off tool. While working on the synthesis flow, several synthesis knobs were used to close the timing. Fig. 6.3 shows how each bucket of negative slack came down as proposed design methodologies were applied.

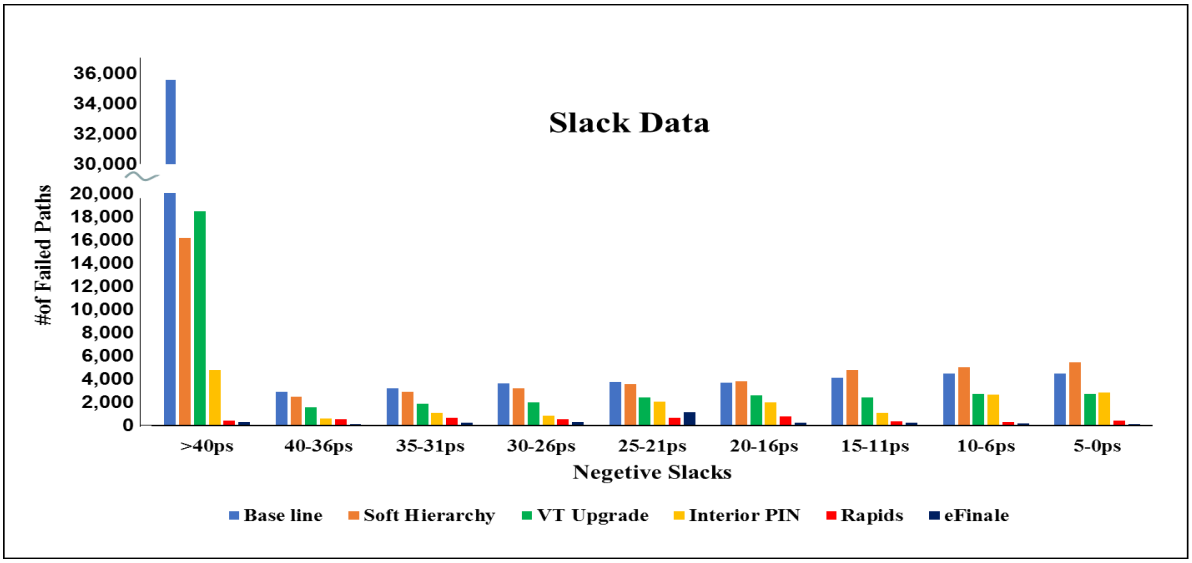


Fig. 6.3: Slack buckets for synthesis methodology.

The slack data in Fig. 6.3 shows that around 65000 negative paths were found in the "base line" experiment. As the proposed methodologies were applied sequentially, approximately 18000, 19000, 9000, and 13000 paths were improved with Soft Hierarchy (SH), Vt Upgrade, Interior PIN and 'Rapids' methodology, respectively. Both SH and Vt upgrade gave the most timing benefit compared to other options just because the synthesis tool found better optimization options to resolve timing fails. Contrary to popular belief, it was interesting to find in this experiment that the "Vt Upgrade" option actually optimized the design better to improve timing-failed paths. For this option, the total power of the design was monitored carefully so that the design complies with the power requirements for the project. The 'Rapids' option brought the negative slacks further down by optimizing the drive load ratio of any given gate. Once 'Rapids' was done, a timing tool was run on the routed netlist and found that 'Rapids' left approximately 3600 total timing paths unfixed. However, of this number, only approximately 600 were unique paths. Analysis further found that most of these 600 paths had approximately 15 common nets; thus, by fixing some of these common nets, the design could be closed. The design was then loaded in a GUI environment. After a few iterations of manual effort, the timing failed paths were closed easily.

In summary, approximately 65000 negative paths were found in the base line experiment. However, after applying the proposed design methodology flow for SH, Vt Upgrade and Interior PIN, the total number of failed paths were only about 18000, which was more than 70% improvement over the base line experiment. As mentioned above, the remaining 18000 timing-failed paths were closed using Rapids and other GUI based home-grown internal tools by the physical design engineers.

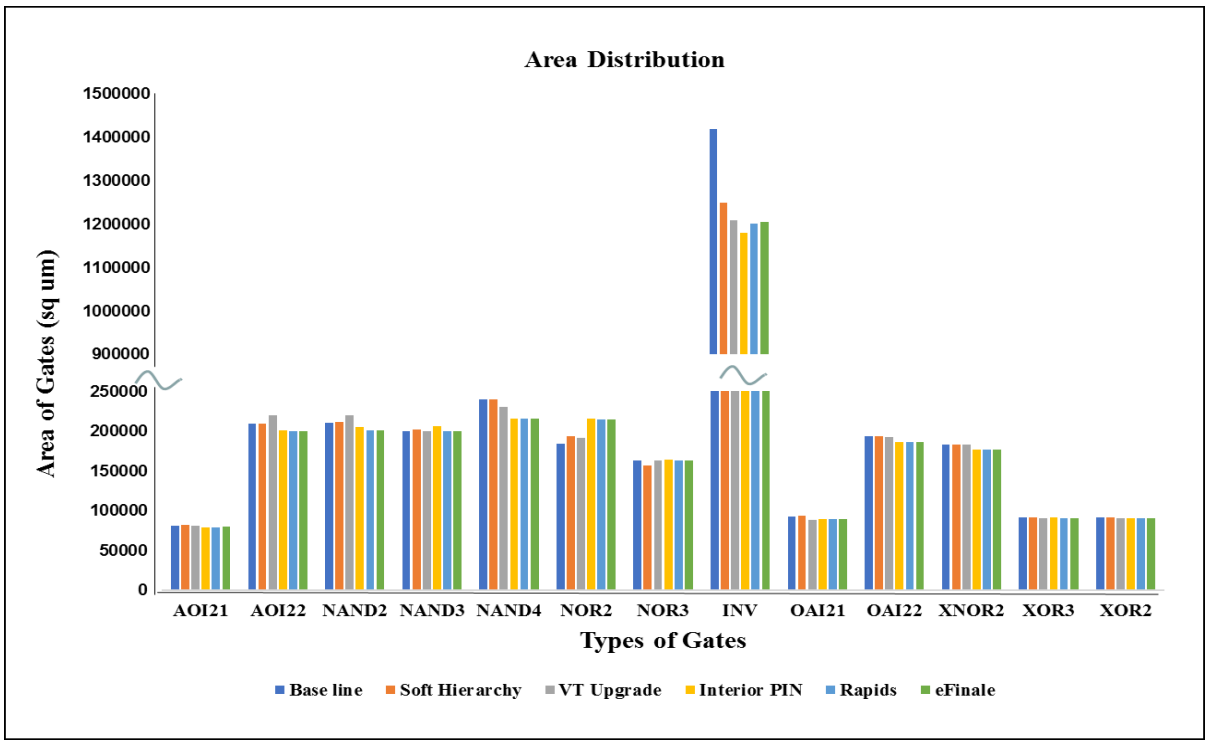


Fig. 6.4: Area distribution of logical gates.

6.1.6 Silicon Area Results

One of the goals of this experiment was to find out how the synthesis tool came out with area distribution for gates used in the design. Since MMLGS has a total picture of the gates to be used in the unit by breaking macro boundary, it can easily optimize gate usages for better timing and power. Fig. 6.4 shows that the complex gate usages did not swing much but the usages of the inverter changed in some cases as the timing was closed with the proposed design methodology.

The usages of inverter in this design was overwhelmingly more than any other gates, which was expected because of longer wire run length (RC) in MMLGS. However, as the timing critical paths were analyzed, logical and electrical efforts were taken into consideration to look at the opportunities to make sure complex gates were utilized for minimum path delay and remove unnecessary buffers, dangling wires etc. The opportunities to forward the logic, isolate critical

gate loading, isolate primary output (PO) from internal logic cone etc. were also looked at as an aid to make timing on critical and highly architected paths in the design. As explained in Section 3.4 of Chapter 3, one of the advantages of MMLGS is, it can do better area optimization by placing logic gates next to each other. It can also eliminate extra buffering or redundant logic and thus saves silicon area. To find the area saving for L2 cache unit, an experiment with 10% less area was conducted and experimental data yielded with acceptable design quality for power, congestion and timing.

6.1.7 Power Data

Four major components of power consumption in VLSI circuits are:

- Clock switching
- Data switching, and
- Device leakage.

In this experiment a goal was set to stay in similar power envelope for the unit because: 1) synthesis has a built in mechanism to balance latch loading for each LCB i.e. if latches are on same clocking and power gating domain, synthesis can move around latches to balance timing and thus dynamic power, 2) it is desirable to get rid of unnecessary buffering and using complex gates to minimize logic functions by collapsing macro boundary, and 3) synthesis also optimizes available wiring resources to balance gate loading and thus saving the driving power of gates. Also, Local Clock Buffer (LCB) enables the latches to operate in pulse-mode, helping the design to stay within power budget by cutting clock toggling power for the design. Experimental data showed that there was no significant power difference between the final solution and custom and semi-custom based L2 cache unit design.

6.1.8 Results on Wire Usages

The major challenges in high speed circuit design, L2 unit as an example here, is to route and make timing internally and externally. One should not only focus on successful routings of the macro but also make sure any future design changes can be incorporated with existing wiring resources and thus allocation of available wiring resources is a crucial step in the design. While there exists extensive literature on repeater insertion and layer assignment lately [84-86], most of them focus on improving the timing performance, and not much have been focused on congestion. Designers should not only pay attention to timing and local hot spots but also global congestions and future ECO process. In the test case design, 4 1X layers, 2 2X layers, 3 4X layers and 3 8X layers were allocated to route L2 unit as shown in Table 6.1. It was found in the experiment that the overall horizontal and vertical usages of the metals were about 56% and 44%, which was optimal even though the macro size was not quite 1:1 in aspect ratio. For more details about routing analysis, please refer to Appendix A3. There is a common misconception that thicker or wider wires are always better but they are not, especially additional VIAs and aggressor capacitance may cause extra delay if attention is not paid. The following recommendation is followed in wire usages to close timing on critical paths in the design.

- Keep wire delay low by using fat wires and redundant VIAs.
- Keep wire capacitance low by placing critical logic together, increased spacing from adjacent aggressor wires, and routing critical nets adjacent to wires with different timing windows or routing them next to any static signals such as power and ground.
- Remove any dangling wires especially for multi fan-out logic cones or share wires wherever possible.

Table 6.1: Wire usages in L2 cache unit routing.

Routing Layer	Preferred Direction (Horizontal/Vertical)	Total Wire Length(micron)	Horizontal Wire Length (micron)	Vertical Wire Length (micron)
M1 (1X Layer)	H/V	0	0	0
M2 (1X Layer)	V	1514426	11655	1502771
M3 (1X Layer)	H	3337125	3331592	5533
M4 (1X Layer)	V	5347433	2173	5345260
M5 (1X Layer)	H	5678731	5676649	2082
B1 (2X Layer)	V	3889129	3401	3885728
B2 (2X Layer)	H	4744529	4740148	4381
E1 (4X Layer)	V	1932741	1731	1931010
E2 (4X Layer)	H	2441000	2440396	604
E3 (4X Layer)	V	322024	238	321786
X1 (8X Layer)	H	575473	575113	360
X2 (8X Layer)	V	413396	92	413304
X3 (8X Layer)	H	134745	134689	56
Total Percent		30330752 100%	16917877 56%	13412875 44%

The test case, L2 cache unit, was routed and built in 22nm SOI technology, POWER8™ (P8), with 13 layers of physical wiring. The design was mostly done in static circuit design with pulse mode being the default working mode for the master-slave latch as a power reduction technique, explained in Section 4.2. Fig. 6.2 shows the base-line L2 unit design with macro partitioning, where all individual macros were designed with either custom, RLM or array design flow and then placed manually.

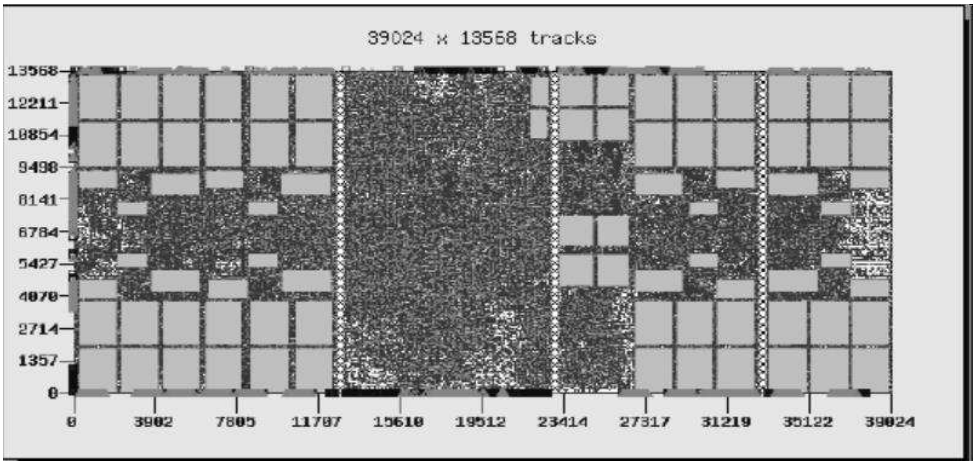


Fig. 6.5: Placed gates in L2 unit with synthesis methodology.

Unit buffering and routing were done with a mix of upper and lower layers of metals by a dedicated unit integrator. Once completed, a unit timer would time the unit to close timing. This whole process was an iterative method that took many iterations for physical design verification checks, noise and electrical and timing checks.

As a part of synthesis based methodology development, L2 cache unit was taken through the design flow using MMLGS methodology. Fig. 6.5 shows physical placement of the gates, including ECO gates for future change, after proposed synthesis flow, where gray boxes were pre-placed array macros. For more details about ECO process, please refer to Appendix A4. In methodology development, it was shown how negative slacks were coming down, keeping the design within similar (less than 2%) power budget to the conventional design. To clean-up some post routing PDV and timing violations, "Rapids" was run to deliver error free routed design to the chip. For more details about timing-failed-paths and post-routing-statistical-data, please refer to Appendix A5 and A6 respectively. Compare to the baseline experiment, P8, newly introduced synthesis methodology and design flow for MMLGS, L2 cache unit design took almost 50% less resources even without dedicated unit timing and integration resources.

Table 6.2: Physical design resource comparison for L2 cache.

Physical Design Resources	Traditional Approach (P8) (man month)	Synthesizable Unit Approach (man month)
Ckt. Designer	18	12
Unit Timer	6	0
Unit Integrator	6	0
Unit Ckt. Lead	6	0
Total Resources	36	12

The PD resources comparison is summarized in Table 6.2.

The L2 cache unit, with around 20 million synthesizable transistors (excluding all array macros), many timing constraints, congestions and routing challenged critical nets, stretched all front and back-end tools by 2-3 times compared to the current available methodology used at IBM. The L2 cache unit designed with proposed design methodology was fully functional to operate around 4.8GHz to 2.4GHz. Due to aggressive fan-out, design integrity, tight power budget and schedules to market, the L2 cache unit had become a tremendous success for developing a MMLGS methodology for the future microprocessor industry.

At the same time, examples were looked at to compare test case results with similar work from other industry leaders such as Intel, AMD and others. It is found that very few published works for high speed and high performing MMLGS. Most examples were found to be in ASIC domain at lower frequency i.e. sub-GHz ranges. Intel's "Superblock" design [87] used similar concept of MMLGS, as stated in Chapter 3, but comparing to L2 cache unit for this experiment, it is at least 2-3 times smaller in transistor counts. Bobcat was AMD's first easily synthesized CPU core area tradeoff when moving away from custom macros to more general designs but it was deemed worthwhile, which operates in the range of 1.3-1.6 GHz in 32 nm technology [88].

Fig. 2.3 shows the comparison of macros design methodology used at IBM for last 16 years starting from 90nm technology. In 90nm, designs were mostly based on custom flow and as time progressed, a huge shift of custom and semi-custom design towards synthesizable macros (Random Logic Macros) was observed.

6.2 Multi-Voltage Synthesis for MMLGS

For synthesis to understand the voltage crossing nets and instances that are connected to different voltage domain, proposed methodology incorporated Algorithm 5.4 along with pre-placement parameter, Fig. 5.4, and VHDL attributes as stated in example (5.2), example (5.3) in the test case. In the experiment, a MMLGS methodology was used, where array IP and LT were instantiated inside a synthesizable macro as shown in Fig. 6.6. This array was a growable 8 transistor SRAM macro designed using semi-custom approach and the Level Translator (LT) was designed as custom using custom design methodology.

In Fig. 6.6, the array is in solid gray color, LT is in solid green color, and Logic Gate (LG) driving Level Translator (LT) is in blue color. LT has both vdd and vio as power supply. LG, the

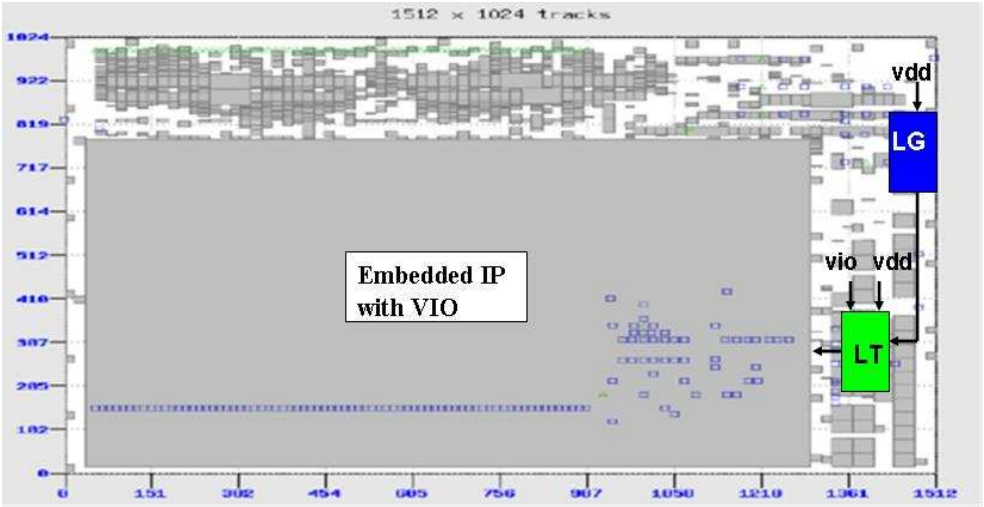


Fig. 6.6: Use of level shifter in MMLGS.

driver of LT, has vdd power supply, while IP is on vio. This macro was built successfully through the proposed MMLGS flow with multi-voltage and routed using 22nm technology with careful pre-placement of the LT along with related logic grouping with respect to multi-power domain in floor-plan. Finally, equivalency check and all backend tools such as LVS, DRC, METH, extraction and timing were ran to make sure that the quality of the design was not compromised. Due to the addition of level translator, design may see degradation of timing. An example in (5.4) of Section 5.2.2.1 shows how wire traits can be used to introduce high performing wires at timing critical areas. In general, when wires are upgraded or made certain wires fat to reduce the RC delay of the timing critical paths, wire usages should be carefully studied in the design and made sure both horizontal and vertical wire usages are at balance and does not create congestion in the design.

The experimental data showed that the proposed synthesis flow built MMLGS macro successfully in multi-power domain with the use of the proposed **Algorithm 5.4**. In addition to the proposed algorithm, wire traits parameters were successfully incorporated to route critical nets and pre-placement algorithm for level translator (LT) at the right power domains. A test case macro, that was built in custom design flow, was used as an example and implemented as MMLGS using the proposed design methodology for multi-power. The custom flow required manual schematics and layout design, where each transistor was hand-crafted to build the macro and satisfy the timing and backend tool flow requirements. On the other hand, proposed MMLGS methodology was fully automated and needed minimum intervention from the designer. This proposed synthesis-based MMLGS methodology allowed us to cut down the physical design efforts by more than 50%, shown in Table 6.3, in terms of physical design resources and time to deliver the timing and routing closed design to the unit and chip.

Table 6.3: Physical design resource comparison for a typical macro design.

Macro Physical Design Activity	Custom Design Approach in previous Chip Design (man week)	Synthesis Design Approach (man week)
Logic Study, Macro Sizing, Input/Output Placement, Routing Resource Study etc.	2	2
Schematic Design and Equivalency Check	3	0
Logic Gate Placement	1	1 (if needed)
Schematic Based Timing Tuning	1	0
Layout Design and Routing	4	1
Post Layout Timing Closure	1	1
Total	**12**	**5**

6.3 Sync-Async Timing Methodology

To take full benefit of the proposed timing methodology at sync-async interface, MMLGS macros were built with the proposed design flow shown in Fig. 5.11. For the proposed timing methodology development, a critical unit, from POWER8^{TM} microprocessor design, was used. It was found that 1265 negative paths (approximately 28% of the total unit paths that were failing) shifted the slack buckets and gave a huge relief to close timing on critical paths as is shown in Fig. 6.7. In order to find data for both ARA and exclusion of ARA for timing at sync-async interface as explained in Section 5.2.3, another set of examples was picked-up that had 6 chip level functional units from POWER8^{TM} microprocessor design . These two sets of examples were independent of each other. “Auto REAL Adjust” was applied across these units to take advantage of unused cycle time. As graphed in Fig. 6.8, the failed timing paths were gathered in three different ways:

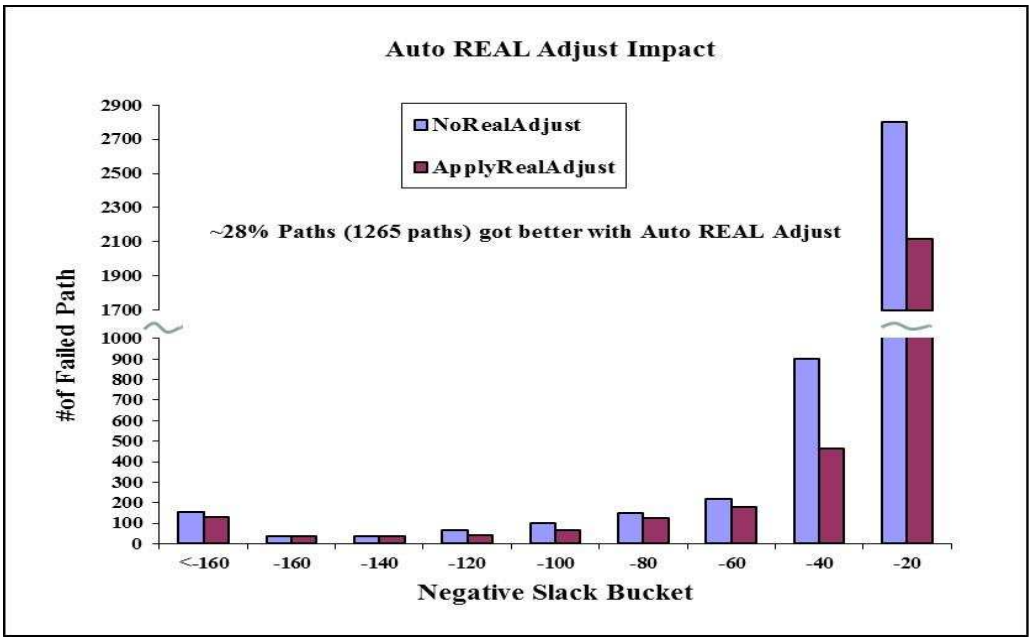


Fig. 6.7: Distribution of negative paths with and without "Auto REAL Adjust".

1) Do not apply any credits as "REAL Adjust", 2) Apply "REAL Adjust" across all macros, and 3) Do not apply "REAL Adjust" for macros with sync-async interface latch and compare data to each other. In this exercise, it is found that in EN unit, 88% of the failed paths got timing relief. Also, only "EN" showed the presence of macros with sync-async interface latches because the total #of failed paths went up by 8% between experiment #2 and #3.

To compensate this increase of timing failed paths, "Physical Design Solution" such as wire traits, *Vt* upgrade, logic re-structure was applied. However, as shown in Fig. 6.8, for "EN" unit there were still some timing failed paths which required logic changes to close timing. These data were based on fully routed, extracted and timed macros with black-boxed at unit and chip timing. For sync-async interface methodology development in synthesis, Fig.5.11, cycle stealing was being turned-off for all frontend tools, i.e. synthesis, empad (Early Mode Padding), MAR (Routing Tool) and 'Rapids', to make sure that macros did not have optimistic timing view before

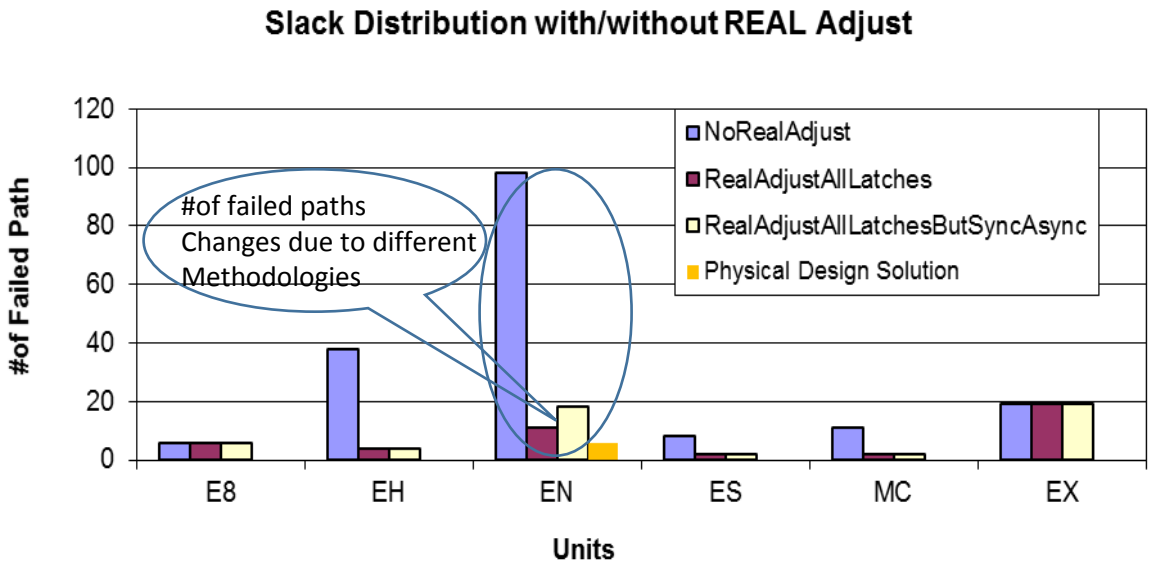


Fig. 6.8: Slack distribution with/without "Auto REAL Adjust".

the unit/chip was timed. Experimental data clearly showed that the "Auto REAL Adjust" can be used across the units because the number of the failed paths improved by approximately 28% in one unit and 88% in another. However, in reality, due to the possibility of metastability at sync-async interface, the full advantage of this timing relief could not be taken in EN unit and thus developed timing algorithm was applied to turn-off slack transfer at sync-async interface. For that reason, slack improvement reduced to 80%, which was still a phenomenal deal with much reduced risk factor, thereby saving a huge design efforts and most cases power for chip because designers did not have to apply power hungry gates to make timing at critical paths or interfaces.

Chapter 7
Conclusion and Future Work

To stay in competitive performance, functions and power envelop, challenging innovative design methodology and implementation techniques are most important to modular designs to cut down development cost. There has been a big shift in microprocessor design in favor of synthesis based design that needs multi-clock and multi-power support. Every power domain requires independent local power supply and grid structure and thus both front and back-end tools along with standard cell libraries need to optimize the design for worst case PVT (Process, Voltage, temperature) variations. Common gate libraries needed to be characterized for different voltage levels that are used in the design. Multi-power domains need multiple power grid structure and a suitable power distribution among them. For microprocessor or ASIC/SoC more careful floor-planning, synthesis algorithm, timing methodology to support power planning with addition of LT, are very essential.

In Chapter 6, it was shown that MMLGS macro with 2-3 times bigger than industry standard was designed using the proposed design flows and algorithms which were explained and elaborated in Chapter 5. With aggressive timing closure, tight power budget and schedules to market, the L2 cache unit was an ideal test case for developing synthesis methodology in MMLGS for the future microprocessor industry. Experimental data showed that the proposed methodology enabled MMLGS in both multi-clock and multi-voltage domain and maximize timing benefits at all level of physical design hierarchies. It was successfully proven that physical design resources can be cut down by about 50%.

As the MMLGS based flow for L2 cache unit suggests, there are some white space at the wings of the unit. Another experiment with 10% less unit area was conducted with acceptable design quality, power, and timing. However, it is left for the future consideration of making the unit smaller. As mentioned earlier, this unit was used as an interesting test case to prove the proposed concept. Thus, upon successful implementation it is needless to mention that, other units such as L3 cache unit, memory controller unit or any other timing and area critical unit can be a good candidate to adopt this methodology to cut down future development cost, and turnaround time significantly as it did for the test case.

While MMLGS methodology is proposed in this dissertation, there are still some areas that could be improved further for more productivity. Future improvement on the proposed methodology includes the following few areas.

- While the MMLGS design methodology was developed, few constraint files were created and included manually in synthesis flow as an aid to improve the quality of the results. These constraints were: 1) wire traits file, 2) pre-placement file for embedded IP and, 3) placement file for soft-hierarchy etc. It would be more productive and time saving for the users if they did not have to create these constraints files manually and future synthesis methodology automatically incorporate the necessary steps as a part of MMLGS in both multi- clock and power domain.

- As millions of gates being packed in MMLGS methodology, the physical size of the macro gets bigger and thus longer and higher levels of metals being used in MMLGS to route the macro within the project defined timing constraints. These longer and higher levels of metals drive more current and may be routed next to each other, which can cause more electromigration and noise problem in MMLGS compare to smaller RLM

macros. Thus, the future MMLGS methodology needs to focus more and work on better correlation of noise and electromigration issues during the synthesis flow so that no big surprise is found once the macro is built and extracted for all backend sign-off tools.

- MMLGS methodology deals with millions of logic gates for both frontend and backend tools to improve the design efficiency. These tools include: synthesis, routing, Test Bench, PDV, electromigration, noise, power etc. The big data volume of MMLGS could create potential tool runtime and data volume problem i.e. users will need more working space to save the design data as well as more memory for the computers. To support these requirements, a future methodology is essential so that each tool can pick-up the data volume and machine with appropriate capacity on the fly to complete the job.

- To save the runtime during ECO, MMLGS will need a methodology so that each tool in the backend flow is ran only on the area of the design that was changed. For example, if there was a noise fix or some logic gates being added in a particular area of the MMLGS, backend tool sets need to have the capability of running design checks only on the area that was changed rather than running on the whole design. This capability will reduce the backend tool run-time and memory usages of the machines significantly.

- The proposed timing methodology for sync-async interface does not share any slack between latches at this interface. A future enhancement to this methodology can be developed so that "some" slack can still be shared so that design does not see any chance of metastability. This methodology development will require a "deep" study of technology variation due to PVT (Process, Voltage and Temperature) and factor that in to determine how much slack can be shared at sync-async interface safely.

Bibliography

[1] F. J. Pollack, “New microarchitecture challenges in the coming generations of CMOS process technologies (keynote address) (abstract only),” in Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture, Washington, DC, USA, 1999.

[2] S. Durairajan et al, “Forecasting Microprocessor Technology in the Multicore Era Using TFDEA,” Proceedings of PICMET '13: Technology Management for Emerging Technologies., pp 2108-2115, 2013.

[3] S. Borkar, “Thousand core chips: a technology perspective,” in Proceedings of the 44th annual Design Automation Conference, New York, NY, USA, 2007, pp. 746–749.

[4] H. P. Hofstee, “Power efficient processor architecture and the cell processor,” in 11th International Symposium on High-Performance Computer Architecture, 2005. HPCA-11, 2005, pp. 258–262.

[5] “International Technology Roadmap for Semiconductors Edition Lithography the Itrs - PowerPoint,” Docstoc.com. [Online]. Available: http://www.docstoc.com/docs/72756251/International-Technology-Roadmap-for-Semiconductors-Edition-Lithography-the-Itrs---PowerPoint. [Accessed: 07-Feb-2013].

[6] S. Borkar and A. A. Chien, “The future of microprocessors,” Communications of the ACM, vol. 54, no. 5, p. 67, May 2011.

[7] POWER8 [Online]. Available: https://www.microway.com/download/presentation/IBM_POWER8_CPU_Architecture.pdf

[8] Technology Roadmap of INTEL's Processors [Online]. Available: https://www.scribd.com/presentation/253118046/TechInsights-Technology-Roadmap-INTEL-Processors-2014.

[9] List of AMD Microprocessor [Online]. https://en.wikipedia.org/wiki/List_of_AMD_microprocessors.

[10] M. Hossain, C. Desai, T. Chen, V. Agarwal "Synthesis Based Design and Implementation Methodology of High Speed, High Performing Unit: L2 Cache Unit Design", Integration the VLSI Journals, vol 49, pp. 125-136, March 2015.

[11] M. Hossain, J. Badar, J. DiLulo, T. Chen , "A Practical Automated Timing and Physical Design Implementation Methodology for the Synchronous Asynchronous Interface and Multi-Voltage Domain in High-Speed Synthesis", Microprocessors and Microsystems, vol 45, pp. 241-252, August 2016.

[12] https://www.google.com/search?q=microprocessor+performance+trends&biw=1680&bih=900&tbm=isch&tbo=u&source=univ&sa=X&ved=0ahUKEwj7_-Wbw93OAhVD6CYKHYVlAugQsAQIZA&dpr=1#imgrc=nc8jPQ2b3BRufM%3A.

[13] M. Hossain, E. Fluhr, A. Hall, V. Agarwal, "Physical Design and Implementation of POWER8™ (P8) Server Class Processor", "Midwest Symposium on Circuits and Systems (MWSCAS)", August 2-5, 2015.

[14] J. Wu et al, "Congestion Aware High Level Synthesis Combined with Floorplanning", 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, pp. 935-938, 2008.

[15] J.Cong et al, “Multilevel approach to full-chip gridless routing”, International Conference on Computer Aided Design. ICCAD, pp. 396-403, 2001.

[16] B. Halpin et al, “Timing driven placement using physical net constraints”, Proceedings of the 38th Design Automation Conference, pp. 780-783, 2001.

[17] S. Thakur et al, “An optimal layer assignment algorithm for minimizing crosstalk for three layer VHV channel routing”, IEEE International Symposium on Circuits and Systems (ISCAS), pp. 207-210, 1995.

[18] H. Zhou et al, “An optimal algorithm for river routing with crosstalk constraints”, Proceedings of International Conference on Computer Aided Design, pp. 310-315, 1996.

[19] R. Dekker, “Electronic design,” [Online}. Available: http://electronicdesign.com/what-s-difference-between/what-s-difference-between-vhdl-verilog-and-systemverilog.

[20] M. Lightner, and W. Wolf. “Experiments in Logic Optimization”. InProc. ACM/IEEE Intl. Conf. on Comp. Aided Design, pages 286-289, Nov. 1988.

[21] J. Vygen, “Steiner Trees in Chip Design,” [Online]. Available: http://www.or.uni-bonn.de/~vygen/files/hz1h.pdf.

[22] Gi-Joon Nam, Charles J. Alpert, Paul Villarrubia, Bruce Winter, Mehmet Yildiz, The ISPD2005 placement contest and benchmark suite, Proceedings of the 2005 international symposium on Physical design, April 03-06, 2005, San Francisco, California, USA.

[23] Gi-Joon Nam, ISPD 2006 Placement Contest: Benchmark Suite and Results, Proceedings of the 2006 international symposium on Physical design, April 09-12, 2006, San Jose, California, USA.

[24] Jarrod A. Roy, Natarajan Viswanathan, Gi-Joon Nam, Charles J. Alpert, Igor L. Markov, CRISP: congestion reduction by iterated spreading during placement, Proceedings of the 2009 International Conference on Computer-Aided Design, November 02-05, 2009, San Jose, California.

[25] Yuncheng Zhang, Chris Chu, CROP: fast and effective congestion refinement of placement, Proceedings of the 2009 International Conference on Computer-Aided Design, November 02-05, 2009, San Jose, California.

[26] G. Kuda, "Using Physically Aware Synthesis Techniques to Speed Design Closure of Advanced-Node SoCs," [Online]. Available: http://chipdesignmag.com/sld/blog/2015/03/23/using-physically-aware-synthesis-techniques-to-speed-design-closure-of-advanced-node-socs/

[27] Chen Li, Min Xin, Cheng-Kook Koh, J. Cong, P. H. Madden, Routability-Driven Placement and White Space Allocation, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.26 n.5, p.858-871, May 2007.

[28] U. Brenner, A. Rohe, An effective congestion-driven placement framework, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.22 n.4, p.387-394, November 2006.

[29] Wenting Hou, Hong Yu, Xianlong Hong, Yici Cai, Weimin Wu, Jun Gu, William H. Kao, A new congestion-driven placement algorithm based on cell inflation, Proceedings of the 2001 Asia and South Pacific Design Automation Conference, p.605-608, January 2001, Yokohama, Japan.

[30] Z Li et al, “Guiding a Physical Design Closure System to Produce Easier-to-Route Designs with More Predictable Timing,” DAC, page 465-470, 2012.

[31] “Eliminating Routing Congestion Issues with Logic Synthesis”, [Online]. Available: http://www.cadence.com/rl/resources/white_papers/routing_congestion_wp.pdf

[32] R.T. Hadsell and P H. Madden, "Improved global routing through congestion estimation," in Proc. Design Automation Conference, pp 28-31, 2003.

[33] I. Cheng, W.N.N. Hung, G. Yang and X. Song, "Congestion elimination for 3D routing," in Proc. VLSI, Proceedings. IEEE Computer Society Annual Symposium, pp. 239-240. 2004.

[34] T. Ling, X Hong, H. Bao, Y. Cai, J. Xu, Y. Wang and J. Gu, "An efficient congestion optimization algorithm for global routing based on search space traversing technology," ASIC, 2001. Proceedings. 4th International Conference, pp.1 14-1 17. 2001.

[35] M. Wang and H. Sarrafzadeh, "Modeling and minimization of routing congestion," Proceedings of the ASP-DAC 2000.Asia and South Pacific, pp 185-190. 2000.

[36] Z. LI, W. Wu and X. Hong, "Congestion driven incremental placement algorithm for standard cell layout," Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003.Asia and South Pacific, pp. 723-728, 2003.

[37] X. Yang, R. Kastner and M. Sarmfadeh, "Congestion reduction during placement based on integer programming," Computer Aided Design, ICCAD 2001, pp.573-576, 2001.

[38] W. Wang, J. Bian and Y. Wang, “A Congestion Driven Re-Synthesis Method after Fhorplanning”, Proceedings of International Conference on Communications, Circuits and Systems.2005, pp 1220-1240.

[39] “Iteration problems in the flow”, [Online]. Available: http://csg.csail.mit.edu/6.375/6_375_2006_www/handouts/lectures/L05-Synthesis-Placement-Routing.pdf

[40] B. Dougherty and M. Kazda, "Rapids: Post-Routing Timing Closure," in *DAC 2010 User track Poster 1U.5.*

[41] “Introduction to High-Level Synthesis,” [Online]. Available: http://charlotte.ucsd.edu/classes/sp02/cse140/gajski1.pdf

[42] “Intel is working on 15-core ‘Ivytown’ processor for servers,” [Online]. Available: http://www.kitguru.net/components/cpu/anton-shilov/intel-is-working-on-15-core-ivytown-processor-for-servers/

[43] R. Varada et al, “Superblock: A method for synthesizing Large High Performance Designs Without Hierarchy Limits”, DAC 2010.

[44] S. Rusu et al, “IEEE International Solid-State Circuits Conference Digest of Technical Papers”, pp. 213-214, 2014.

[45] A. E. Sjogren et al, “Interfacing Synchronous and Asynchronous Modules Within a High-Speed Pipeline,” IEEE Transaction on Very Large Scale Integration (VLSI) Systems, vol. 8, No. 5, Oct. 2000.

[46] Y. Zhi et al, “An efficient Algorithm for Multi-Domain Clock Skew Scheduling,” Design, Automation & Test in Europe Conference & Exhibition, pp. 1-6, 2012.

[47] B. Tsakin and I. Kourtev, “TIME BORROWING AND CLOCK SKEW SCHEDULING EFFECTS ON MULTI-PHASE LEVEL-SENSITIVE CIRCUITS,” Proceeding of the International Symposium on Circuits and Systems, pp. 617-6202004.

[48] R. Kaushik et al, “Multi-Domain Clock Skew Scheduling,” The international Conferenve on Computer Aided Design, pp801-808, 2003.

[49] L. Li, Y. Lu, H. Zhou, “Optimal Multi-Domain Clock Skew Scheduling,” Design Automation Conference (DAC), pp. 523-527, 2011.

[50] R. Mehra, “Commercial low-power EDA tools: a review,’ Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design, PP. 67-71, 2012.

[51] Synopsys Education & support (2007, December) [Online]. Available: http://www.eda.org/p1801/hm/att-0252/LevelShifter_Syntax_V1.pdf

[52] S. Karapetyan (2011, March 23). Low Power Design Methods: Design Flows and Kits [Online]. Available: http://wwwmayr.informatik.tu-muenchen.de/konferenzen/MB-Jass2011/courses/2/Karapetyan_2_presentation.pdf[73] K. Matt et al, (2006). Multi-Voltage implementation flow with Synopsys tools [Online]. Available: http://vsevteme.ru/attachments/show?content=6305

[53] Power Forward. Verification of Low Power Intent with CRF [Online]. http://www.powerforward.org/media/p/96.aspx

[54] V. Gourisetty et al, “Low power design flow based on Unified Power Format and Synopsys tool chain,” Interdisciplinary Engineering Design Education Conference (IEDEC), pp. 28-31, 2013.

[55] Multi-VDD Design Flow (2013, January) [Online]. Available:

http://venividiwiki.ee.virginia.edu/mediawiki/index.php/Multi-VDD_Design_Flow#Multi_Voltage_Design.

[56] B. Stolt et al, “Design and Implementation of the Power6 Microprocessor,” IEEE Journal of Solid State Circuits, Vol. 43, NO. 1, pages 21-28, 2008.

[57] M. Keating et al, “Low Power. Methodology Manual”, pp 21-31, Springer, July, 2007.

[58] D. W. Dobberpuhl, et al, “A 200 MHz 64 bit dual-issue cmos microprocessor,” IEEE J. Solid-State Circuits, vol. 27, pp. 1155–1167, Nov. 1992.

[59] M. Cho et al, “Soft hierarchy-based physical synthesis for large-scale, high-performance circuits,” U.S. Patent 8 516 412, Aug. 20, 2013.

[60] W. Liu et al, “Routing Congestion Estimation with Real Design Constraints,” DAC, pages 1-8, 2013.

[61] J. Badar, M. Hossain, D. Geiger, P. Villarrubia, “Technique to enable multi power synthesis”, U.S. Patent 9483596 B1, Nov. 01, 2016.

[62] G. Annsen et al, “Circuit verification using computational algebraic geometry”, U.S. Patent 8640065 B2, January 27, 2012.

[63] Taraneh Taghavi, Charles Alpert, Andrew Huber, Zhuo Li, Gi-Joon Nam, Shyam Ramji, “New placement prediction and mitigation techniques for local routing congestion”, Proceedings of the International Conference on Computer-Aided Design, November 07-11, 2010, San Jose, California.

[64] Jarrod A. Roy, Natarajan Viswanathan, Gi-Joon Nam, Charles J. Alpert, Igor L. Markov, “CRISP: congestion reduction by iterated spreading during placement”, Proceedings of the 2009

International Conference on Computer-Aided Design, November 02-05, 2009, San Jose, California.

[65] Charles J. Alpert, Zhuo Li, Michael D. Moffitt, Gi-Joon Nam, Jarrod A. Roy, Gustavo Tellez, "What makes a design difficult to route", Proceedings of the 19th international symposium on Physical design, March 14-17, 2010, San Francisco, California, USA.

[66] Taraneh Taghavi, Charles Alpert, Andrew Huber, Zhuo Li, Gi-Joon Nam, Shyam Ramji, "New placement prediction and mitigation techniques for local routing congestion", Proceedings of the International Conference on Computer-Aided Design, November 07-11, 2010, San Jose, California.

[67] Jarrod A. Roy, Natarajan Viswanathan, Gi-Joon Nam, Charles J. Alpert, Igor L. Markov, CRISP: congestion reduction by iterated spreading during placement, Proceedings of the 2009 International Conference on Computer-Aided Design, November 02-05, 2009, San Jose, California.

[68] Yue Xu, Yanheng Zhang, Chris Chu, "Fast Route 4.0: global router with efficient via minimization", Proceedings of the 2009 Asia and South Pacific Design Automation Conference, January 19-22, 2009, Yokohama, Japan.

[69] Yen-Jung Chang, Yu-Ting Lee, Ting-Chi Wang, "NTHU-Route 2.0: a fast and stable global router", Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, November 10-13, 2008, San Jose, California.

[70] M. Cho, Katrina Lu, Kun Yuan, David Z. Pan, BoxRouter 2.0: architecture and implementation of a hybrid and robust global router, Proceedings of the 2007 IEEE/ACM

international conference on Computer-aided design, November 05-08, 2007, San Jose, California.

[71] Huang-Yu Chen, Chin-Hsiung Hsu, Yao-Wen Chang, High-performance global routing with fast overflow reduction, Proceedings of the 2009 Asia and South Pacific Design Automation Conference, January 19-22, 2009, Yokohama, Japan.

[72] Hamid Shojaei, Azadeh Davoodi, Jeffrey T. Linderoth, Congestion analysis for global routing via integer programming, Proceedings of the International Conference on Computer-Aided Design, November 07-10, 2011, San Jose, California.

[73] Jin Hu, Jarrod A. Roy, Igor L. Markov, Completing high-quality global routes, Proceedings of the 19th international symposium on Physical design, March 14-17, 2010, San Francisco, California, USA.

[74] Myung-Chul Kim, Jin Hu, Dong-Jin Lee, Igor L. Markov, A Simple method for routability-driven placement, Proceedings of the International Conference on Computer-Aided Design, November 07-10, 2011, San Jose, California.

[75] J.R. Black, "Electromigration – A brief Survey and Some Recent Results," IEEE Transactions on Electron Devices, Vol. 16, 1969, pp. 338-34.

[76] R. Berridge et al, ÍBM POWER6 microprocessor physical design and design methodology," IBM Journal of Research and Development, Vol 51, Issue 6, pages 685-714, 2007.

[77] J. Friedrich et al, "Design methodology for the IBM Power7 microprocessor," IBM Journal of Research and Development, Vol 55, Issue 3, pages 294-307, 2011.

[78] A. H. Chao, E. M. Nequist, and T. D. Vuong, “Direct solutions of performance constraints during placement,” in Proc. IEEE Custom IC Conf., 1990, pp. 27.2.1–27.2.4.

[79] J. Frankle, “Iterative and adaptive slack allocation for performance-driven layout and FPGA routing,” in Proc. IEEE Design Automation Conf., 1992, pp. 539–542.

[80] P. S. Hauge, R. Nair, and E. J. Yoffa, “Circuit placement for predictable performance,” in proc. IEEE Int. Conf. Computer-Aided Design, 1987, pp. 88–91.

[81] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, “Generation of performance constraints for layout,” IEEE Trans. Computer-Aided Design, vol. 8, no. 8, pp. 860–874, 1989.

[82] M. Sarrafzadeh, D. Knol, and G. Tellez, “Unification of budgeting and placement,” in Proc. IEEE Design Automation Conf., 1997, pp. 758–761.

[83] H. Youssef, R.-B. Lin, and S. Shragowitz, “Bounds on net delays,” IEEE Trans. Circuits Syst., vol. 39, no. 11, pp. 815–824, 1992.

[84] C. Alpert et al, “Buffer insertion for noise and delay optimization” DAC '98 Proceedings of the 35th annual Design Automation Conference, pp. 362-367, 1998.

[85] A. Acosta et al, “Effects of buffer insertion on the average/peak power ratio in CMOS VLSI digital circuits,” VLSI Circuits and Systems III, Proc. of SPIE Vol. 6590, 659007, 2007.

[86] G. Karimi, “Buffer Insertion for Delay Minimization using An Improved PSO Algorithm,” Applied Mathematics & Information Sciences, An International Journal, pp. 2277-2285, 2014.

[87] R. Varada et al, “Superblock: A method for synthesizing Large High Performance Designs Without Hierarchy Limits”, DAC 2010.

[88] ”CPU World” [Online]. Available:http://www.cpu-world.com/CPUs/Bobcat/AMD-E%20Series%20E-350%20-%20EME350GBB22GT.html.

Appendix

A1 RodRunner library

In this dissertation, IBM's RodRunner based standard cell library was used to generate the layout for actual logic gates. This pcell-based gate layout generation is automatic and physical design error free, which reduces the cost of standard cell library design. Designers can choose different gates with different thresholds and power levels to design power efficient circuits. In addition, the resources required to create the full cell library were greatly reduced because layout for each cell can be generated and updated automatically. For RLM, the use of RodRunner cells allowed a very large library of standard cells to be created giving synthesis maximum flexibility. A key benefit to RodRunner cell was the ability to make any DRC or methodology updates in a single location within the RodRunner cell. This change is instantly picked up across all instantiations of the cell, including standard cell library [56].

A2 Congestion Analysis

At high level design, routing resources are carefully analyzed to build bottom-up chip. Macros, unit and chip share the routing layers and create contracts between all hierarchies and make sure designs are routable and timing closed. Routing at highly critical areas and interfaces must be analyzed first so that designers have freedom to upgrade or widen wires to make time. However, widening wires of course create a congestion problem for the router. Congestion has been discussed in chapter 2.7 and 2.8 and explained how it is important to floor-plan so that local and global congestion can be avoided by smart PIN and IP placement inside unit and MMLGS. Also, the congestion is a function of the aspect ratio of the macro, unit and chip. Thus, when it

comes to floor-plan, designer should carefully analyze the data flow of the design and come-up with smart routing to avoid congestion.

A2.1 Vertical Wiring Congestion

Vertical wiring congestion is a function of the width, PIN and internal IP placement of the macro/unit. In L2 cache unit routing, experiment, as expected, not much congestion was observed due to unit's aspect ratio and data flow structure, Fig. A.1. The green color represents the good routing results in the design. The yellow color is of concern in the design but as expected, those areas are due to the access to the custom arrays which are pre-placed. Thus, routings are not expected to change at these interfaces. The other colors, purple, also represents concern in the design because those vertical lines are pre-placed and customized with clock sector buffers placement. And thus, these columns are also expected to be highly densed.

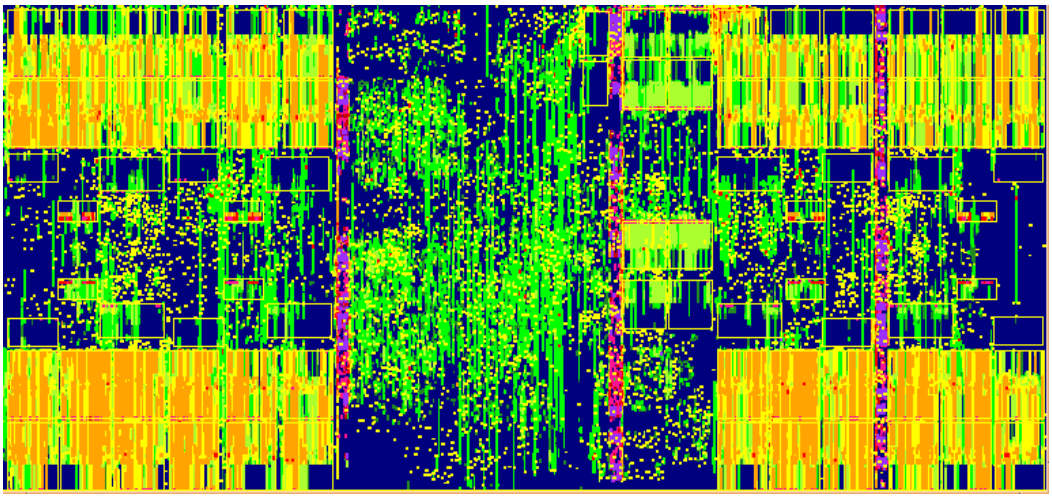


Fig. A.1: Vertical wiring solution.

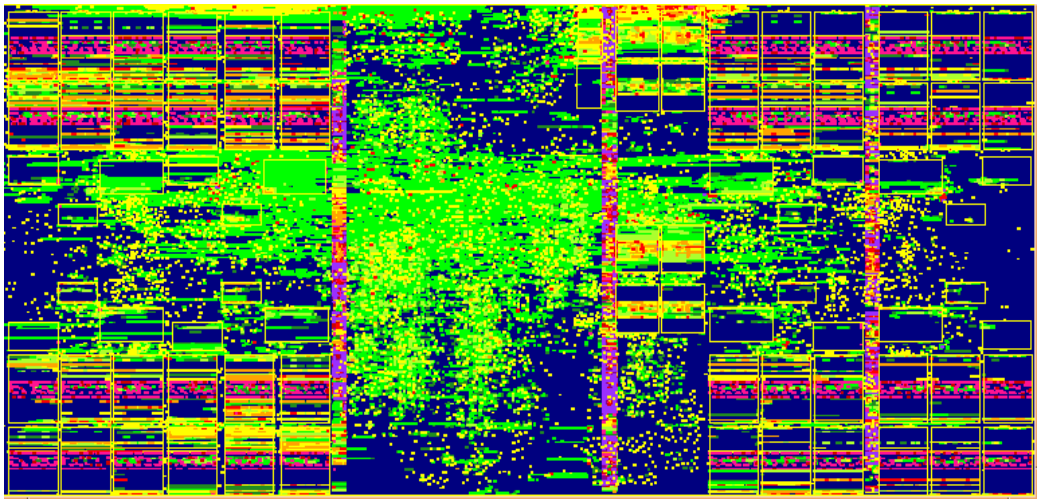


Fig. A.2: Horizontal wiring solution

A2.2 Horizontal Wiring Congestion

As oppose to vertical wiring congestion, in L2 cache design, horizontal congestion is of a concern due to aspec ration. As Fig. A.2 suggests, the test case see a fairly good number of places with higher horizontal wiring congestion, specially at array interface areas. Designers need to analyze the congestions at these areas and look at the routing resources available and make sure noise is not of a concern due to high congestion. Several technique such as wire traits, pre-placement of logic gates can be implemented in order to close timing on critical wiring areas as explain in section 5.3.

As previously stated, to make timing on these critical long horizontal wires, the following techniques were followed during routing the routing of L2 cache unit.

1) Keep wire delay low by using fat wires and redundant VIAs by reducing resistance.

2) Keep wire capacitance low by placing critical logic together and increase spacing from the adjacent aggressor wires. Also, route critical nets adjacent to wires that have different timing window or static signals such as power and ground.

3) Remove any dangling wires especially for multi fan out logic cones or share wires wherever congestion.

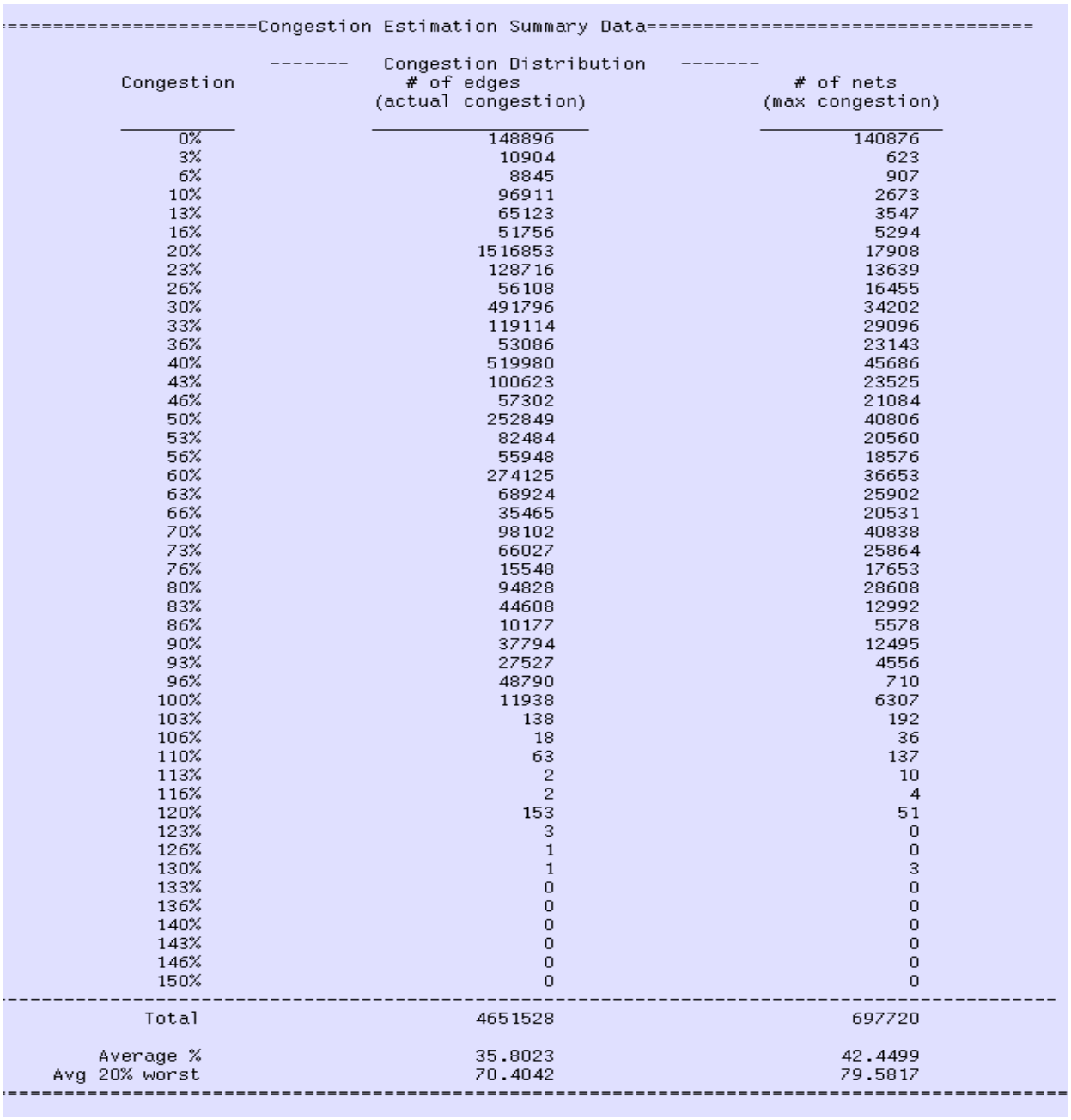
A3 Routing Analysis

Fig. A.3 and A.4 give us some statistical data about routing the L2 Cache unit. The more congested the design is, the more iterations are taken by the router to complete the job. And of course, the routing or compute time goes-up with the congestion of the design. Both routing and post-routing tool try to fix the DRC rules after initial routing and try to fix all during detail routing (droute). Fig. A.3 is an example of how router was able to connect all nets but left 1440 DRC violations in the design.

As circuit designers carefully evaluate the routing of the macros for better timing and congestion, it is highly relevant that the statistics of congested area is analyzed. As seen in Fig. A.4, most of the congested nets are in 40-60% area, while only about 380 nets are above 100% congested area. With these statistics, the routing for this unit does not look bad during congestion estimation. When it comes to incremental routings to fix noise and congestion, router see these numbers easily fixable.

```
Begin droute...
    Begin cycle 1...
        Begin Collection of shapes to route (unroutes and errors)...
        End Collection of shapes to route (unroutes and errors) 0.0s (kernel), 0.0s (user), 0.0s (elapsed), 55198.0MB vm, 55198.0MB peak vm
        [pass  1 index  1 begin] rips  6632  (unroutes   104  errors  6521  violatees     0  weak     7  offgrid     0 giveUp     0)
        [pass  1 end  ] 169.55 secs, 169.55 secs total
        [pass  2 index  2 begin] rips  5427  (unroutes   104  errors  4232  violatees  1091  weak     0  offgrid     0 giveUp     0)
        [pass  2 end  ] 148.51 secs, 318.06 secs total
        [pass  3 index  3 begin] rips  4149  (unroutes   104  errors  3085  violatees   960  weak     0  offgrid     0 giveUp     0)
        [pass  3 end  ] 158.83 secs, 476.89 secs total
        [pass  4 index  4 begin] rips  3216  (unroutes   104  errors  2546  violatees   566  weak     0  offgrid     0 giveUp     0)
        [pass  4 end  ] 187.52 secs, 664.41 secs total
        [pass  5 index  5 begin] rips  2655  (unroutes   104  errors  2167  violatees   384  weak     0  offgrid     0 giveUp     0)
        [pass  5 end  ] 397.31 secs, 1061.72 secs total
        [pass  6 index  6 begin] rips  2143  (unroutes   104  errors  1680  violatees   359  weak     0  offgrid     0 giveUp     0)
        [pass  6 end  ] 532.61 secs, 1594.33 secs total
        [pass  7 index  7 begin] rips  1902  (unroutes   104  errors  1528  violatees   270  weak     0  offgrid     0 giveUp     0)
        [pass  7 end  ] 407.93 secs, 2002.26 secs total
        [pass  8 index  8 begin] rips  1702  (unroutes   104  errors  1402  violatees   196  weak     0  offgrid     0 giveUp     0)
        [pass  8 end  ] 761.26 secs, 2763.52 secs total
        [pass  9 index  9 begin] rips  1729  (unroutes   104  errors  1403  violatees   222  weak     0  offgrid     0 giveUp     0)
        [pass  9 end  ] 360.00 secs, 3123.52 secs total
        [pass 10 index 10 begin] rips  1620  (unroutes   104  errors  1344  violatees   172  weak     0  offgrid     0 giveUp     0)
        [pass 10 end  ] 145.56 secs, 3269.08 secs total
        [pass 11 index  6 begin] rips  1581  (unroutes   104  errors  1300  violatees   177  weak     0  offgrid     0 giveUp     0)
        [pass 11 end  ] 99.45 secs, 3368.53 secs total
        [pass 12 index  7 begin] rips  1412  (unroutes     0  errors  1248  violatees   164  weak     0  offgrid     0 giveUp   104)
        [pass 12 end  ] 137.24 secs, 3505.77 secs total
        [pass 13 index  8 begin] rips  1421  (unroutes     0  errors  1261  violatees   160  weak     0  offgrid     0 giveUp   104)
        [pass 13 end  ] 4125.15 secs, 7630.92 secs total
        [pass 14 index  9 begin] rips  2020  (unroutes     0  errors  1462  violatees   558  weak     0  offgrid     0 giveUp   104)
        [pass 14 end  ] 16.05 secs, 7646.97 secs total
        [pass 15 index 10 begin] rips  1703  (unroutes     0  errors  1691  violatees    12  weak     0  offgrid     0 giveUp   105)
        [pass 15 end  ] 11.40 secs, 7658.37 secs total
        [pass 16 index  6 begin] rips  1682  (unroutes     0  errors  1677  violatees     5  weak     0  offgrid     0 giveUp   105)
        [pass 16 end  ] 23.74 secs, 7682.11 secs total
        [pass 17 index  7 begin] rips  1625  (unroutes     0  errors  1594  violatees    31  weak     0  offgrid     0 giveUp   105)
        [pass 17 end  ] 12.64 secs, 7694.75 secs total
        [pass 18 index  8 begin] rips  1594  (unroutes     0  errors  1580  violatees    14  weak     0  offgrid     0 giveUp   105)
        [pass 18 end  ] 5066.63 secs, 12761.38 secs total
        [pass 19 index  9 begin] rips  1909  (unroutes     0  errors  1457  violatees   452  weak     0  offgrid     0 giveUp   105)
        [pass 19 end  ] 18.18 secs, 12779.56 secs total
        [pass 20 index 10 begin] rips  1545  (unroutes     0  errors  1527  violatees    18  weak     0  offgrid     0 giveUp   105)
        [pass 20 end  ] 22.07 secs, 12801.63 secs total
        [pass 21 index  6 begin] rips  1514  (unroutes     0  errors  1508  violatees     6  weak     0  offgrid     0 giveUp   105)
        [pass 21 end  ] 22.35 secs, 12823.98 secs total
        [pass 22 index  7 begin] rips  1465  (unroutes     0  errors  1442  violatees    23  weak     0  offgrid     0 giveUp   105)
        [pass 22 end  ] 12.12 secs, 12836.10 secs total
        [pass 23 index  8 begin] rips  1452  (unroutes     0  errors  1439  violatees    13  weak     0  offgrid     0 giveUp   105)
        [pass 23 end  ] 9.64 secs, 12845.74 secs total
        [pass 24 index  9 begin] rips  1456  (unroutes     0  errors  1441  violatees    15  weak     0  offgrid     0 giveUp   105)
        [pass 24 end  ] 5.48 secs, 12851.22 secs total
        [pass 25 index 10 begin] rips  1447  (unroutes     0  errors  1440  violatees     7  weak     0  offgrid     0 giveUp   105)
        [pass 25 end  ] 4.10 secs, 12855.32 secs total
```

Fig. A.3: Routing iteration to fix design violation.



Congestion Estimation Summary Data

Congestion	Congestion Distribution: # of edges (actual congestion)	# of nets (max congestion)
0%	148896	140876
3%	10904	623
6%	8845	907
10%	96911	2673
13%	65123	3547
16%	51756	5294
20%	1516853	17908
23%	128716	13639
26%	56108	16455
30%	491796	34202
33%	119114	29096
36%	53086	23143
40%	519980	45686
43%	100623	23525
46%	57302	21084
50%	252849	40806
53%	82484	20560
56%	55948	18576
60%	274125	36653
63%	68924	25902
66%	35465	20531
70%	98102	40838
73%	66027	25864
76%	15548	17653
80%	94828	28608
83%	44608	12992
86%	10177	5578
90%	37794	12495
93%	27527	4556
96%	48790	710
100%	11938	6307
103%	138	192
106%	18	36
110%	63	137
113%	2	10
116%	2	4
120%	153	51
123%	3	0
126%	1	0
130%	1	3
133%	0	0
136%	0	0
140%	0	0
143%	0	0
146%	0	0
150%	0	0
Total	4651528	697720
Average %	35.8023	42.4499
Avg 20% worst	70.4042	79.5817

Fig. A.4: Congestion data in L2 cache unit.

A4 Distribution of Gate Array (GA) Cells

Another component of physical design is to look forward and plan on future design changes once the macro is built. These changes involve addition of new logic, change of existing logic, circuit design violation fixes etc. Circuit design part includes changes such as: addition of buffers, increasing drive strength, change logic topology to make timing, allocate better wire resources etc. With the future changes in mind, synthesis methodology would allow designers to add Engineering Change Order (ECO) gates which is also called Gate Array (GA) cells in the design, Fig. A5, so that during the automatic or manual ECO process, these GA cells could be used to implement desired changes in the design. These GA cells are laid out in such a way that they could be converted easily into basic functions such as inverter, NAND, NOR, AOI etc. Also, there are several sizes available in the design so that different strength of functional gates can be made from these GA cells. As far as location goes, synthesis tool place function gates first and fill out rest with GA cell even though there is another algorithm available to evenly distribute minimum % of GA cell all over the design and rest are in empty areas. Similar to GA cell, empty space is also filled with decoupling capacitors to compensate the droop in the design.

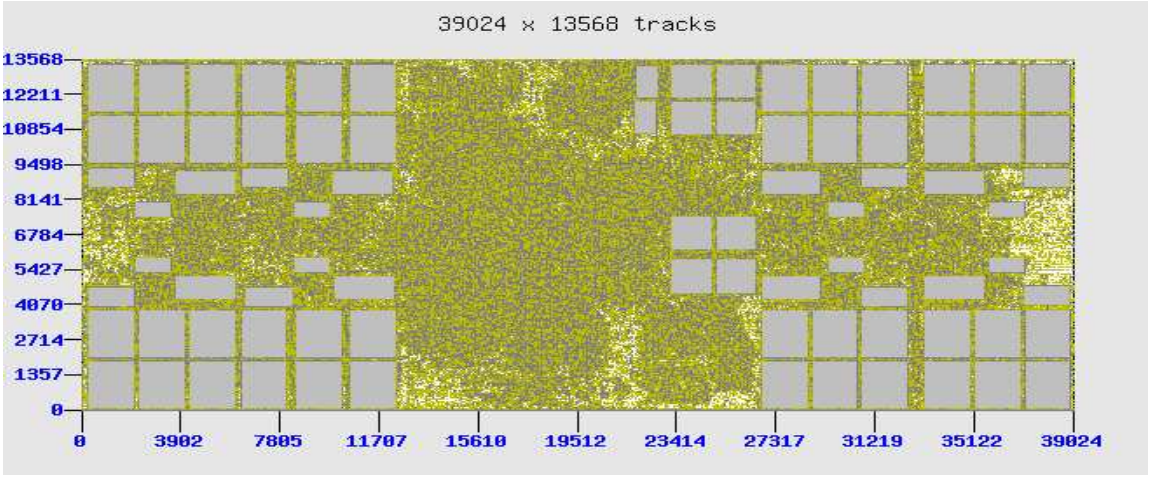


Fig. A.5: Distribution of GA cell after synthesis.

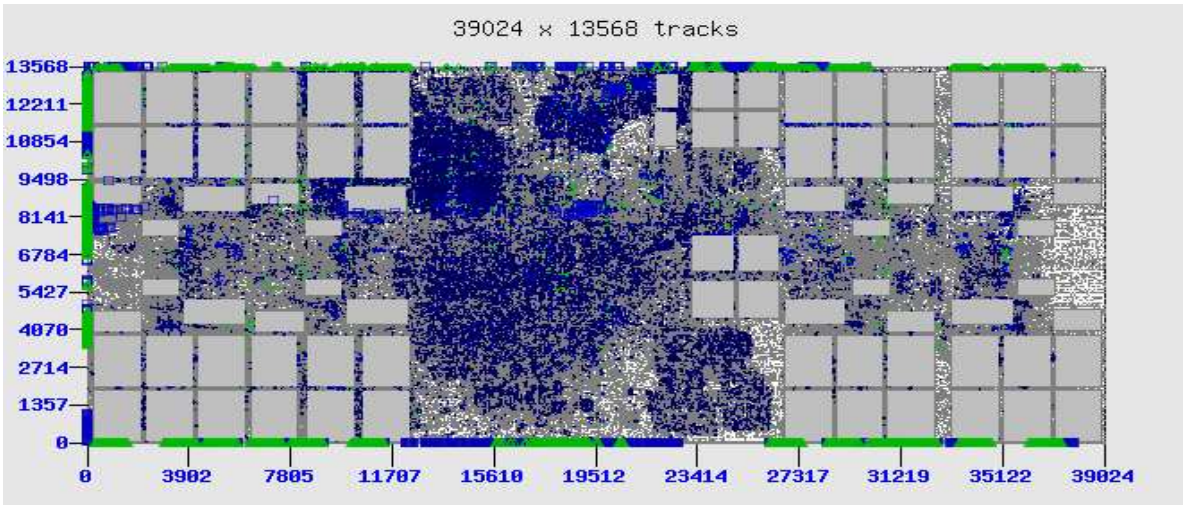


Fig. A.6: Distribution of negative slack in synthesis.

A5 Slack Distribution

Synthesis has an ability to write out the slack information in a pictorial view. Designer can easily open the output file, Fig. A6, and look at the most timing critical places and visually inspect them. They can look at these places where most of the negative/positive slacks are and can take advantages of logic and placement optimization.

A6 Post Routing Statistical Data

The uses of post-routing tool (Rapids) has been discussed in Chapter 6. This tool is basically used after routing the macro to fix the electrical violations such as slew, slack and some DRC (Design Rule Check). This tool does not move around and rip-off placed gates rather it work on changing the drive strength so that delay on critical paths are optimized. Some cases, tool does re-route some scenic paths to make it straight on all routing layers. Fig. A.7 shows a statistical data that was produced during 'Rapids' run on L2 cache units. Last column of the data shows % improve on several matric. For example: in this exercise, late mode L2L worst slack

(Latch to Latch) improves by ~20% and FOM (Figure Of Merit) changes by ~3%. This tool also work on power optimization because it can swap gates with low power when drive strength is needed.

```
RAPIDS Optimization Final Summary for /TECH/l2_top/:
 Valid Routing Layers           : "M2 M3 M4 M5 B1 B2 E1 E2 E3 X1 X2 X3"
                                     Initial    PostOpt      Final   %improve
-------------------------------------------------------------------------------
 Late worst slack               :    -89.413    -88.066    -89.962     -0.614
 Late L2L worst slack           :    -80.650    -50.310    -64.650     19.839
 Late FOM                       : -58888.293 -32505.939 -46613.676     20.844
 Late negatives                 :       2931       1868       2513     14.261
 Early worst slack              :   -404.800   -404.800   -404.195      0.149
 Early L2L worst slack          : -9999.000 -9999.000 -9999.000     -0.000
 Early FOM                      : -723969.937 -718067.625 -704219.937      2.728
 Early negatives                :       2381       2381       2381      0.000
 Electrical FOM                 :    177.000     72.000     90.000     49.153
 Slew violations                :       1374         34        750     45.415
 Load violations                :        246        225        251     -2.033
 Placement violations           :      35163       2169       2169     93.832
 Area (icells)                  :   17460647   17488594   17488594     -0.160
 Area Utilization               :      59.300     59.200     59.200      0.169
 Wire length (tracks)           : 345949639 345956855 345852925      0.049
 Wire length (microns)          :   30345768   30346509   30330759      0.049
 Guides wire length (microns)   :       3342       3342          0      0.000
 Nets with global routes        :          4       7787          4      0.000
 Global wire length (microns)   :         10      92906          7      2.580
 Scenic Nets                    :        122        104         60     62.000
 Route opens (clock)            :        433        433        176     59.353
 Route opens (signal)           :          2          1          0      0.000
 Route opens (pwrgnd)           :          0         18          0      0.000
 Route shorts (clock)           :          1        335        556 -55500.000
 Route shorts (signal)          :          9      93479        577 -6311.111
 Route shorts (pwrgnd)          :     737488     758781     737499     -0.001
 DRC errors                     :        786       6167       1303    -65.776
 LCB max cap viols              :         32          0          4     87.500
 LCB min cap viols              :         32          0          2     93.750
 Tiles Below 4% GA Density      :          0                     0      0.000
 Tiles Below 8% GA Density      :          0                     0      0.000
 Nets                           :     691358     694645     694645     -0.475
 Gates                          :     621338     624625     624625     -0.529
 Cells resized                  :                 46227      46227
 Cells added                    :                  3355       3355
 Cells moved                    :                  2031       2031
-------------------------------------------------------------------------------
```

Fig. A.7: Post routing tool improvement statistics.