"Structured ASIC: Evaluation and Implementation"

Major Project Report

Submitted in Partial Fulfilment of the Requirements for

Semester III-IV

of

Master of Technology

In

ELECTRONICS & COMMUNICATION ENGG.
(VLSI Design)

By :
Manish U.Patel
(05MEC012)

Under the Guidance of

Mr.Varun Jindal,
ST Microelectronics,
Noida.

Prof. N. P. Gajjar,
Nirma University,
Ahmedabad.

Department of Electronics & Communication Engineering
Institute of Technology
Nirma University of Science & Technology
Ahmedabad-382481

May 2007

CERTIFICATE

This is to certify that the M. Tech Dissertation Report entitle **"Structured ASIC : Evaluation and Implementation"** submitted by **Manish Patel (Roll no. 05MEC012)** towards the partial fulfillment of the requirements for the Sem III and Sem IV in **Master of Technology (Electronics and Communication)** in the Field of **VLSI Design**, **Nirma University of Science and Technology, Ahmedabad** at **S.T.Microelectronics, Noida** is the record of the work carried out under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for examination. The results embodied in this Dissertation-project work to the best of our knowledge have not been submitted to any other University or Institute for award of any degree or diploma.

Date:

Project Guide
Mr. Varun Jindal
ST Microelectronics,
Greater Noida.

Internal Project Guide
Prof. N.P.Gajjar
Institute of Technology
Nirma University,
Ahmedabad.

HOD
EE – Department,
Nirma University, Ahmedabad.

Director
Institute of Technology,
Ahmedabad.

ACKNOWLEDGEMENT

Any fruitful effort in a new work needs a direction and guiding hands that shows the way. It is proud privilege and pleasure to bring out indebtness and warm gratitude to respect **Ms. Namerita Khanna** (Project Manager –eCL Group) and **Mr.Varun Jindal** (Design Engineer), STMicroelectronics Pvt. Ltd. India for their support during my thesis work.

I would like to express my profound gratitude to **Mr. Kanwar Singh, Mr. Ajay Gautam** and all other team memebers for outstanding support and time to time guidance during my project work.

I would like to express my sincere thanks to my internal project guide **Prof. N.P.Gajjar** for his able guidance and timely help, his critical observations and encouragement that made successful completion of this work possible.

I would like to specially thank **Dr. N. M. Devashrayee**, P. G. Co-ordinator, M. Tech VLSI, Nirma University, Ahmedabad, for providing me with an apportunity to take up this training and for their constant support and encouragement.

Finally I would like to thank my class mates for their continuous support and constant encouragement during the project work. I would like to thank each and everyone who directly or indirectly helped me in the accomplishment of the project.

Finally I would like to thank **STMicroelectronics** for providing me with an opportunity to work with them and carry out a project of this importance.

Manish Patel
05MEC012
Nirma University

ABSTRACT

Structured ASIC is very latest concept in the field of Configurable Logic. It bridges the gap between FPGA and Standard Cell based Design hence having advantages of both. With an optimized and programmable structure, the structured ASIC technology indeed introduces a dramatically reduce ASIC cost and manufacturing turn-around time. The structured ASIC implementation flow is more complex than the conventional cell-based flow.

This project explore the differnt Structured ASIC architecture and Implement variuos designs on available Structured ASIC core like eASIC, ViASIC and LightSpeed hence customer can get best option for their Silicon Implementation.

This report compares eASIC, ViASIC and LIghtSpeed architecture goodness using experimental techniques. Comparison is made with respect to each other in terms of frequency, utilization and capacity.

As the design flow is complex various design issues encountered during the flow which may not occur in the ASIC flow. These issues are discussed and remedies for the same also proposed.

CONTENTS

List of Figures

List of Tables

Chapter 1: Introduction

1.Gate Array

Traditional gate-arrays used a "sea-of-gates" approach where pre-defined array of transistors is built on common base wafers that can be manufactured in advance and stored in a wafer bank. The transistors can then be interconnected using metal layers to implement a given design, which means that only the metal and VIA layer masks are specific to a given design and only the metalization steps need to be performed on the stored wafers.

This solution worked well with older process geometries, but as the industry approached the 0.35µ technology node and beyond, the advantage of shrinking transistors was lost due to the limitations on contact and metal pitch that limited the utilization of the array. Using Gate Arrays could no longer save any significant cost or time. Hence, for many of today's applications, gate-array technology is not a viable option.

2.Standard ASICs

In the case of ASICs – of which the currently dominant form is that of *standard cell (SC)* devices – these are extremely expensive and time-consuming to develop. As IC implementation technologies move into the *ultra-deepsubmicron (USDM)* realm (specifically the 90 nanometer node and below), power, timing, and signal integrity issues become evermore complex. Reaching closure on these issues takes so much effort that the design team now spends more time addressing these aspects of the design than they spend architecting, capturing, and verifying the logical functionality of the device. In addition to protracted development times, the photomasks associated with a new ASIC are becoming prohibitively expensive (in the order of $1 million for a reasonably complex 90 nanometer device). Furthermore, the manufacturing turnaround time to actually fabricate these devices significantly impacts their time-to-market. The long development and manufacturing times associated with standard cell ASICs pose particular problems with regard to today's short product life cycles and the need to address constantly evolving standards and protocols. However, these devices do have the advantages that they can be used to implement the largest, most complex, high-performance designs. They also have a low per-unit cost when used in large production runs in the order of 50,000 units or more.

3.FPGAs

Today's state-of-the-art FPGAs can provide up to 10M system gates, which – depending on the application – equates to somewhere between 1M to 3M ASIC equivalent gates. The addition of features such as embedded RAM, embedded processor cores, and gigabit transceivers means that FPGAs can now be used to implement reasonably large and sophisticated designs (although the largest and most complex designs remain the domain of ASICs). FPGAs are fully prefabricated by the vendor, which means that there is no manufacturing turnaround time to be accounted for in the design cycle. Furthermore, creating an FPGA design is, in many respects, simpler and faster than would be its ASIC counterpart. This is because considerations such as signal integrity have already been addressed by the device's manufacturer and/or are automatically handled by the FPGA's design tools (in both cases this occurs transparently to the end user). The fact that many FPGA families can be reconFig.d (reprogrammed) means that they are ideal for use with applications whose standards and protocols are constantly evolving. However, FPGAs also have significant disadvantages, in that their designs consume significantly more power and have much lower performance than equivalent ASIC implementations. Furthermore, FPGAs have a high per-unit cost, which makes them an extremely expensive option for anything other than prototyping applications or relatively small production runs.

The requirement

All of the above points serve to illustrate that there is a huge gap between the two main technologies currently used to implement custom digital IC designs (*Fig. 1*).

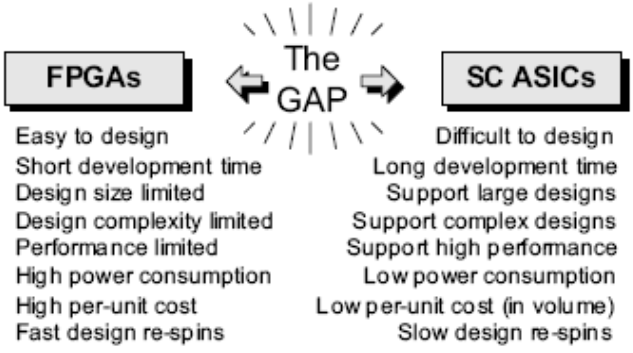


Fig 1: FPGA-ASIC gap

The rising cost of developing standard cell ASICs means that many companies can no longer afford to use these devices. At the same time, FPGAs aren't appropriate for many of these designs due to capacity and performance issues and/or high per-unit costs. What is required is a

new implementation technology that overcomes the design size, complexity, performance, and power consumption limitations of FPGAs, but which also addresses the long development times, high development costs, and long manufacturing lead times associated with standard cell FPGAs. In addition, this new technology should offer a reasonably low per-unit cost, thereby making these components suitable for medium-size production runs *(refer fig 2*). The solution may well be a new class of devices known as *structured ASICs (SAs)*. This chapter introduces the concept of structured ASICs along with some comparisons between standard cell, structured ASIC, and FPGA implementations. Also provided is an overview of some of the alternative structured ASIC architectures tht are currently being made available to the market.

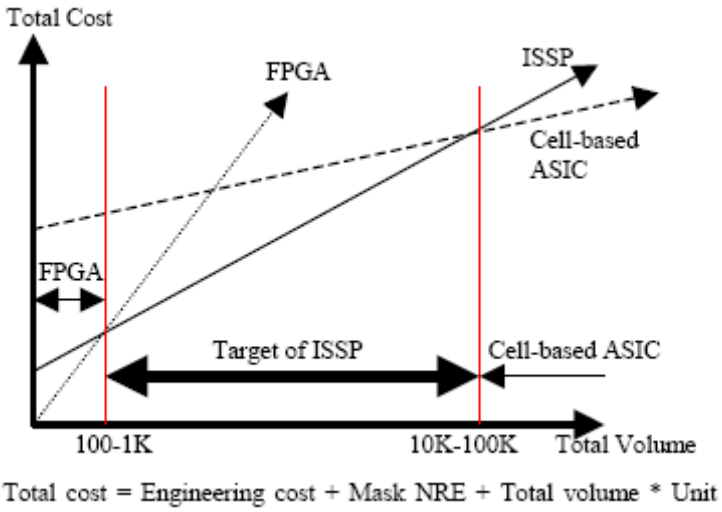


Fig 2: Total volume vs. total cost

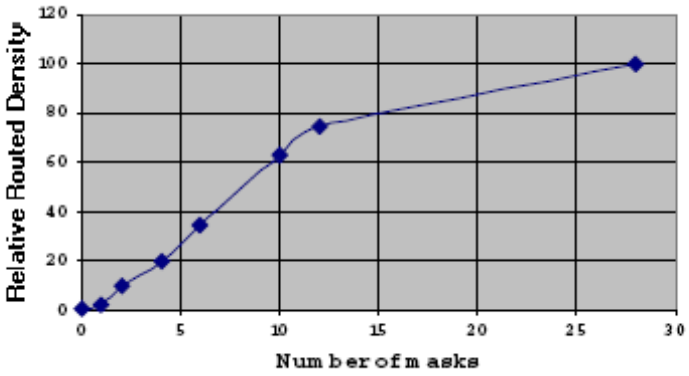


Fig 3: Density vs. Masks.

4. The structured ASIC concept

The underlying concept behind structured ASICs is actually fairly simple. Although there are a wide variety of alternative architectures, they are all based on a fundamental element called a “tile” by some or a “module” by others (this paper will use the term “tile” henceforth). This tile contains a small amount of generic logic implemented either as gates and/or multiplexers and/or a lookup table. Depending on the particular architecture (see also the discussions below), the tile may contain one or more registers and possibly a very small amount of local RAM. An array (sea) of these tiles is then prefabricated across the face of the chip. Structured ASICs also typically contain additional prefabricated elements, which may include configurable general-purpose I/O, microprocessor cores, gigabit transceivers, embedded (block) RAM, and so forth Structured ASIC technology is especially suitable for platform ASIC designs that have integrated most of the IP blocks and leave some space for customer function changes or bugs fixing in the future. A structured ASIC is different from the traditional cell-based ASIC. It contains an array of well-structured and optimized elements across the entire chip. The structured element is designed for implementing the desired functional by making changes to fewer upper layer mask. The structured ASIC FPGA-based products uniquely address the economic issues for lower volume applications by being easy to design and program within the shortest possible time. However, FPGAs consume more power, are much lower in performance, and come with substantially higher units costs when compare to cell-based ASIC.

In many respects these devices are similar to modern, high-end gate array ASICs. The key differentiator with regard to Structured ASICs is that the majority of the metallization layers are also prefabricated. This means that the transistors forming the core logical functions comprising each tile (gates, multiplexers, etc) are already wired together. Also, much of the local and global interconnect has also been implemented. Depending on the architecture, the design engineers need specify only one, two, or very few metallization layers in order to complete the device.

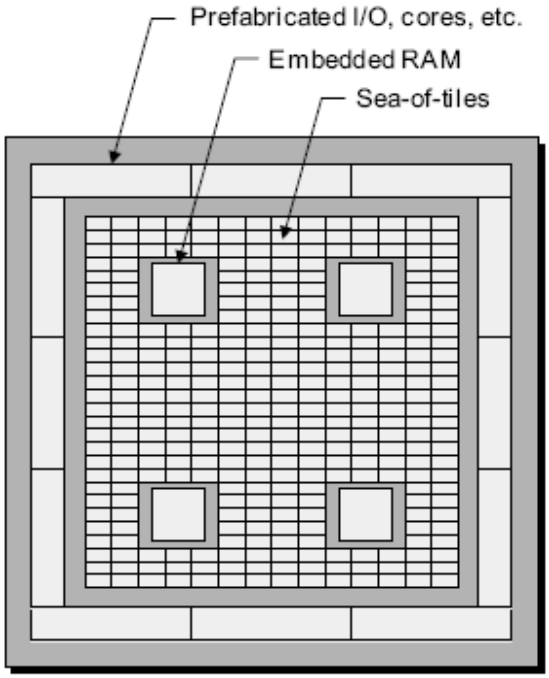


Fig 4: The structured ASIC concept

4.1 Advantages

- These devices are much easier and faster.
- Multiple global and local clock domains are typically prefabricated in the master fabric and are implemented in such a way that there are no skew problems that need be addressed by the design engineers.
- Design-for-test considerations are addressed by the fact that functions such as boundary scan (JTAG), full internal scan, and BIST are all typically embedded in the basic fabric.
- Due to the fact that structured ASICs need only a limited number of metallization layers to complete them, the costs associated with generating the photo-masks are dramatically reduced.
- Device is largely prefabricated radically shrinks the turnaround time to working silicon. This also means that structured ASICs can undergo faster and cheaper modification cycles in order to accommodate evolving standards and protocols.

- Overall, the capacity, performance, and power consumption of a structured ASIC is much closer to that of a standard cell realization of the design as opposed to an FPGA implementation.
- Additionally, the faster design time, lower mask costs, and quicker turnaround to final silicon – along with the lower costs resulting from the fact that the majority of the device is pre-fabricated – means that the per-unit cost of structured ASICs is extremely reasonable for medium-low to medium-high production runs.

4.2 Disadvantages

- The current design tools – which are currently predominantly based on traditional ASIC offerings – are both expensive and not well-suited to the task.
- The diverse architectures fielded by the various vendors are so new that they have not yet been subject to any form of formal evaluation and comparative analysis
- Chip area/performance overhead

  The area per unit gate would be 3 to 1.5 times in comparison with the conventional standard cell design; the average performance degrade would be 10% to 50%. It depends on the architectures.

4.3 The Architectures of Structured ASICs

The structured ASIC architecture mainly consists of two parts:

1. Structured element,
2. Array of structured element.

The general architecture of structured ASIC contains an array of structured elements. The structured element is similar to the gate array due to its well-structured nature. Gate array also contains many of gates across the entire chip. This gate is basically an array of uncommitted MOS transistors. The difference between gate array and structured ASIC is that the structured ASIC offers an array of partially formed elements. These elements are fully optimized in terms of area and performance. The structured element is also similar to the FPGA programmable element but does not have the additional overhead required for field programmability. There may exist one or more types of structured elements in the chip. These different types of elements form the array in the Structured ASIC. Each type of elements is activated for different purposes. For example, one type of element is for combination logic, and another type is for sequential element.

Fig. 5 illustrates the general architecture of the structured ASIC. There are 2 types of structured element. The first one is the logical element, and the other is the storage element. The different structured elements can be different or the same size. Logical element Storage element

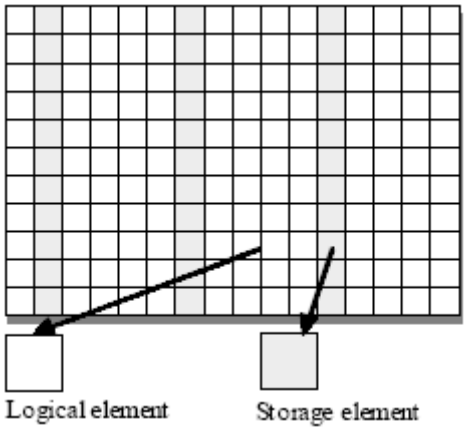


Fig 5: The structured ASIC general architecture

There are 2 types of structured elements.

1. Based on look-up tables,
2. Based on transmission gate.

Look-up table style

Each structured element is designed as look-up table to implement logical function. The look-up table architecture is widely used in FPAG logic block design. The structured element can be programmed any *n*-input logical function, where *n* is the inputs
of the structured element. Typically, the value of *n* is less than 5 for the circuit performance issue.

Transmission gate style

Each structured element implements the logical function from generic transmission gates. Disadvantages to use transmission gate style are the restricted provided logical function in comparison with look-up table style. On the other hand, the transmission gate style can provide a substantially lower performance unit cost.

There are 2 models of array:

1. Uniform array styles.
2. Non-uniform array styles.

Uniform array model

The uniform array model (*refer fig 6*) is setup as a regular grid array over the entire chip. There is one type of the structured element. That is, the structured element can be programmed as a logical element and sequential element. A typical uniform array is shown in Fig. 2. All structured elements are uniform.

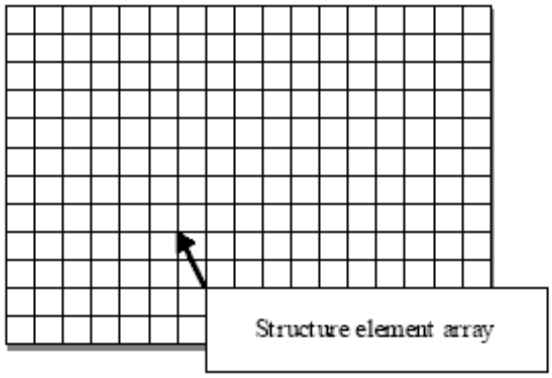


Fig 6: Uniform array model

Non-uniform array model

In non-uniform model, there are two more types of the structured element. Each type element has to be allocated on the specific locations on the chip. The specific locations have to be designed before the structured ASIC implementation. Fig. 3 shows a typical non-uniform array model. There are 3 types of structured elements: FE, BUF and FF. The ratio of the number of different types is 4:2:1. The FE element is for logical function implementation, BUF element is a buffer for timing optimization, and FF element is for sequential function.

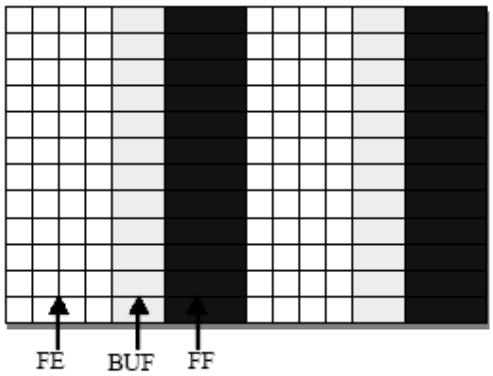


Fig 7: Non-uniform array model

Alternative SA architectures

This is a somewhat "gray" area, because the majority of vendors with structured ASIC offerings are still working in "stealth mode," which means that detailed descriptions of their internal architectures are not readily available. Thus, the following architectural descriptions are "composites" that have been gleaned from a variety of sources. In addition to its own unique version of a basic tile, each vendor offers its own selection of hard, firm, and soft IP. Hard IP comes in the form of configurable I/O blocks that can be modified (via the user-definable metallization layers) to handle a variety of standard I/O interfaces. Other hard IP blocks include standard interfaces like PCI, gigabit transceivers, microprocessor cores, embedded RAM, and so forth. Each vendor may offer a family of devices containing different combinations of hard IP blocks combined with various quantities of basic tiles. Firm IP comes in the form of a library of high-level functions that have been optimally mapped, placed, and routed for this vendors particular architecture, while soft IP is presented as a source-level library of high-level functions that can be included into the users' designs. In many cases the hard, firm, and soft IP from the various vendors are simply variations on a theme. The real differentiator between devices comes in the contents and architecture of the basic tile.

Extremely fine-grained

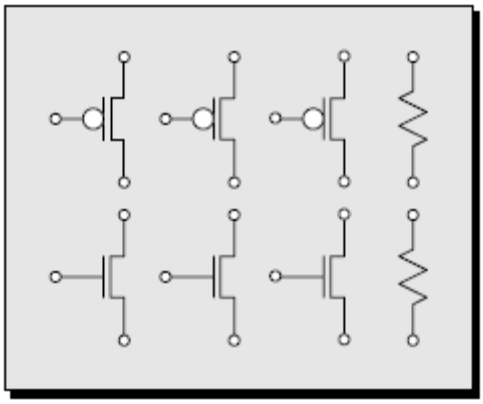


Fig 8: An extremely fine-grained tile

These architectures are extremely close to those of modern high-end gate array devices. The difference being that – in the case of the structured ASIC, metallization has been added so as

to almost connect these components in a variety of pre-defined configurations. Thus, the userdefinable metallization layers are used to complete the appropriate connections, and to link the tiles into the local and global routing architecture.

Medium-grained tiles

Other vendors have opted for a medium-grained architecture. In this case, the tile might contain some generic logic in the form of gates and/or multiplexers along with one or more flip-flops (Fig. 9).

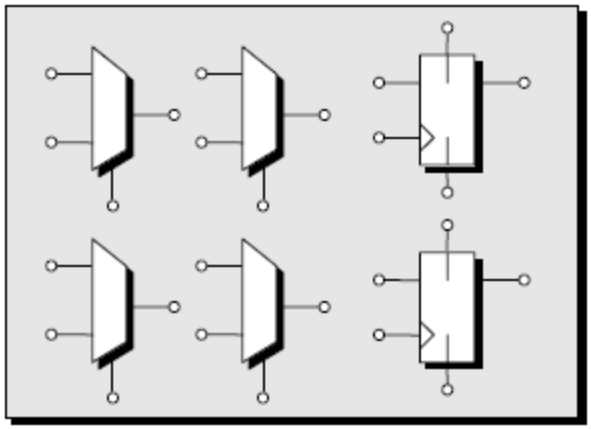


Fig 9: Mux-based medium-grained tile

Alternatively, some medium-grained architecture is based on tiles containing one or more lookup tables (LUTs) along with one or more flip-flops (*Fig. 10*).

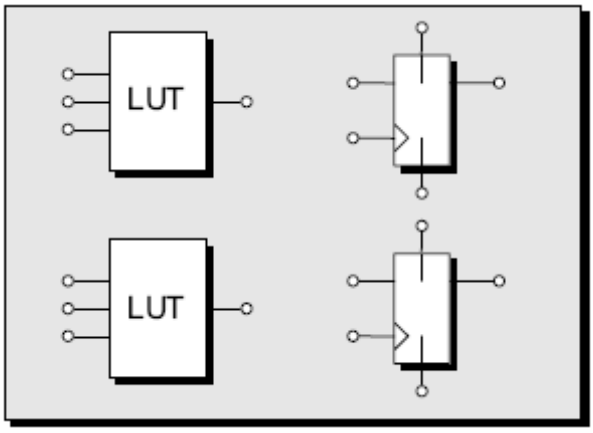


Fig 10: LUT-based medium-grained tile

In both of these cases the polarity of the flip-flops' clock inputs (i.e., whether each register should be positive- or negative-edge-triggered) and the polarity of their set and reset inputs can be determined by the customized metallization layers.

Hierarchical tiles

As yet another alternative, some architecture commence with a *base tile* containing only generic logic in the form of prefabricated gates and/or multiplexers and/or lookup tables (Fig. 6).

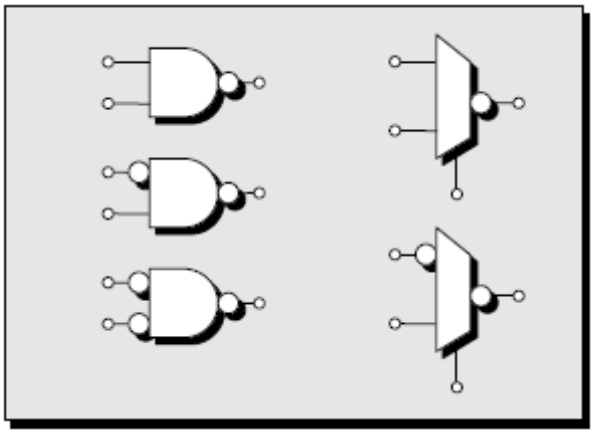


Fig 11: An example "base tile"

An array of these base tiles (say 4 x 4, or 8 x 8, or 16 x 16) are combined with special tiles containing registers, memory elements, and other logic to form a *master tile*, then an array (sea) of these master tiles is prefabricated across the face of the chip.

Fine versus medium versus coarse

One consideration with regard to the granularity of the architecture is that fine-grained implementations require a lot of connections into and out of each tile compared to the amount of functionality that can be supported by the time. By comparison, as the granularity of the tile increases to medium-grained and higher, the amount of connections into the tile compared to the functionality it can support decreases.

4.4 Key Market Players

Standalone Structured ASICs

Altera has HardCopy and HardCopy 2. These are meant to be designed using Altera FPGA and then converted to their structured products.

Embedded Structured ASICs

eASIC has the most unique product. The interconnects and routing are done by single mask customization. The logic section is handled by Look Up Tables. This part of design is field-reprogrammable and is loaded at power up.

ViASIC uses simple logic gates (NAND, MUX) to map the design into, while connections are done by single via layer customization

LightSpeed uses pre-optimized, pre-characterized & pre-qualified standard cells and customize througth 2M2V to all Metal layers

4.5 Comparision

	FPGA	Structured ASIC	Standard Cell Based ASIC
Performance	**Low**	**High**	**Very High**
Cost of re-Spin	**Low**	**Very High**	**Very High**
Time to Market	**Low**	**Medium**	**Very High**
Power Consumption	**Very High**	**Low**	**Low**
Complexity	**Low**	**Low**	**High**
Density	**Low**	**High**	**Very High**
Unit Cost	**Depends on Volume**		

Table 1: Comparision between FPGA, Str ASIC and Standard ASIC

Area

Keeping this in mind, it is generally accepted that standard cell architectures can support an equivalent gate density of approximately 100,000 gates/mm2, while FPGAs can only offer around 1,000 gates/mm2, which is a factor of 100:1. By comparison, some structured ASIC architectures are rumored to support around 33,000 gates/mm2, which is a factor of 3:1 compared to standard cells. That is, a structured ASIC can support 0.33x the number of gates as a standard cell device and 33x the number of gates in an FPGA component in the same area.

Performance

With regard to performance, if the same design is implemented in standard cell and FPGA devices, it is typically the case that the FPGA can only achieve 10% to 20% of the performance if the standard cell implementation (in terms of clock frequency). By comparison, early results on structured ASICs suggest that these devices can achieve 70% to 80% of the performance of a

standard cell implementation. In the case of power, FPGAs typically dissipate 10x to 15x that of an equivalent standard cell implementation.

Power Consumpsion

Once again, early results on structured ASICs suggest that these devices consume only 2x to 3x the power of their standard cell counterparts. Some additional metrics that are being quoted although not referenced) is that the development costs of a structured ASIC design are only 25% those of a standard cell equivalent. Furthermore, the production unit price of a structured ASIC is said to be only 10% of an equivalent FPGA (assuming that an FPGA can meet the design's size, complexity, and performance requirements).

	FPGA	Structured ASIC	Cell-based ASIC
Total Design Cost	$165K	$500K	$5.5M
Vendor NRE	None	$100K~$200K	$1M~$3M
Cost of EDA Tools	$30K	$120K~$250K	> $300K
Man Power	1 to 2	2 to 3	5 to 7
Price per Chip	$200 ~ $1K	$30 ~ $150	$30
Unit Cost (Qty 1K)	$1000	$500~$650	$55K
Unit Cost (Qty 5K)	$220	$110~$150	$1.1K
Unit Cost (Qty 500K)	$40	$21	$11
Source: Semiview, December 2003			

Table 2: FPGA, structured ASIC, and cell-based ASIC development costs

4.6 The Implementation Flow

The implementation flow for structured ASICs involves mainly 6 steps, which include logical synthesis, DFT insertion, placement, physical synthesis, clock tree synthesis and routing. Fig. 4 shows the structured ASIC implementation flow. The logical synthesis, which maps RTL design into structured elements, is the

DFT insertion adds the scan circuitry to improve the testability and fault coverage. Placement involves the mapping the smaller structured elements onto array elements.

Physical synthesis improves the timing by placement-based optimization. Clock tree synthesis distributes the clock network to minimize the clock skew and delay. The final step is routing.

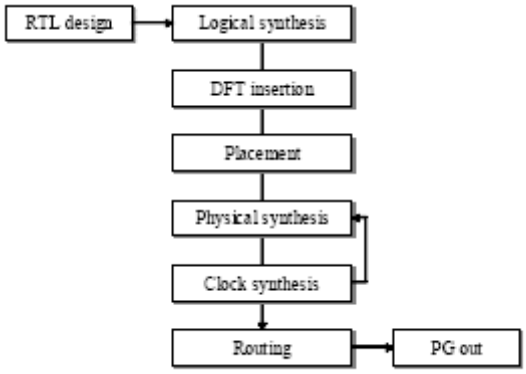


Fig 12: Implementation flow of structured ASICs

Logical synthesis, placement and routing problems in structured ASICs are slightly more complex than logical synthesis problems in other design styles. The other DFT insertion, physical synthesis and clock tree synthesis problems are the similar to the conventional ones.

Chapter 2: Basics Terminilogy and ASIC Flow

1. Generelize ASIC FLOW

Figure 1 shows the simple and generalize ASIC flow. In the following session we will understand terminologies and proicess associate with it.

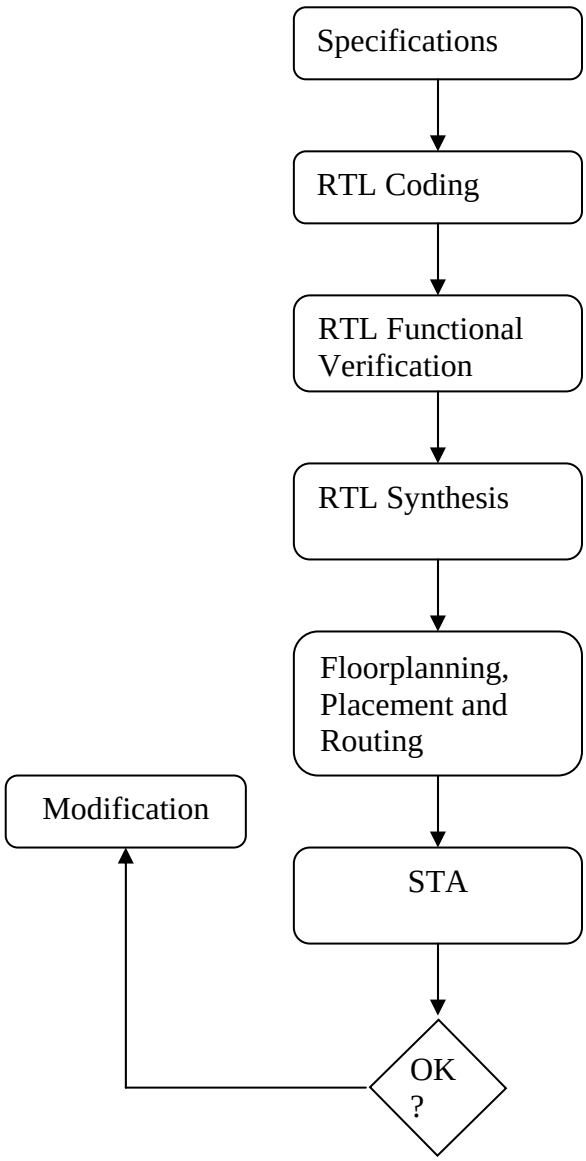


Fig 1: Generalize ASIC Flow

1.1 Specification

The first step in a high-level design flow is the design specification process. This process involves specifying the behavior expected of the final design. The designer puts enough detail into the specification so that the design can be built. The specification is usually written in the designer's native language and specifies the expected function and behavior of the design using textual description and graphic elements.

1.2 RTL Coding

After the specification has been completed, the designer or designers can begin the process of implementation. Some design teams create a high- level behavioral or algorithmic description of the design to verify design intent, then convert that description to RTL (Register Transfer Level) later. However, most design teams skip the behavioral description and implement the RTL directly.

1.3 RTL Functional Verification

The RTL Functional Verification step is used to verify the correctness of the RTL VHDL description. The designer has described the clock-by-clock behavior of the design. Now, the designer uses stimulus that represents the design environment to drive the design and check to make sure that the results are correct. A standard VHDL simulator can be used to read the RTL VHDL description and verify the correctness of the design.

1.4 Synthesis

Definition

It is the process to convert RTL code to the gate level netlist.

It also optimize the netlist and this generated netlist contain components which are specified in the library.It can be area driven, power driven or timing driven.

1.5 Floorplanning

Definition:

Floorplannig is the placement of flexible blocks that is block of fix area and but unknown dimensions.it is much more difficult than the placement. Here several layout alternatives for each block considered.

Floorplanning is the process of:

- Positioning blocks on the die or within another block, thereby defining routing areas between them
- Creating and developing a physical model of the design in the form of an initial optimized layout because floorplanning significantly affects circuit timing and performance, especially for complex hierarchical designs, the quality of your floorplan directly affects the quality of your final design.

You have refined your floorplan and before you continue on to detailed cell placement and detailed routing.

1.6 Placement

Definition:

In the placement placement phase blocks are assign a specific shape and are positioned on a layout surface in such a fatshion that no two blocks are overlapping and enough space is left on layout to complete the interconnections.the blocks are position such that it will take minimum area.

Fixed blocks the blocks for which dimensions are known and blocks for which dimensions are yet to determine called as flexible blocks.Problem of assingning locations to fix block called as placement problems. If some or all blocks are flexible than problem called as floorplanning problem Standard cell placement and physical optimization occurs during the fix cell stage.

Placement stages include:

Global placement

Global placement distributes the cells uniformly across the available core area, minimizing wire length and ensuring constant delay. The global placer uses a gain-based wire delay model, in which the cell area (and not wire capacitance) affects timing. After achieving an initial placement, you can incrementally run the global placer (for designs with greater timing considerations than congestion considerations).

Coarse placement

Coarse placement can be used to refine the preliminary placement of a design that has greater congestion considerations than timing considerations. It adjusts the placement to

distribute the standard cells evenly among the sea of buckets, while minimizing wire length. Coarse placement does not eliminate cell overlap or place cells precisely into rows.

1.7 STA

Definition:

Static Timing Analysis is a method for determining if a circuit meets timing constraints without having to simulate

A static timing analyzer traces each path in the design and keeps track of the timing from a clock edge or an input. A timing report is then generated in a number of formats. For instance, the designer can ask for all paths and get an enormous listing of every path in the design.

1.8 Routing

Definition:

The process of finding the goematric layout of all nets is called routing.

Global routing generates ‘loose’ route for each net.in detailed routing actual geometric layout of net assigned.

It handles crosstalk avoidance, electromigration issues, and constrained wire patterns while ensuring that all design spacing requirements are followed. If design rule violations occur, the router uses rip-up and reroute methods to repair them. Design rule violations that cannot be repaired are reported. There are five routing stages, in the following order:

Stub routing

The first step in signal routing. During stub routing, the detailed router identifies very short nets and routes them on the Metal1 layer only. The purpose of stub routing is to utilize the otherwise unused Metal1 resources and to avoid inaccuracy during the global routing of short nets.

Global routing

Focuses on resolving congestion and timing issues. The global router creates net segments and defines the initial bucket-level routing topologies and spreads wire density evenly over the design by assigning each *net segment* to buckets and layers.

Track routing

Track routing also works on the bucket level. It orders and spaces the net segments and assigns them to legal track positions. Proper ordering and spacing greatly reduces crosstalk coupling and noise. The track router does not adjust the topologies set by the global router; therefore, the congestion picture predicted by the global router is not altered. Track routing processes the layout, within buckets, row by row or column by column to assign the routing segments generated by the global router to legal track positions, define segment order and spacing, and resolve alignment, crosstalk

Detailed routing

It analyzes the net-segment topologies, converts the segments to actual wires and vias that connect all pins of all nets, and begins the process of correcting design rule violations The detailed router is region-based. It reads the content of a general rectangular region containing several buckets, performs ripup-and-reroutes, and updates the region when it accepts the resulting routing (or leaves it unchanged if unacceptable).

2. Detailed ASIC Flow

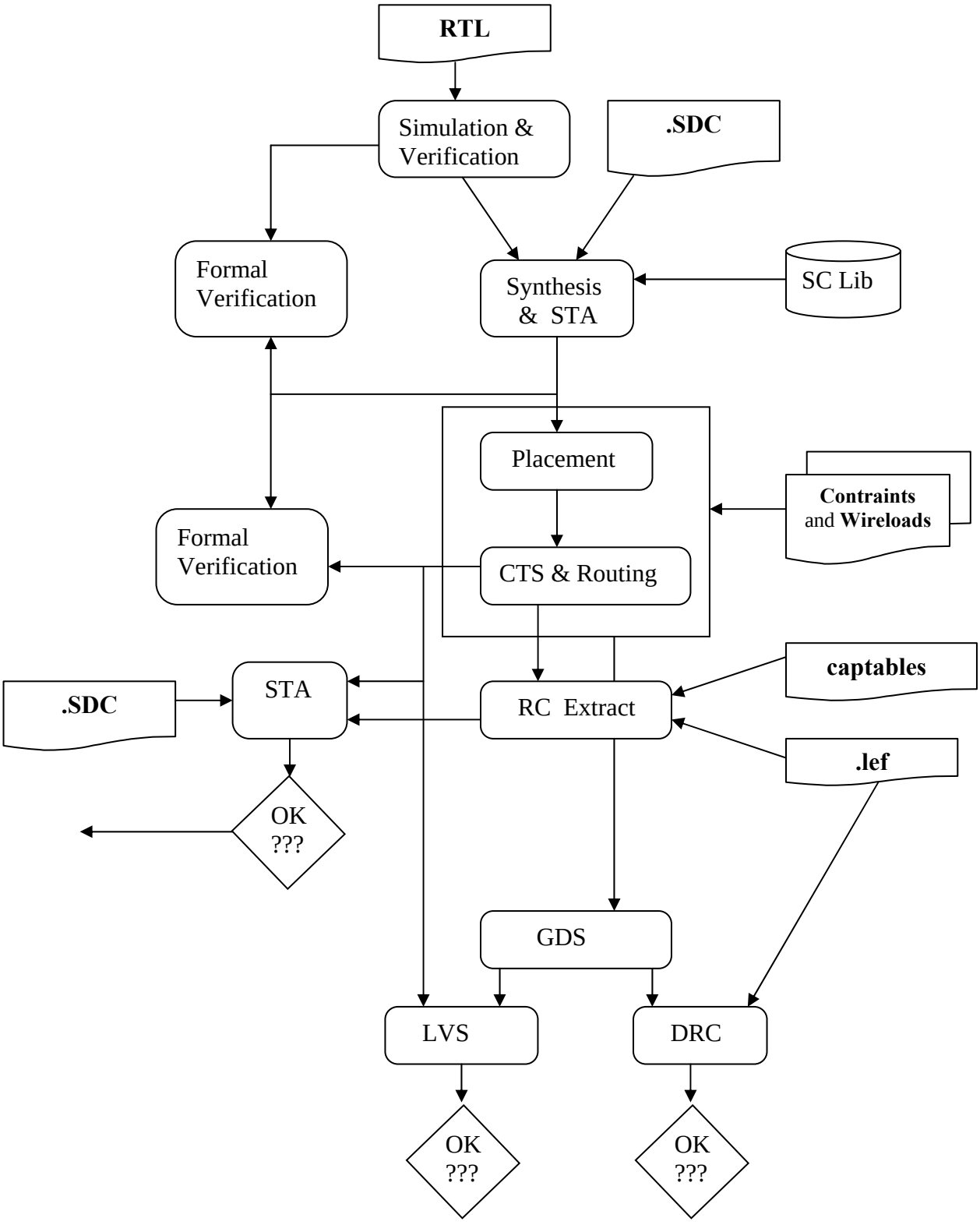


Fig 2: Detailed ASIC Flow

2.1 Synthesis

Synthesis is an automatic method of converting a higher level of abstraction to a lower level of abstraction. There are several synthesis tools available currently, including commercial as well as university-developed tools. The current synthesis tools available today convert Register Transfer Level (RTL) descriptions to gate level netlists. These gate level netlists consist of interconnected gate level macro cells. Models for the gate level cells are contained in technology libraries for each type of technology supported.

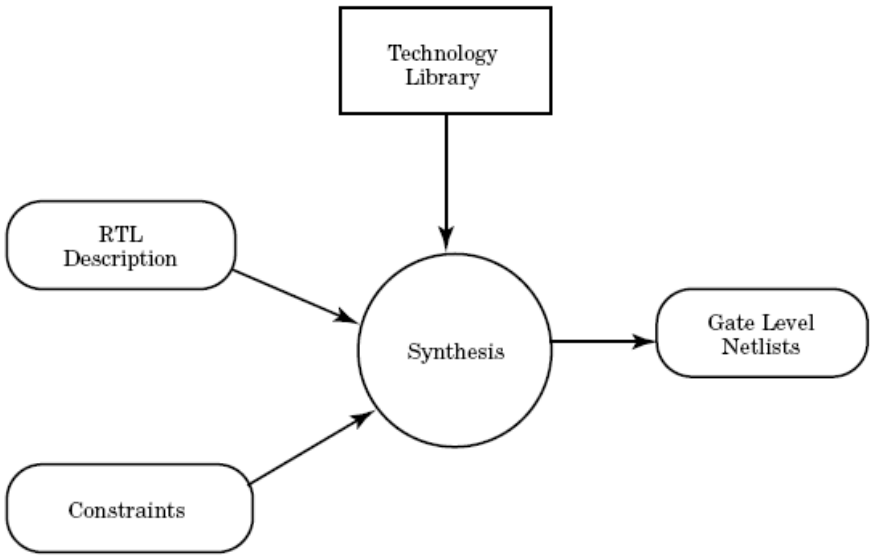


Fig 3: Synthesis environment

These gate level netlists currently can be optimized for area, speed, testability, and so on. The synthesis process is shown in *Figure 3*. The inputs to the synthesis process are an RTL (Register Transfer Level) VHDL description, circuit constraints and attributes for the design, and a technology library. The synthesis process produces an optimized gate level netlist from all of these inputs. In the next few sections, each of these inputs is described, and we discuss the synthesis process in more detail.

Constraints

Constraints are used to control the output of the optimization and mapping process. They provide goals that the optimization and mapping processes try to meet and control the structural implementation of the design. They represent part of the physical environment that the design has to interface with. The constraints available in synthesis tools today include area, timing, power, and testability constraints. In the future, we will probably see packaging constraints, layout constraints, and so on. Today, the most common constraints in use are timing constraints. A block diagram of a design with some possible constraints is shown in *Figure 4*. Again, the design is shown using the cloud notation. The combinationl logic between registers is represented as clouds, with wires going in and out representing the interconnection to the registers. There are a number of constraints shown on the diagram including required time constraints, late arrival constraints, and clock cycle constraints.

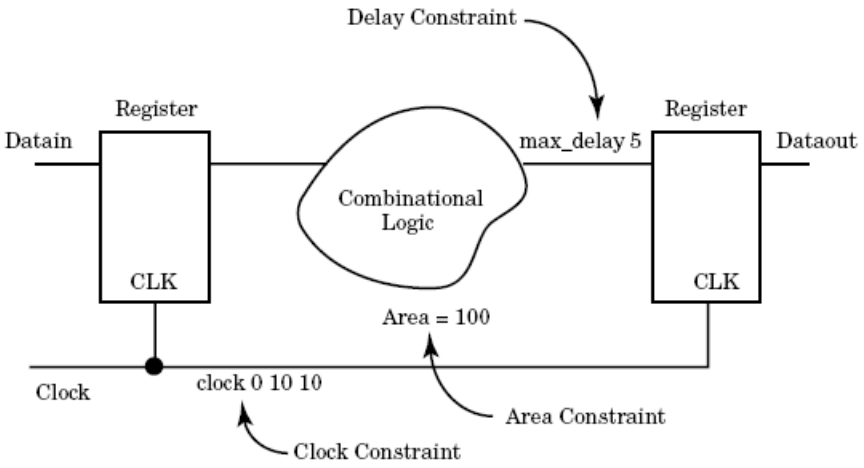


Fig 4: Constraints in Synthesis process

1. Timing Constraints

Typical uses for timing constraints are to specify maximum delays for particular paths in a design. For instance, a typical timing constraint [*refer figure 4*] is the required time for an output port. The timing constraint guides the optimization and mapping to produce a netlist that meets the timing constraint. Meeting timing is usually one of the most difficult tasks when

designing an ASIC or FPGA using synthesis tools. There may be no design that meets the timing constraints specified.

2. Clock Constraints

One method to constrain a design is to add a required_time constraint to every flip-flop input with the value of a clock cycle. The resulting design would be optimized to meet the one clock cycle timing constraint. An easier method, however, is to add a clock constraint to the design. A clock constraint [*refer figure 4*] effectively adds an input required_time constraint to every flip-flop data input.

Attributes

Attributes [*refer figure 5*] are used to specify the design environment. For instance, attributes specify the loading that output devices have to drive, the drive capability of devices driving the design, and timing of input signals. All of this information is taken into account by the static timing analyzer to calculate the timing through the circuit paths.

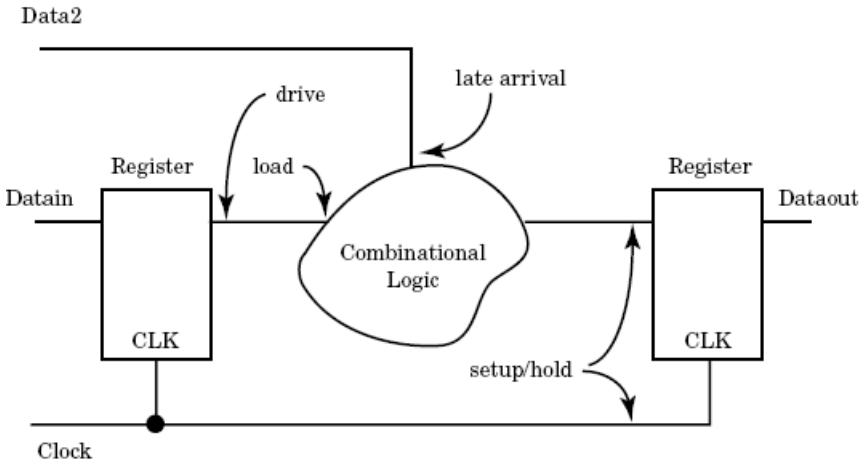


Fig 5: Attributes in Synthesis process

1. Load

Each output can specify a drive capability that determines how many loads can be driven within a particular time. Each input can have a load value specified that determines how much it

will slow a particular driver. Signals that are arriving later than the clock can have an attribute that specifies this fact.

2. Drive

The Drive attribute specifies the resistance of the driver, which controls how much current it can source. This attribute also is specified in the units of the technology library. The larger a driver is the faster a particular path will be, but a larger driver takes more area, so the designer needs to trade off speed and area for the best possible implementation.

Synthesis Process

To convert the RTL description to gates, three steps typically occur. First, the RTL description is translated to an unoptimized Boolean description usually consisting of primitive gates such as AND and OR gates, flip-flops, and latches. This is a functionally correct but completely unoptimized description. Next, Boolean optimization algorithms are executed on this Boolean equivalent description to produce an optimized boolean equivalent description. Finally, this optimized Boolean equivalent description is mapped to actual logic gates by making use of a technology library of the target process.

1. Translation

The translation from RTL description to Boolean equivalent description is usually not user controllable. The intermediate form that is generated is usually a format that is optimized for a particular tool and may not even be viewable by the user. All IF, CASE, and LOOP statements, conditional signal assignments, and selected signal assignment statements are converted to their Boolean equivalent in this intermediate form. Flip-flops and latches can either be instantiated or inferred; both cases produce the same flip-flop or latch entry in the intermediate description.

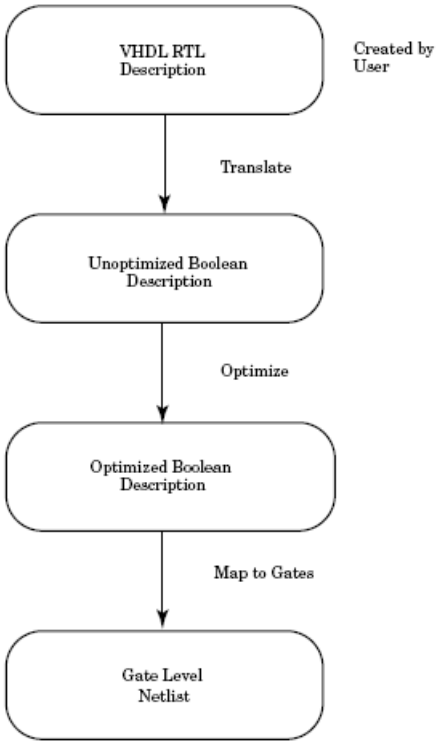


Fig 6: Synthesis process

2. Boolean Optimization

The optimization process takes an unoptimized boolean description and converts it to an optimized boolean description. In many designers' eyes, this is where the real work of synthesis gets done. The optimization process uses a number of algorithms and rules to convert the unoptimized boolean description to an optimized one. One technique is to convert the unoptimized boolean description to a very low-level description (a pla format), optimize that description (using pla optimization techniques), and then try to reduce the logic generated by sharing common terms (introducing intermediate variables).

3. Flattening

The process of converting the unoptimized boolean description to a pla format is known as *flattening*, because it creates a flat signal representation of only two levels: an **AND** level and an **OR** level. The idea is to get the unoptimized boolean description into a format in which optimization algorithms can be used to optimize the logic. A pla structure is a very easy

description in which to perform boolean optimization, because it has a simple structure and the algorithms are well known. An example of a boolean description is shown here:

Original equations
a = b and c;
b = x or (y and z);
c = q or w;
after flattening,

a = (x and q) or (q and y and z) or (w and x) or (w and y and z);

This second description is the boolean equivalent of the first, but it has no intermediate nodes. This design contains only two levels of logic gates: an **AND** plane and an **OR** plane. This should result in a very fast design because there are very few logic levels from the input to the output. In fact, the design is usually very fast. There are, however, a number of problems with this type of design. this type of design can actually be slower than one that has more logic levels. The reason is that this type of design can have a tremendous fanout loading on the input signals because inputs fan out to every term. Second, this type of design can be very large, because there is no sharing between terms.

Flattening gets rid of all of the implied structure of design whether it is good or not. Flattening works best with small pieces of random control logic that the designer wants to minimize. Used in conjunction with structuring, a minimal logic description can be generated. Usually, the designer wants a design that is nearly as fast as the flattened design, but is much smaller in area. To reduce the fanout of the input pins, terms are shared. Some synthesis vendors call this process *structuring* or factoring.

4. Factoring

Factoring is the process of adding intermediate terms to add structure to a description. It is the opposite of the flattening process. Factoring is usually desirable because, as was mentioned in the last section, flattened designs are usually very big and may be slower than a factored design because of the amount of fanouts generated. Following is a design before factoring:

x = a and b or a and d;

y = z or b or d;

After factoring the common term, (**b** or **d**), is factored out to a separate intermediate node. The results are shown here:

x = a and q;

y = z or q;

q = b or d;

The ideal case is one in which the critical path was flattened for speed and the rest of the design was factored for small area and low fanout. After the design has been optimized at the boolean level, it can be mapped to the gate functions in a technology library.

5. Mapping to Gates

The mapping process takes the logically optimized boolean description created by the optimization step and uses the logical and timing information from a technology library to build a netlist. This netlist is targeted to the user's needs for area and speed. There are a number of possible netlists that are functionally the same but vary widely in speed and area Some netlists are very fast but take a lot of library cells to implement, and others take a small number of library cells to implement but are very slow.

Synthesis Result

Cell	Reference	Library	Area	Attributes
U3	NR_NOR4BX4	lst90b_gp_ss125	19.756800	
U4	NR_NOR2X4	lst90b_gp_ss125	6.585600	
U5	NR_OR2X4	lst90b_gp_ss125	10.976000	
U6	NR_OR4X4	lst90b_gp_ss125	19.756800	
U7	NR_OR4X4	lst90b_gp_ss125	19.756800	
U8	NR_NOR2X4	lst90b_gp_ss125	6.585600	
U9	I_INVX4	lst90b_gp_ss125	4.390400	
U10	NR_NOR3X4	lst90b_gp_ss125	17.561600	
U30	NR_AND4BX4	lst90b_gp_ss125	24.147200	
U31	ND_NAND2X4	lst90b_gp_ss125	6.585600	
U32	NR_AND4X4	lst90b_gp_ss125	19.756800	
U33	NR_AND4X4	lst90b_gp_ss125	19.756800	
U35	NR_NOR3X4	lst90b_gp_ss125	17.561600	
U44	NR_OR4X4	lst90b_gp_ss125	19.756800	
U45	NR_OR4X4	lst90b_gp_ss125	19.756800	
U46	NR_OR4X4	lst90b_gp_ss125	19.756800	
U47	NR_OR4X4	lst90b_gp_ss125	19.756800	
U48	NR_OR4X4	lst90b_gp_ss125	19.756800	
U49	NR_OR4X4	lst90b_gp_ss125	19.756800	
U77	M2_MX2X4	lst90b_gp_ss125	13.171200	
U78	M2_MX2X4	lst90b_gp_ss125	13.171200	
U79	M2_MX2X4	lst90b_gp_ss125	13.171200	
U80	M2_MX2X4	lst90b_gp_ss125	13.171200	
U81	M2_MX2X4	lst90b_gp_ss125	13.171200	
U82	M2_MX2X4	lst90b_gp_ss125	13.171200	
U83	M2_MX2X4	lst90b_gp_ss125	13.171200	
U84	M2_MX2X4	lst90b_gp_ss125	13.171200	
U85	M2_MX2X4	lst90b_gp_ss125	13.171200	
U86	M2_MX2X4	lst90b_gp_ss125	13.171200	
U87	M2_MX2X4	lst90b_gp_ss125	13.171200	

2.2 Static Timing Analysis

Defination

It is an effective methodology for verifying the timing characteristics of a design without the use of test vectors or Static Timing Analysis is a method for determining if a circuit meets timing constraints without having to simulate

Advantages

- Fast, exhaustive
- Better analysis checks against timing requirements
- Conventional verification techniques are inadequate for complex designs

Disadvantage

- Less accurate
- Must define timing requirements/exceptions
- Difficulty handling asynchronous designs, false paths
- Proper circuit functionality is not checked

Clock Latency

It is a difference between the reference (source) clock slew to the clock tree endpoint signal slew values here rise latency and fall latency are specified *[refer figure 7*]

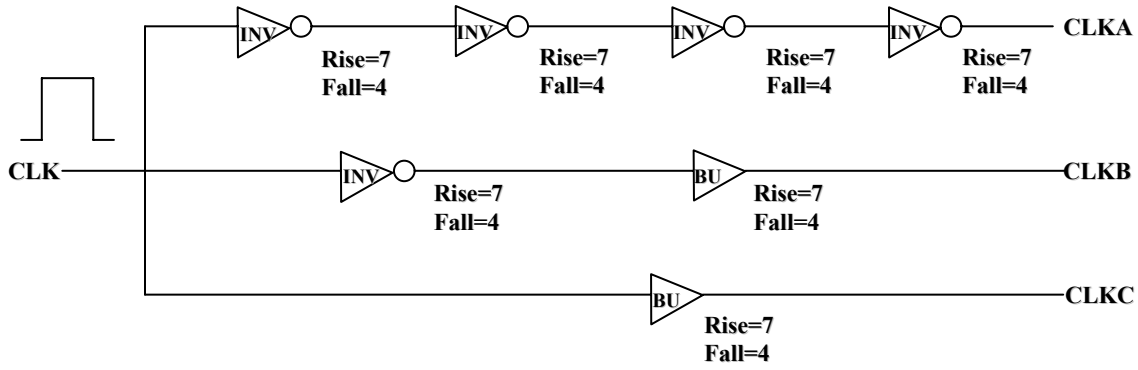


Fig 7: Clock Latency

Input Arrival time

An arrival time defines the time interval during which a data signal can arrive at an input pin in relation to the nearest edge of the clock signal that triggers the data transition

Output required time

It specifies the data required time on output ports

Slack

It is the difference between the required (constraint) time and the arrival time (inputs and delays).Negative slack indicates that constraints have not been met, while positive s lack indicates that constraints have been met. Slack analysis is used to identify timing critical paths in a design by the static timing analysis tool

Critical path

Any logical path in the design that violates the timing constraints or the longest delay path in the circuit called as critical path.

Recovery time

It is like setup time for asynchronous port (set, reset) It is the time available between the asynchronous signal going inactive to the active clock edge

Removal time

It is like hold time for asynchronous port (set, reset)

It is the time between active clock edge and asynchronous signal going inactive

False paths

Paths that physically exist in a design but are not logic/functional paths

These paths never get sensitized under any input conditions [*refer figure 8*]

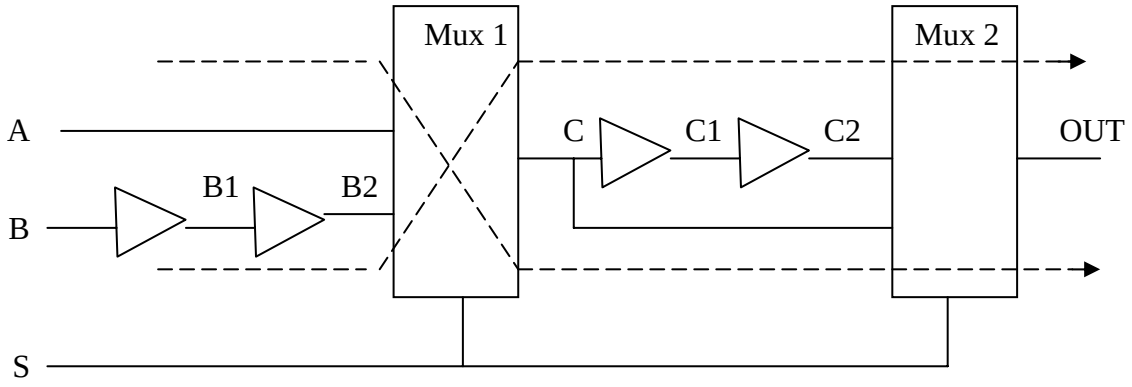


Fig 8: False Path

Multi-cycle paths

Data Paths that require more than one clock period for execution [*refer figure 9*]

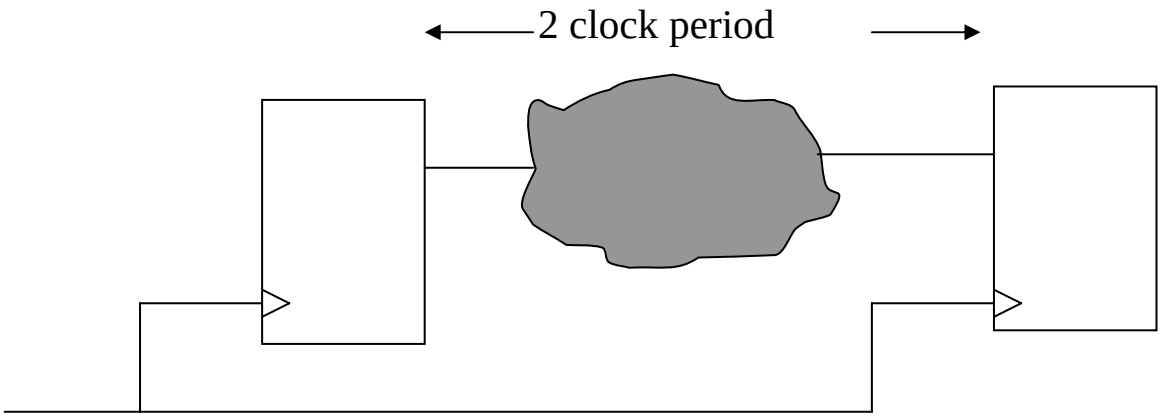


Fig 9: Multi-Cycle Paths

Clock Skew

If a clock edge does not arrive at different flip-flops at exactly the same time, then the clock is said to be skewed between these flip-flops. The difference between the times of arrival at the flip-flops is said to be the amount of clock skew.

Clock skew [*refer figure 10*] is due to different delays on different paths from the clock generator to the various flip-flops.

- Different length wires (wires have delay)
- Gates (buffers) on the paths
- Flip-Flops that clock on different edges (need to invert clock for some flip-flops)
- Gating the clock to control loading of registers (a very bad idea)

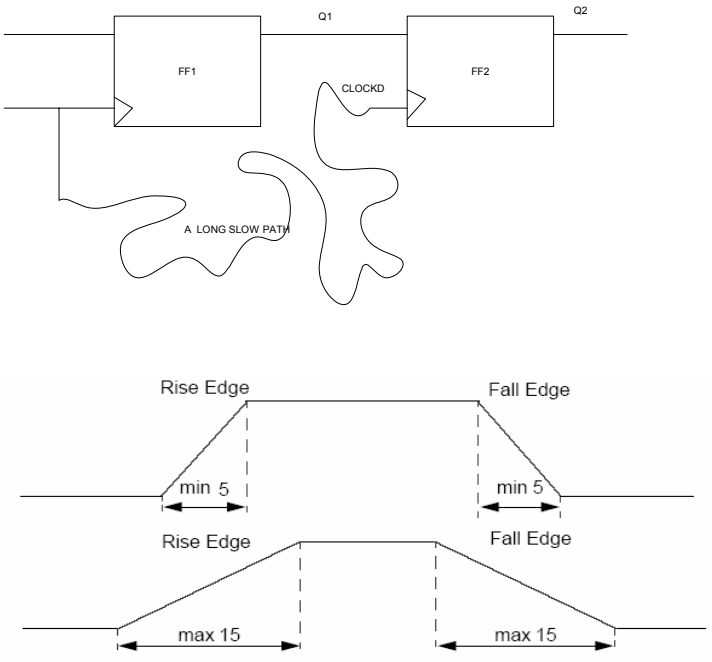


Fig 10: Clock Skew

Propagation delay (t_{clk-q}):

The amount of time needed for a change in the flip-flop clock input *D* to result in a change at the flip-flop output *Q*. When the clock edge arrives, the *D* input value is transferred to output *Q*. After time t_{clk-q} the output is guaranteed not to change value again until another clock edge trigger arrives.

Contamination delay (t_{cd}):

This value indicates the amount of time needed for a change in the flip-flop clock input to result in the initial change at the flip-flop output *Q*. The output of the flip-flop maintains its

initial value until time t_{cd} has passed and is guaranteed not to show any output change in response to an input change until after t_{cd} has passed.

Setup time (t_{su}):

The amount of time before the clock edge that data input *D* must be stable the rising clock edge arrives [*refer figure 11*].

Hold time (t_{hold}):

This indicates the amount of time after the clock edge arrives that data input *D* must be held stable in order for the flip-flop to latch the correct value. Hold time is always measured from the rising clock edge (for positive edge-triggered) to a point after the clock edge [*refer figure 11*].

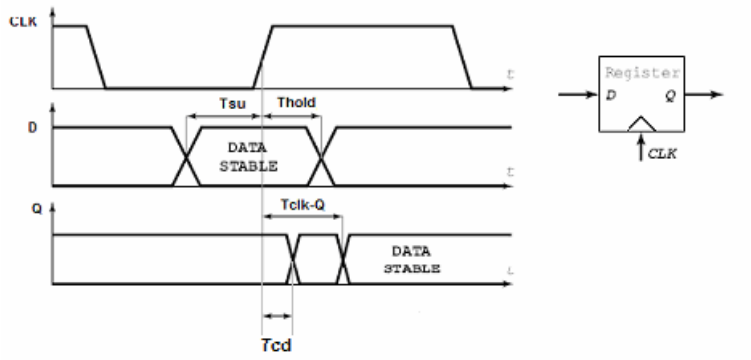


Fig 11: Setup and Hold time

Setup and hold times are *restrictions* that a flip-flop places on combinational or sequential circuitry that drives a flip-flop *D* input. The circuit has to be designed so the *D* input signal arrives at least t_{su} time units before the clock edge and does not change until at least t_{hold} time units after the clock edge. If either of these restrictions is violated for any of the flip-flops in the circuit, the circuit will not operate correctly. These restrictions limit the maximum clock frequency at which the circuit can operate.

Delay Modeling

Gate propagation delay (t_{PHL} and t_{PLH}):

Gate propagation delay [*refer figure 12*] is measured from 50% input to 50% of the output. t_{PHL} is measured from 50% of the rising edge of the input voltage to 50% of the rising edge of the output voltage. Similarly, t_{PLH} is measured from 50% of the falling edge of the input voltage to 50% of the falling edge of the output voltage.

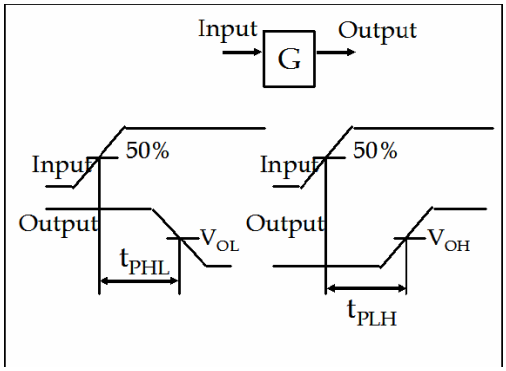


Fig 12: Propagation delay

Interconnect Delay:

This is delay caused by wires. Interconnect introduces three types of parasitic effects – capacitive, resistive, and inductive – all of which influence signal integrity and degrade the performance of the circuit.

After the signal frequencies of our interests (up to several GHz), we can ignore the effects of inductance of interconnect. When gate G1 drives gate G2, we can model the circuit as the one showing below in the right-hand side figure where R_{out} is the output resistance of G1, R_{in} and C_{in} are the interconnect resistance and capacitance, and C_L is the input capacitance of G2.

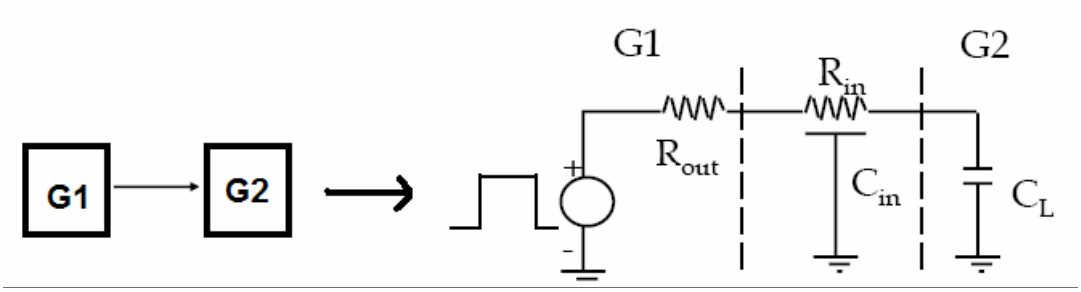
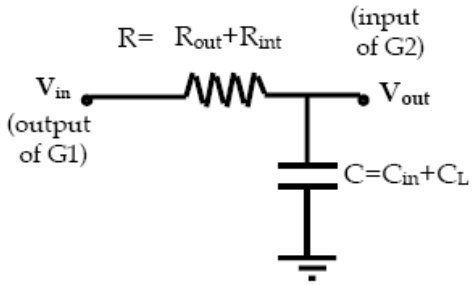


Fig 13: Interconnect Delay

This model can be further simplified as the one shown in the right. Now if V_{in} (i.e. output of G1) switches from 0V to V_{DD}, the waveform at V_{out} (i.e. input of G2) can be expressed as:

$V_{out} = V_{DD} (1 - e^{-t/RC})$



Rise time (Fall time) –

The time it takes for a waveform to rise from 10% to 90% (90% to 10%) of its steady state value – as illustrated in *figure 14*.

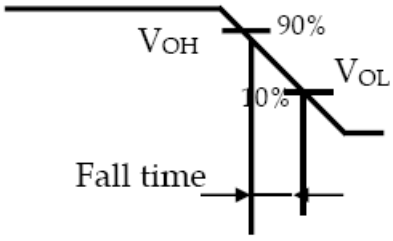


Fig 14: fall time

The time for the waveform $V_{out} = V_{DD}(1- e^{-t/RC})$ to rise from $0.1V_{DD}$ from $0.9V_{DD}$ can be easily calculated:

Rise Time ~= 2.2 * R * C

Note that the rise time is linearly proportional to the product of ($R_{out}+R_{in}$) and ($C_{in}+C_L$).So, if the interconnect is long, R_{in} and C_{in} will be large, and, in turn, the rise time will belong. Similarly, if a gate drives a large number of gates, the rise time will be long too asC_L, which is proportional to the number of driven gates, is large.

Max. Frequency under Clock Skew and Clock Jitter

Clock Skew (δ):

The spatial variation in arrival time of a clock transition is known as clock skew.

The clock skew between two points *j* and *k* is given by $t_j - t_k$, where $t_j - t_k$ are the rising edge of the clock with respect to the reference. Clock skew is constant from cycle to cycle and does not cause clock period variation, but only phase shift. Clock skew might cause the race problem – illustrated in the lecture slides.

Clock Jitter:

Clock jitter refers to the temporal variation of the clock period, that is, the clock period can expand or reduce on a cycle-by-cycle basis.

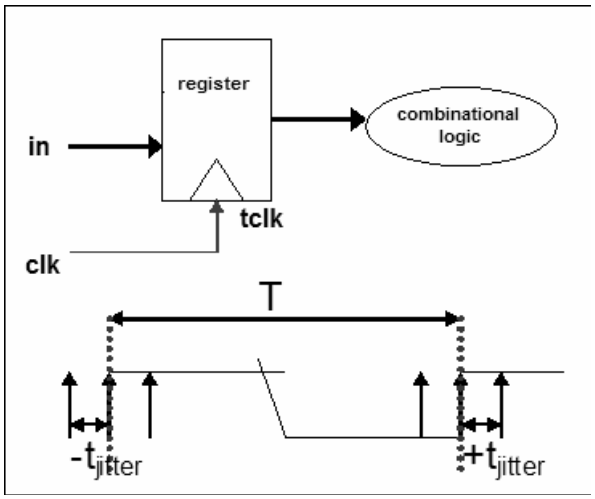
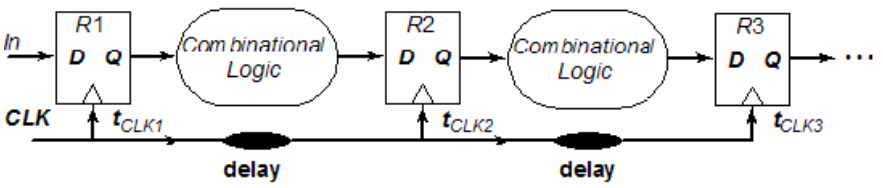


Fig 15: Clock Jitter

Positive Skew: Clock and data flow in the same direction.



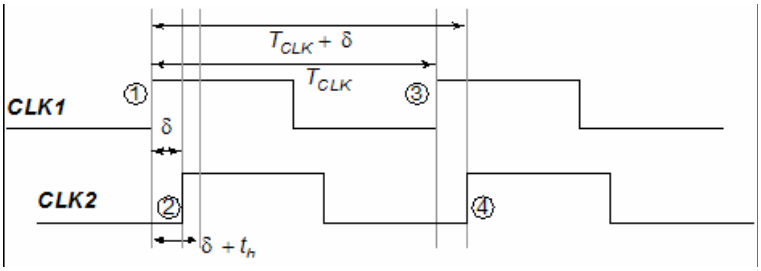


Fig 16: Positive Skew

Minimum cycle time: $T_{clk} + \delta \geq t_{clk\text{-}q} + \text{CriticalPathDelay} + t_{su}$

This means that clock skew has the potential to improve the performance of the circuit (minimum required clock period reduces!). However, increasing clock skew makes the circuit more susceptible to race conditions.

Negative Skew: Clock and data flow in opposite directions

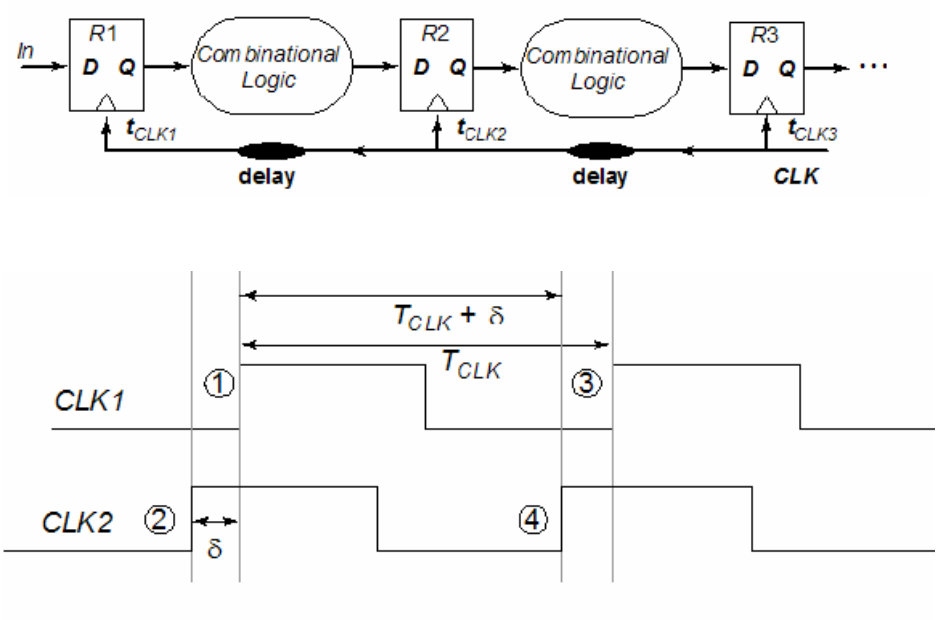


Fig 17: Negative Skew

Note: Receiving edge arrives before the launching edge On one hand, negative skew adversely impacts the performance (increase the clock period). On the other hand, negative skew implies that the system never has the race problem (since receiving edge happens before).

Slack

Setup Check in BC-WC Timing Analysis Mode

At each node is a group of events modeling signal transitions

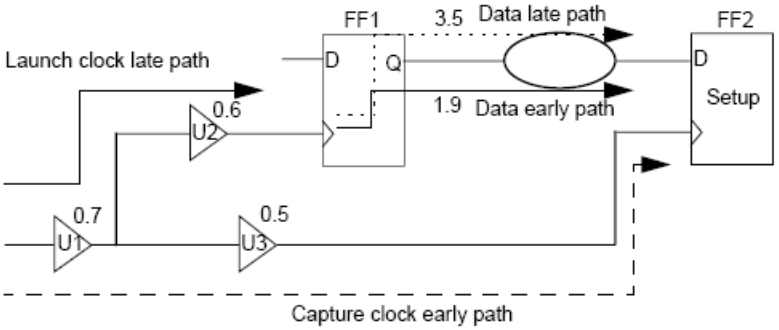


Fig 18: Setup Check

The software uses the Max library to scale all delays at WC conditions.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock Mode = Propagated clock mode

The software computes the slack as follows:

Launch clock late path delay = 0.7 + 0.6 = 1.3

Data late path delay = 3.5

Capture clock early path delay = 0.7 + 0.5 = 1.2

Setup = 0.2

Data arrival time = 1.3 + 3.5 = 4.8

Data required time = 4 + 1.2 – 0.2 = 5

Slack = 5 - 4.8 = 0.2

Hold Check in BC-WC Timing Analysis Mode

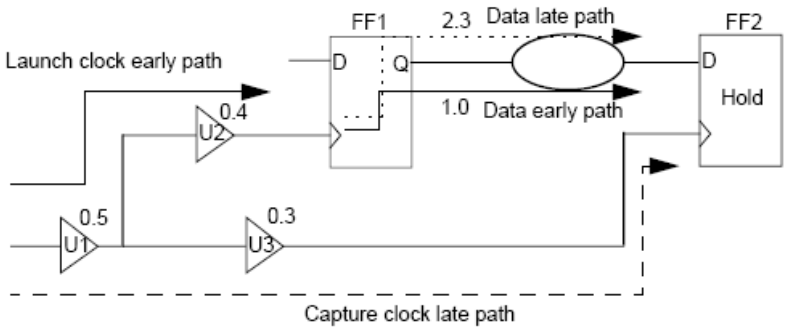


Fig 19: Hold Check

The software uses the Min library to scale all delays at BC conditions.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock Mode = Propagated clock mode

The software computes the slack as follows:

Launch clock early path delay = 0.5 + 0.4 = 0.9

Data early path delay = 1.0

Capture clock late path delay = 0.3 + 0.5 = 0.8

Hold = 0.1

Data arrival time = 0.9 + 1 = 1.9

Data required time = 0.1 + 0.8 = 0.9

Slack = 1.9 - 0.9 = 1

STA Results for FPU Design

```
Summary
Timing Path Group 'clk_i' (max_delay/setup)
  ----------------------------------------
  Levels of Logic:                    46
```

```
  Critical Path Length:                    4.61
  Critical Path Slack:                     0.19
  Total Negative Slack:                    0.00
  No. of Violating Paths:                     0
  --------------------------------------------

  Timing Path Group 'pis' (max_delay/setup)
  --------------------------------------------
  Levels of Logic:                           14
  Critical Path Length:                    2.42
  Critical Path Slack:                     2.23
  Total Negative Slack:                    0.00
  No. of Violating Paths:                     0
  --------------------------------------------
```

Detail

```
  Startpoint: s_opb_i_reg_24_
              (rising edge-triggered flip-flop clocked by clk_i)
  Endpoint: i_serial_div_s_dvsor_i_reg_18_
              (rising edge-triggered flip-flop clocked by clk_i)
  Path Group: clk_i
  Path Type: max

  Point                        Fanout    Cap      Trans       Incr       Path
  ------------------------------------------------------------------------------
-
  clock clk_i (rise edge)                          0.0000     0.0000     0.0000
  clock network delay (ideal)                                 0.0000     0.0000
  s_opb_i_reg_24_/CP (FD1QSVTX2)                   0.0000     0.0000     0.0000 r
  s_opb_i_reg_24_/Q (FD1QSVTX2)                    0.1526     0.2525 &   0.2525 f
  n175 (net)                     2   0.0572
  U9197/A (IVSVTX2)                                0.1526     0.0000 &   0.2525 f
  U9197/Z (IVSVTX2)                                0.1075     0.1180 &   0.3705 r
  n9799 (net)                    6   0.0335
  U9802/C (ND3SVTX8)                               0.1075     0.0000 &   0.3705 r
  U9802/Z (ND3SVTX8)                               0.0467     0.0536 &   0.4241 f
  n11878 (net)                   1   0.0115
  U13217/B (NR2SVTX8)                              0.0467     0.0000 &   0.4241 f
```

```
U13217/Z (NR2SVTX8)                          0.0768     0.0574 &   0.4815 r
i_pre_norm_mul_s_opb_dn (net)
                              7   0.0422
U11725/A (AN2BSVTX4)                         0.0768     0.0000 &   0.4815 r
U11725/Z (AN2BSVTX4)                         0.0315     0.0908 &   0.5723 r
n4251 (net)                   2   0.0121
U11424/C (AO6ASVTX4)                         0.0315     0.0000 &   0.5723 r
U11424/Z (AO6ASVTX4)                         0.0362     0.0411 &   0.6134 f
```

……..

2.3 Wire Load Models

For flows that run timing-based logic optimization before placement, there are three basic types of WLMs that can be used:

Statistical WLMs are based on averages over many similar designs using the same or similar physical libraries.

Structural WLMs use information about neighboring nets, rather than just fanout and module size information.

Custom WLMs are based on the current design after placement and routing, but before the current iteration of preplacement synthesis.

Custom WLMs further divided into three subcatagary

1. Top

   It is the most pessimistic WLM and it considers full top design area taken into account for net delay calculation.
2. enclosed

   It considers the module size in which the net belong in order to find out the net delay.
3. segmented

   It is most pessimistic WLM and it consider individual module thru which net passes for the net delay calculation.

Wire load models (WLMs) are generally perceived to be inaccurate and inadequate for good optimization. The traditional wisdom is that accuracy of WLMs will worsen as die sizes expand and feature sizes shrink, and as wire loads become less predictable and more dominant over pin loads. In many industry white papers and academic works, the weaknesses of WLMs are used to motivate the unification of logic synthesis and physical layout into a single tool. We believe, however, that care must be taken in how we derive our motivations for new flows. In previous studies, evidence against WLMs was generally anecdotal or based on limited data (e.g., from a single design). Today, the maturation of Cadence Design Systems' PKS design tool affords us a unique opportunity to study WLMs in greater depth, and to quantify the timing improvements achieved by the unification of synthesis and layout. Using PKS, we have performed extensive experiments on fifteen real industry test cases. Our results confirm much of the conventional wisdom about WLMs, but also indicate that WLMs probably can still perform a useful function in the design flow.

The typical chip implementation flow starts with an RTL description and moves through logic synthesis and optimization in the front end – to placement, routing, extraction and performance analysis in the back end. At each step of the flow, the designer must ensure that design constraints are met with respect to timing, signal integrity, power and area. If at any step the constraints are violated, the design will need to be sent back one or more steps to be re-optimized. Backward iterations – those that use current layout solutions as estimates for the next pass of logic synthesis – have been necessary because of a fundamental chicken-and-egg problem:

(1) The front-end designer needs knowledge of placement to estimate net parasitics during timing-driven optimization, while

(2) timing-driven placement requires knowledge of the actual netlist area, connectivity, and timing. To reduce time-to-market, minimizing the number of backward iterations is a high-priority objective for CAD tools.

The most popular way to solve the chicken-and-egg dilemma is to estimate parasitics during logic optimization with a wire load model (WLM), a lookup table that maps the fanout of a net to the corresponding estimated capacitance and resistance.1 We formally define a WLM as

a pair of functions *cap*(*f o*) and *res*(*f o*) where the integer parameter *f o* denotes the net fanout. Each WLM lookup table should contain a set of capacitance pairs
and resistance pairs, such as:
fanout_capacitance (1, 0.007250)
fanout_resistance (1, 0.036048)
...
e.g., fanout-1 nets have wire capacitance 0.00725 pF and wire resistance of 0.036048 ohms. In general, not all fanouts are mentioned in a given WLM lookup table. For example, a WLM lookup table might only have capacitance and resistance values for fanouts 1, 2, 3, 4, 5, 10, 20, and 99. Estimates for fanouts in the gaps (e.g., from 6 to 9) are calculated using (linear) interpolation. Estimates for fanouts outside the range of WLM table fanouts (e.g., greater than 99) are calculated using extrapolation based on the values for the two closest fanouts in the table. WLMs are often used in pre-placement optimization to drive speedups of critical paths. Since timing-driven placement plausibly makes nets on critical paths shorter than average, some *optimism* may be incorporated into the WLM. Thus, a WLM may actually consist of more than one lookup table, with each table corresponding to a different optimism level. There are several ways to incorporate the optimism level. If we use the WLMs that come from the (ASIC vendor's) design library, usually there are several tables from which we can select. We can also increase the optimism level of a WLM by multiplying all values in the WLM by some factor less than 1.2 For example; we can use 0.25, 0.5, or 0.75.3

2.4 Formal Verification

Defination

Formal verification can be used to verify a design against a reference design as it progresses through the different levels of abstraction and it verifies functionality without test vectors

Advantages

- Need for reliable hardware validation
- compare to simulation, which explores some of possible behaviors if correct, all behaviors are verified if incorrect, a counter-example (proof) is presented

- It can be used to verify a design against a reference design as it progresses through the different levels of abstraction
- Verifies functionality without test vectors
- Obtaining a complete FSM description of the system.
- BDD operates on sets of points.
- The formula representing the property, i.e., the system is a model of the property.
- Correctness guaranteed mathematically, regardless the input values
- No need to generate expected output sequences
- Formal verification useful to detect and locate errors in designs

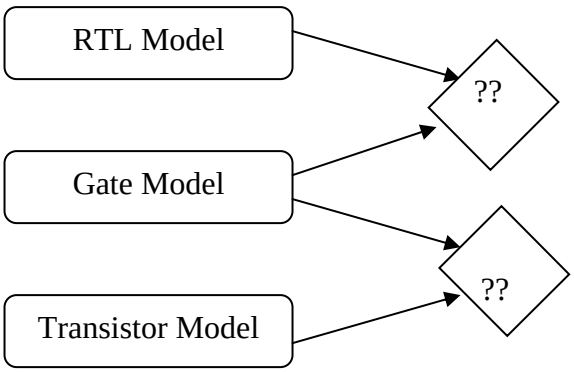


Fig 20: Formal Verification requirement

Simulation vs. Formal Verification

- Simulation: complete model, partial verification
- Simulation still needed to tune specifications; for large complete designs
- The techniques are complementary - formal verification gives additional confidence
- Common difficulty in all verification methods:

Three main categories:

Model Checking

- Compare a design to an existing set of logical properties (that are a direct representation of the specifications of the design).
- Properties have to be specified by the user (far from a “push-button” methodology)
- To check Reachability of states, Deadlock avoidance and Completion of transaction in an interface
- Properties about the system are expressed as formulas in temporal logic of which the state transition system is to be a ``a model".
- Model checking consists of traversing the graph of the transition system and of verifying that it satisfies the formula representing the property, i.e., the system is a model of the property.

Theorem Proving

It requires that the design is represented using a “formal” specification language. Present-day HDLs are not suitable for this purpose.

Equivalence Checking

- It is the most widely used. It performs an exhaustive check on the two designs to ensure they behave identically under all possible conditions.
- Compares two netlists or two RTL codes
- To check: scan chain insertion, clock tree synthesis, manual modification.
- Tools verify the combinational equivalence of two flattened networks
- Any set of functions (the roots), defined over any set of intermediate variables (the leaves), can be checked for equivalence between two networks.
- Roots and leaves are subsets of the nodes of a network, with the restriction that the leaves should form a complete support for the roots
- Two networks are declared combinationally equivalent if they have the same outputs for all combinations of inputs and pseudo-inputs.

Compare Points in Formality (A Synopsys Tool)

A compare point is a design object used as a combinational logic endpoint during verification. A compare point can be an output port, register, latch, black box input pin, or net driven by multiple drivers. Formality uses the following design objects to automatically create

Compare points

- Primary outputs
- Sequential elements
- Black box input pins
- Nets driven by multiple drivers, where at least one driver is a port or black box

Formality verifies a compare point by comparing the logic cone [*refer fig 21*] from an implementation compare point against a logic cone for a matching compare point from the reference design.

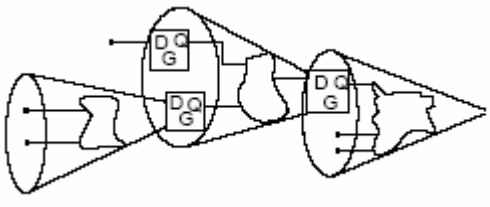


Fig 21: Formality logic cone

When functions defining the cones of logic for a matched pair of compare points (one from the reference design and one from the implementation design) are proved by Formality to be functionally equivalent, the result is that both the reference and the implementation compare points have passing status. If all compare points in the reference design pass verification, the final verification result for the entire design is a successful verification. Prior to design verification, Formality tries to match each primary output, sequential element, black box input pin, and qualified net in the implementation with a comparable design object in the reference design.

Chapter: 3 Physical Synthesis and results

Physical synthesis starting from the netlist and up to the corrects GDS file sign off.It includes the Floorplanning, placement and routing. The individual processes discussed in previous chapter.here practical aspect of those given with the relevant images.

Step 1: Synthesis

First RTL code is given to Synthesizer which will generate optimized netlist as per the given library. Synthesis result gives us design characteristics like no of componenents required, total area , no of pins , no of nets , combination area , sequential area etc.

Step 2: Globle placement

Global placement distributes the cells uniformly across the available core area, minimizing wire length and ensuring constant delay.the cells are placed optimally with the highest performance possible but the placement is not legal as the cells are overlapped and orientation is not proper.[*refer figure 1*]

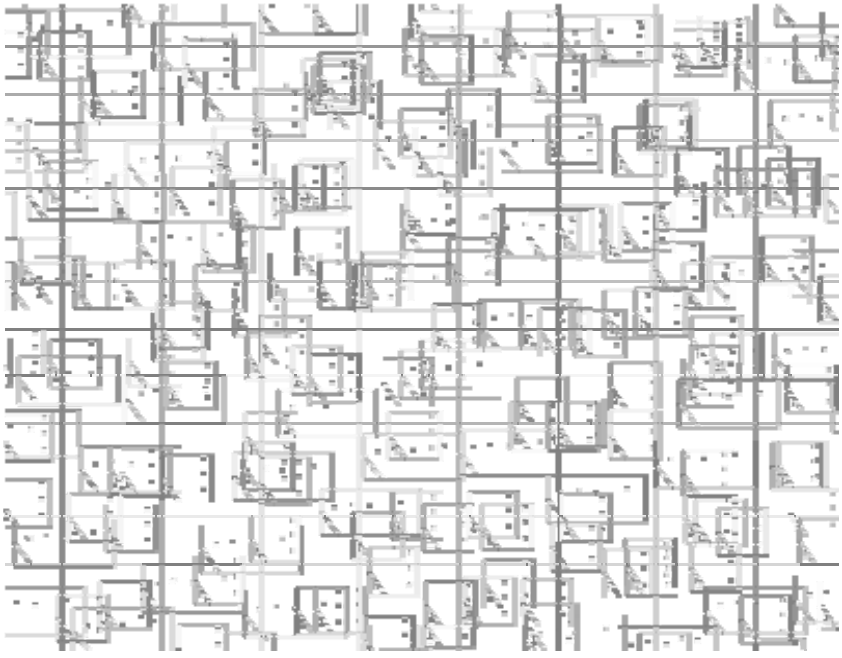


Fig 1: Globle placement

Step 3: Floorplanning

By applying suitable floorplan parameters you can have rectilinear floorplan as shown in below *Fig 2*.actually floorplan also consist Macro placement but in this design no Macros.In this case the task of floorplaning is to differtiate the cells and places them at assign locations as shown in *figure 2*.

Step 4: Detail Placement

Detailed placement minimizes local congestion, reduces routing complexity, and places cells in final positions.After detail placement all cells are placed legally and having propere orientation.

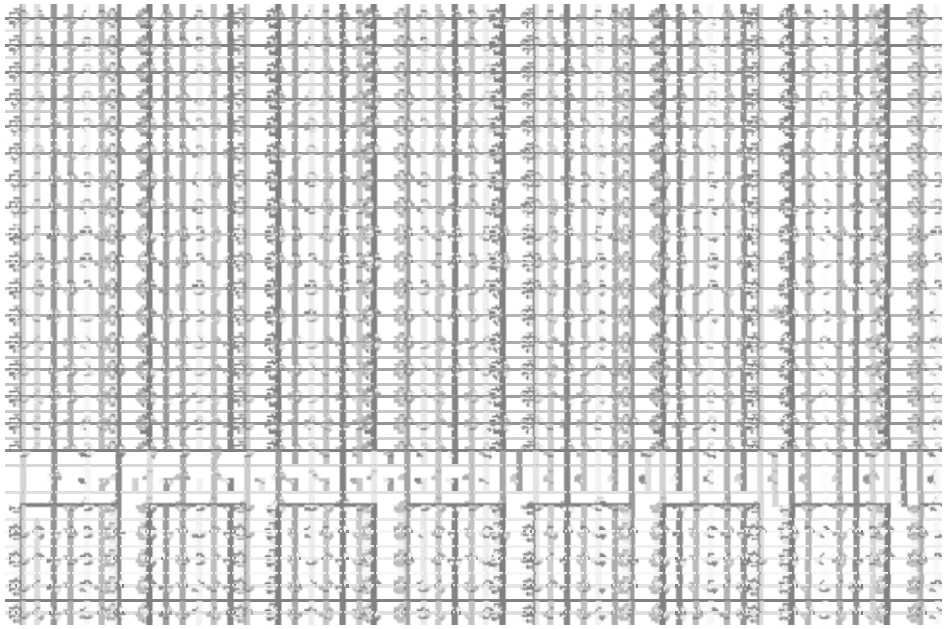


Fig 2: Detailed placement

Step 5: Global Routing

Global routing generates ‘loose’ route for each net.in detailed routing actual geometric layout of net assigned.

It handles crosstalk avoidance, electromigration issues, and constrained wire patterns while ensuring that all design spacing requirements are followed. If design rule violations occur,

the router uses rip-up and reroute methods to repair them. Design rule violations that cannot be repaired are reported. There are five routing stages, in the following order:

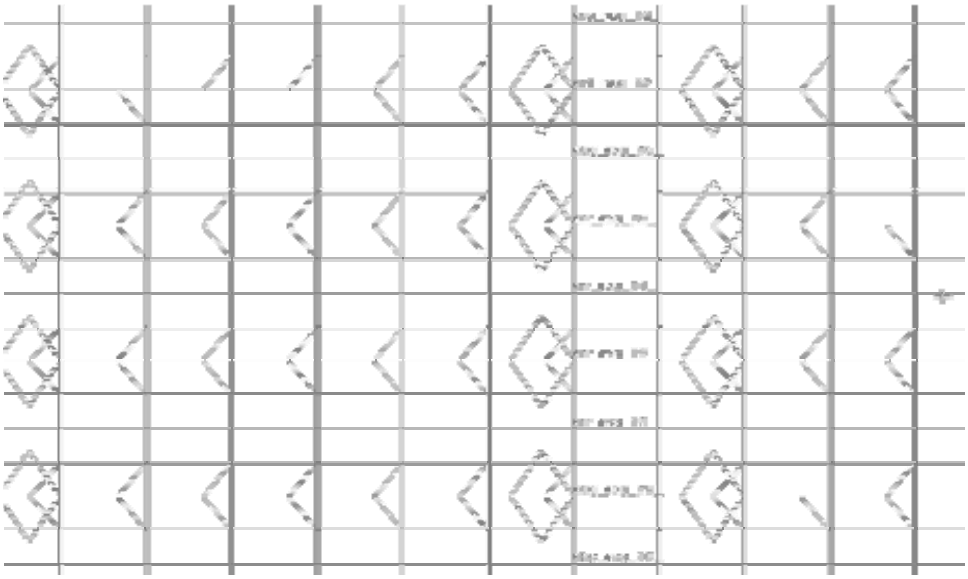


Fig 3: Global Routing

Step 6: Detail Routing

Track routing also works on the bucket level. It orders and spaces the net segments and assigns them to legal track positions.Detailed routing converts the segments to actual wires and vias that connect all pins of all nets

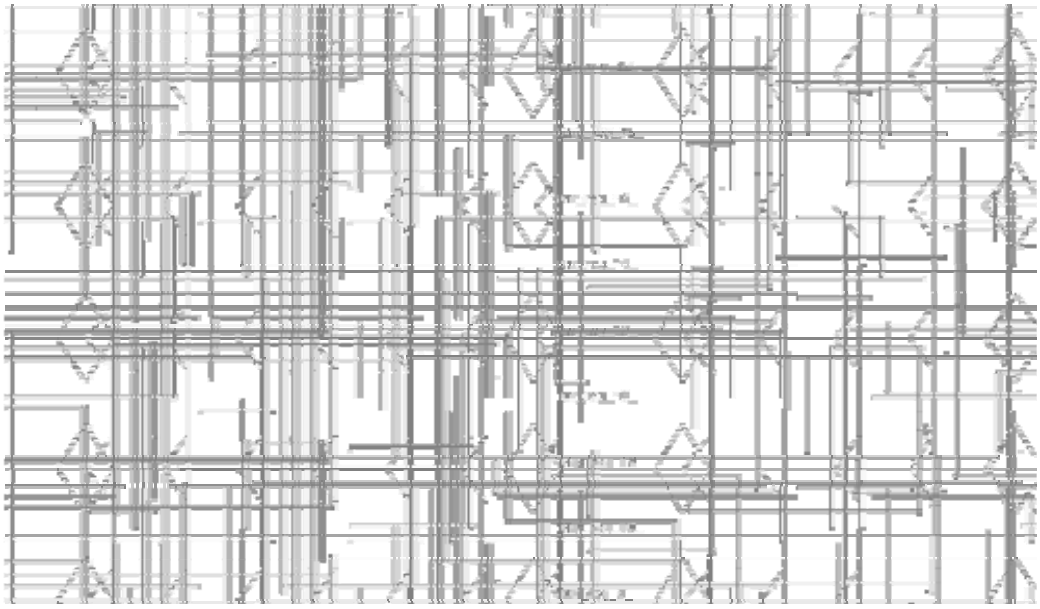


Fig 4 : Detailed Routing (Segmented)

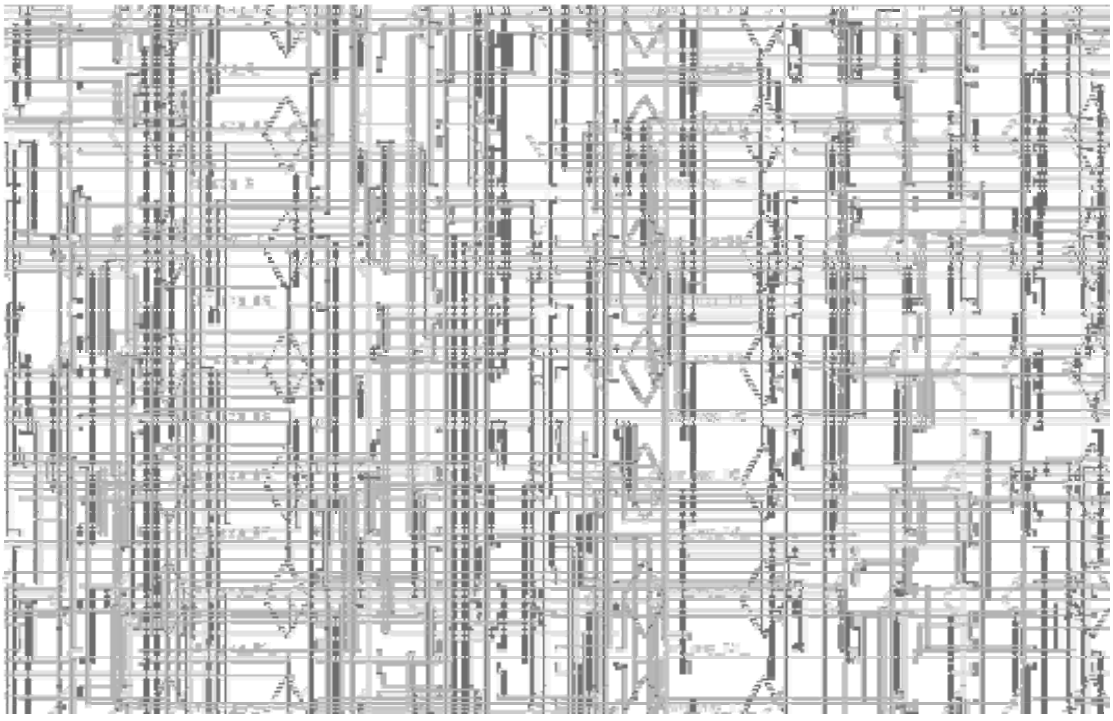


Fig 5 : Detailed Routing (Wire)

Chapter: 4 About Strucrured ASIC Core

1. eASIC

1.1 Architecture:

- eASIC consists of 8 eCores which are made up from 8 eUnit each consists of 256 eCells
- The eASIC Architecture is an array of logic cells, called eCells with SRAM based LUTs (Look Up tables) and flip flops built with bulk silicon and a few metal layers.
- The eCells are connected to a segmented wiring grid built with upper metal layers that can be customized with a single via mask to implement a specific design
- Logic functionality of the eCell is achieved by downloading a bit stream to conFig. the LUTs and flip flops after powering up the device.

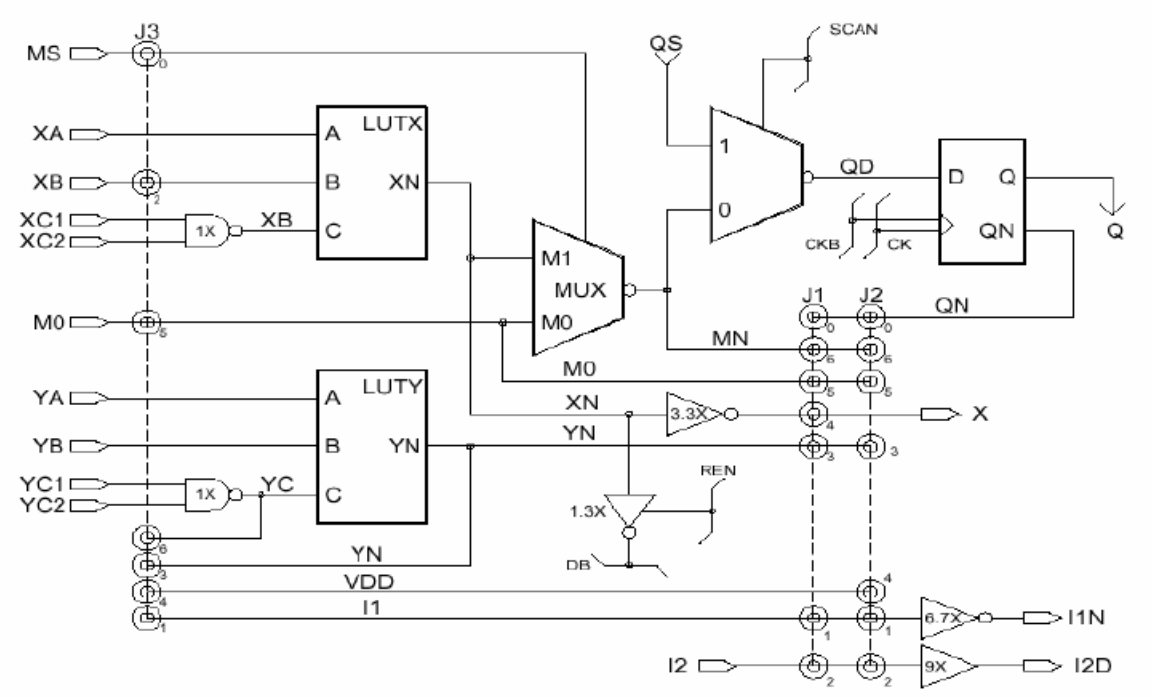


Fig 1: eCell Structure

The wafers for Structured eASIC are pre-processed and stored in a wafer bank at the designated foundry. These wafers can be processed all the way up to the metal 6, since all layers starting with bulk silicon up to metal 6 are identical for all customer designs.

When a customer completes his design, the relevant design files are sent to eASIC. After performing layout verification, eASIC will generate the GDSII file for the custom VIA6 layer based on the customer design. Thereafter, a singleVIA6 mask is generated in the mask shop in order to customize VIA6 and fabricate the required volume. Optionally, an eBeam machine can be used at the foundry to cost effectively customize VIA6 (per the GDSII data) for prototypes or low volume runs (*refer fig 4*).

The main advantage of eBeam customization is in eliminating the mask cost, since the design data is written directly on the wafer. This mask-less method is performed on a per-die basis as opposed to the conventional lithography approach that uses masks and a stepper where the same reticle is stepped across the entire wafer.With eBeam customization, multiple designs from various customers can be written on the same wafer, allowing for very flexible and economical prototyping and low volume production. Standard wafer processing with one custom mask can be then used for high volume production for a given design.

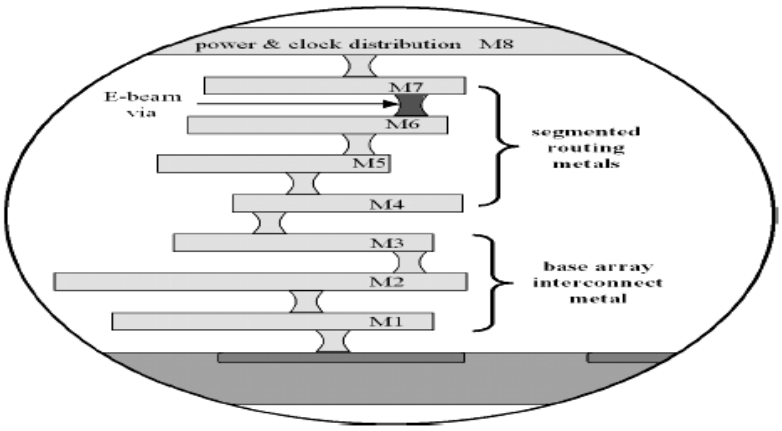


Fig 2: Single Via Mask Customization

1.2 Implementation Flow

The eASIC design flows use standard tools from either Synopsys or Magma to create a netlist-level design. The Magma design flow creates a fully-placed design that is routed using the eASIC eVrouter (*refer fig 3*).

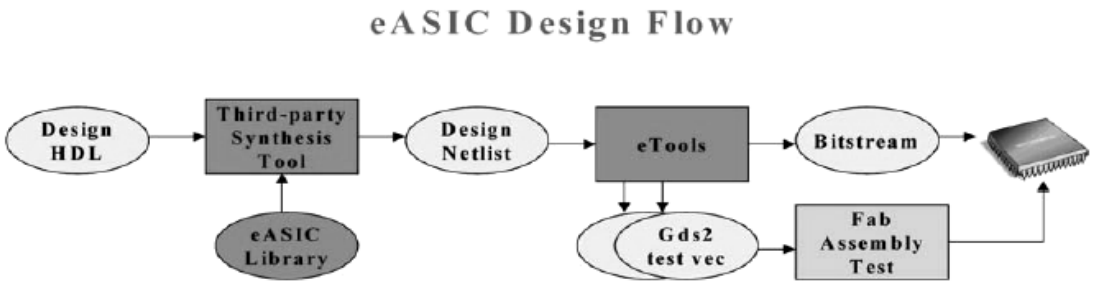


Fig 3: eASIC Design Flow

Logic Synthesis:

Tool: Synopsys DC

Process: IT converts RTL to gaet lavel netlist. Here netlist includes basic gates

Mapping and Placement:

Tool: Magma- SA

Process:

After logic synthesis Magma-SA tool maps the netlist to he LUTs and Muxes followed by tile packing.Tile packing place the LUT and Muxes tile by tile.Finally Magma-SA tool places all the tile withing given core area and try to optimize the performance.

CTS: Clock is pre-routed and balaced.

Routing:

Tool: eTool

Process**:** Simply configure via 6 layers as per the connectivity.

There are several ways in which eASICore technology can be used for implementing ASPP platforms (*refer fig 4*). The design flow and operating model depend on the mode in which eASICore technology is deployed.

Some suggested modes of operation are:

ASPP mode in which eASICore is used by the FPGA vendor to implement industry segment-specific IP combinations to offer a value-added platform to customers. In this mode, the manufacturing flow remains very similar to the traditional standard product flow used for generic FPGA products.

CSPP mode in which the customer determines which IP blocks as well as portions of their design containing re-usable logic need to be implemented using eASICore. This mode can be very attractive for major customers who want to implement a differentiated platform based on their particular needs. In this mode, the manufacturing flow is similar to as ASIC flow.

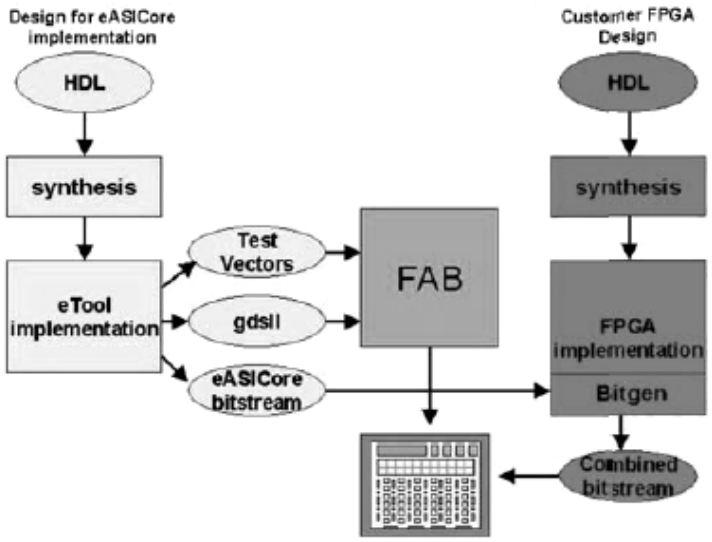


Fig. 4: eASIC Design Flow ASPP and CSPP mode

1.3 Advantages

- The upper metal layers are only used to customize the routing, and not to customize the logic within each eCell.
- Logic customization is achieved by downloading a bit stream. There is no need to have a connection go all the way from upper layers to the diffusion and poly layers. This increases the effective density.
- Since, only one VIA mask needs to be customized, it can be done by direct eBeam lithography, instead of masks. This further lowers NRE costs.
- The interconnects are fully implemented with metals, thereby minimizing routing delays.
- Testing and debugging the design becomes easier as different bit patterns can be downloaded to isolate different section of the chip.

1.4 Limitations

- Due to LUT based architecture, it doesn't correlate with standard cell utilization for all the designs
- Flow limitations with hold violation correction
- Pre-routed clocks, placement is constrainted by clock distribution

2. Vi ASIC

2.1 Architecture:

It consist array of logic cells which are composed of optimized simple gates and SRAM cell bits. *(Refer Fig. 5)*

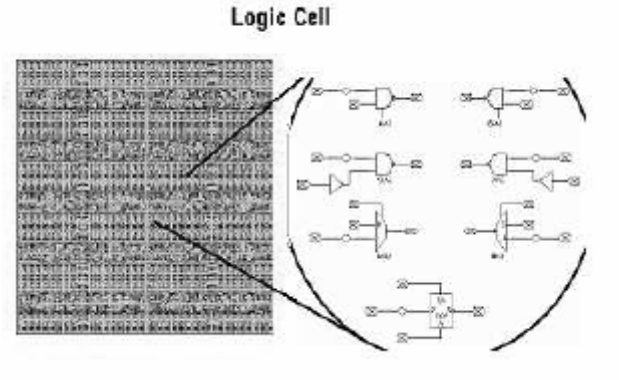


Fig 5: ViASIC Logic Cell

2.2 Implementation Flow

Software for the physical implementation of modular arrays including ViASIC own ViaMask array architecture.It takes in gate level netlist in Verilog or VHDL and produces the physical design . Placement, clock & power insertion, ATPG, routing, timing closure and design optimization perform by ViASIC dedicated tool

Logic Synthesis:

Tool: Synopsys DC

Process: IT converts RTL to gaet lavel netlist. Here netlist includes Muxes and Nand Gates

Placement and Buffering:

Tool: ViaPath

Process: After logic synthesis ViaPAth tool maps the netlist to the Muxes and Nand gates available on tile.Buffering also get done as per the timing requirement using Nand gates available on the tile.

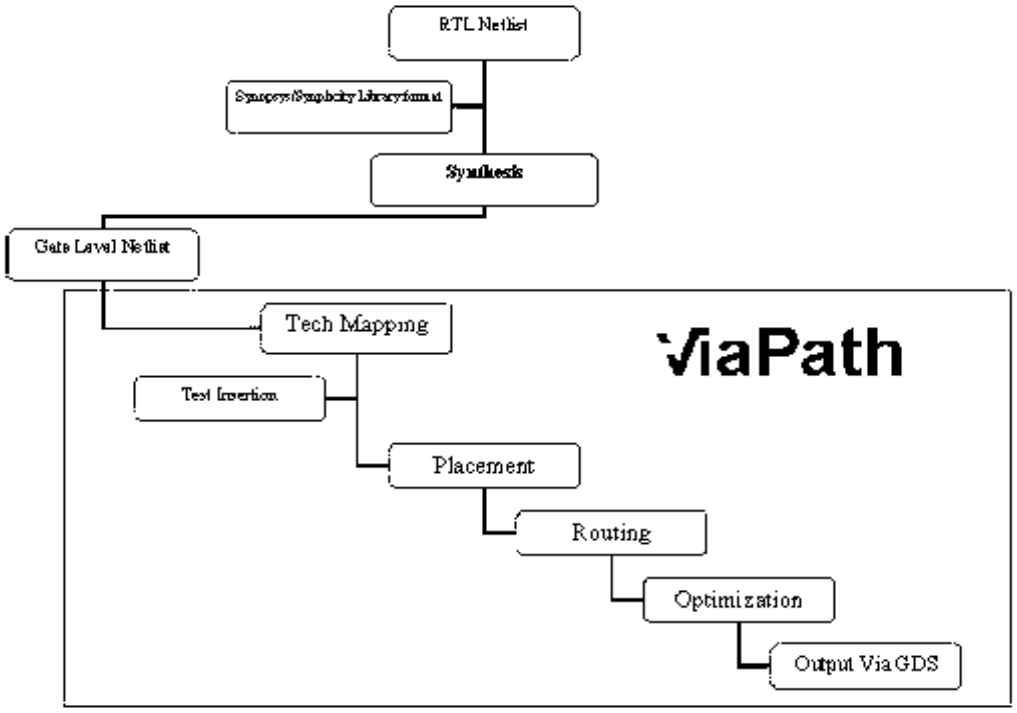


Fig 6: ViaPath

CTS: Clock is pre-routed and balaced.

Routing:

Tool: ViaPath

Process**:** Configure via 3 layer as per the connectivity as shown in below figure.

2.3 Advantages

- economic sense with FPGA, gate array or standard cell solutions
- Faster times to markets
- Low NRE & part cost
- Excellent gate density
- FPGA like design flows

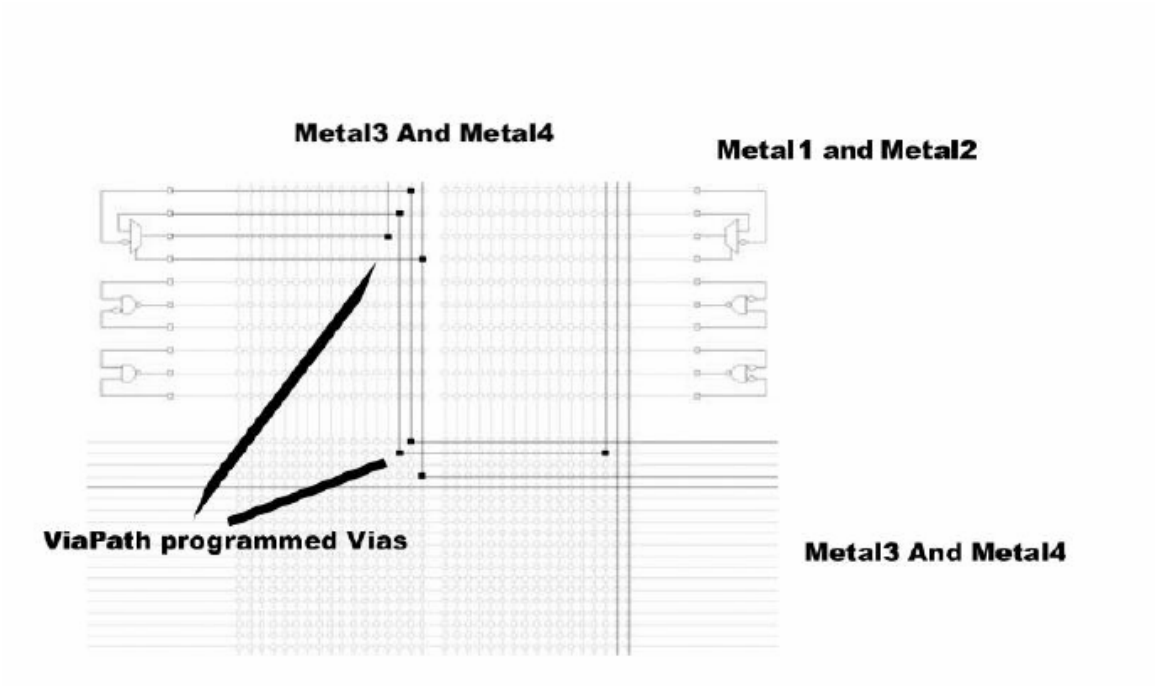


Fig 6A: ViaPath Routing

2.4 Limitations

- There is no buffers in the architecture so it makes higher strength by connecting two cells in paralle.This cell arrangement do not understand by prime time.

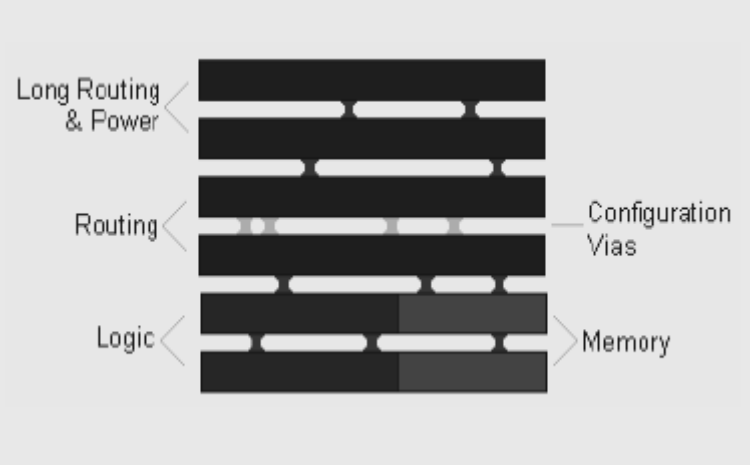


Fig 7: Via Customization

3. LightSpeed

3.1 Architecture:

- Logic Tile built from existing standard cells
  - Pre-characterized & qualified
  - Pre-optimized
- Customization Layers:
  - Range from 2M2V to All-Layer
  - Diffusion & remaining metal/vias are invariant across multiple chip designs
  - Density & performance scale with # of customization layers
- Logic Array is independent of embedded SRAM, analog or other blocks
- Mask Reconfigurable I/Os may also be used

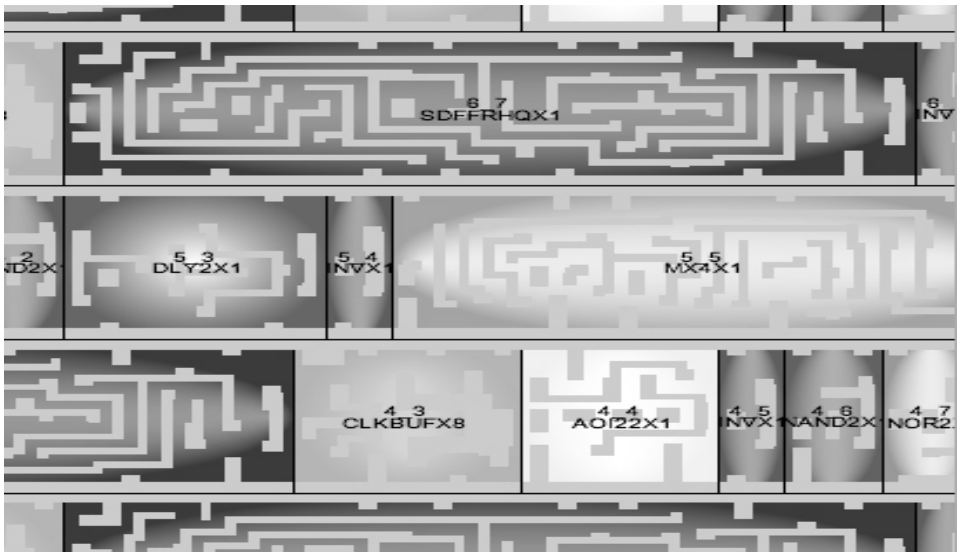


Fig 8: LightSpeed Logic Tile

3.2 Design Flow

Logic Synthesis:

Tool: Synopsys DC

Process: IT converts RTL to gaet lavel netlist. Here netlist includes few ST standard cells and Macro cells like AOI, adders, AB+CD made up from the sane standard cells.

Placement and Buffering:

Tool: DesignBuilder

Process:

Clustering & Mapping Placement

- Timing driven
- Standard SDC support
- Groups related logic & critical paths
- Minimizes global interconnect by maximizing short routes
- Maps logic to maximize utilization

Buffering

- Timing driven
- Low skew clock trees
- Fully gated-clock aware

Buffer high fan-out and long nets

FastFlow Incremental ECO

Common ECO commands

Fully incremental

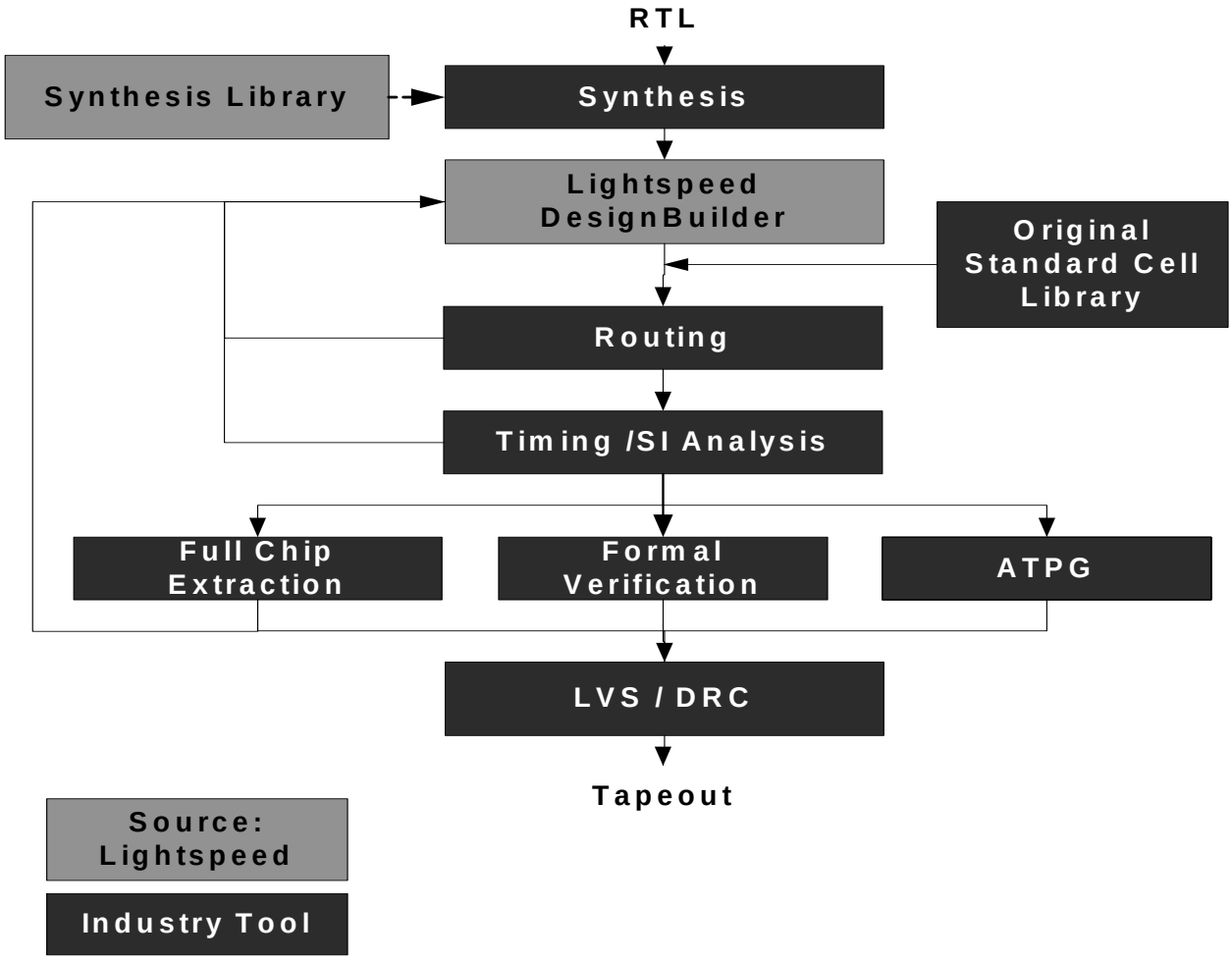


Fig 9: LightSpeed Design flow

CTS:

Tool: Nanorouter- Cadence

Clock tree synthesized at the time of routing as per design constraints given and design placement already done by the DesignBuilder Tool

Routing:

Tool: Nanorouter- Cadence

Process**:** Here as per the core various routing customizationoption available ranging from 2M2V, 3M3V to 4M4V.

3.3 Advantages

- The Industry's leading Reconfigurable Logic Array
- Best density and performance in the industry
- Portable to any standard cell library & foundry
- Significantly reduces deep submicron effects
- Lowers cost & improves time-to-market

3.4 Limitation

- 8-layer change
- Tool maturity is an issue

3.5 Comparision

Feature	Lightspeed	ViASIC	eASIC
Architecture	Few mask re-configurable: 2M2V to 6M6V	Only one via mask reconfig.	Need mask reconfig & external PROM
Basic Element	Logic Block consisting of combination of various Standard Cells Buffer,AOI,NAND,NOR	Logic Cells Consisting of 4 NAND gates, 2 MUX and 1 FLOP and also dedicated memories	eCells Consisting of 2 3 – input LUTs, Flip – Flop and some additional logic
Max. Capacity	250KGates	150KGates	150KGates
Performance (Max Freq.)	0.7	0.3	0.2
RAM	No Dedicated RAM Block	Dedicated RAM Blocks	Dedicated RAM Block
Granularity	Fine Grain	Fine Grain	Coarse Grain as having LUTs
Clock Tree	Synthesized on Metal	Via Synthesizable	Pre Synthesized
EDA Flow	Std ASIC synthesis Vendor placement Vendor CTS Std ASIC routing	Std ASIC synthesis Vendor placement Vendor CTS Vendor routing	Std ASIC synthesis Magma-SA placement Vendor routing
Density	~75% Standard Cell	~33% Standard Cell	~33% Standard Cell
Logic Flexibility customization	No	No	Up to some extend as configuration by bit stream
Clock Tree	Synthesized on Metal	Via Synthesizable	Pre Synthesized
Multi Clock	Any no. of Clock	Any no. of Clock	Limited no. of clock

Table 1: Structured ASIC architecture comparision

4. Other Cores

Worldwide merchant market dollar shipments of structured ASIC products are forecast to soar from the $209.8 million reached last year to $2.53 billion by 2009, eports In-Stat (http://www.in-stat.com). While the use of this technology will be dominated by ASIC designs throughout the forecast period, it will find growing applications in the world of Application-Specific Standard Products (ASSPs), whose revenues will reach nearly 30% of the market's total by 2009.

We enter a domain that will probably become the exclusive property of structured and platform ASIC. With 90nm offerings moving this range up to 10-million ASIC gates with copious amounts of on-chip memory and robust IP libraries, there is no technology that threatens serious competition with these devices. FPGAs don't match their density, performance, power consumption, or unit cost. Full-blown ASICs cost ten to a hundred times more to develop because of mask, NRE, tool, and design team costs.

eASIC's structured Asic product enables the ARM926EJ processor to be made available in a configurable fabric for mass usage

4.1 Altera: Hardcopy

Altera's HardCopy® structured ASICs gives a design process with the flexibility of FPGAs and the low cost of ASICs for your high-volume applications. Develop your design and test it in-system with an Altera® FPGA. When your design is fully tested to meet your requirements, the Altera HardCopy Design Center migrates the design to a functionally equivalent, pin-compatible HardCopy device. You get:

- A complete production solution, from prototype to high volume
    - Devices, tools, and intellectual property (IP)
    - Single vendor from prototype to production
- Minimal design risk and guaranteed functionality
    - FPGA-proven functionality preserved—netlist is unchanged
    - Proven process technology (same as the FPGA)
    - Same package and pin-outs as the FPGA
- Fast device turnaround
- Minimal effort during the migration process

Altera’s HardCopy structured ASICs are unique because they embed hard functions from the Stratix FPGA series (and equivalent I/O) into the base layers, delivering unprecedented design flexibility. HardCopy devices allow you to:

- Use Altera's Quartus® II software along with the EDA tools of your choice to generate your design
- Test your design in-system and at-speed with a Stratix or Stratix II FPGA
- Migrate seamlessly from your FPGA design to a low cost, pin-compatible HardCopy device with no risk and very little effort
- Switch back to FPGAs if you need to change the design to accommodate a new standard, customize for a specific market or application, or if production volumes decrease

4.2 Faraday Technology: MPCA

Metal Programmable Cell Array (MPCA) Library™

Faraday's metal programmable standard cell library is optimized for deep sub-micron designs with a focus on optimizing routability, speed and minimizing power consumption. As shown below, only the top three metal layers are needed to program the library cells as well as place and route the design.

- UMC 0.35µm, 0.25µm, 0.18µm, 0.15µm, 0.13µm and 90nm CMOS technologies
- Supports unlimited clock domains and gated clocks
- Area and performance close to that of the standard cell implementation
- Very low system power
- Built-in scan-based storage elements
- Supports tri-state designs
- Uses conventional standard cell design flow, works smoothly with most EDA tools
- Provides complete EDA views for most popular design flows

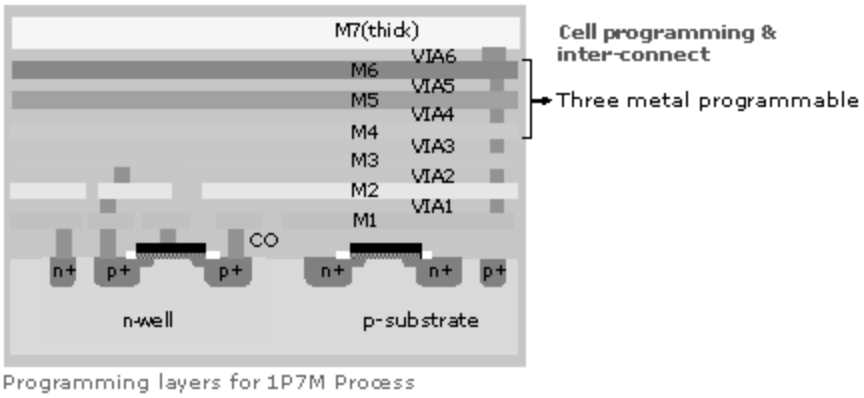


Fig 10 cross section view MPCA

Metal Programmable I/O (MPIO) Library

Imagine one I/O capable of meeting various applications! By changing just one top metal mask, Faraday's MPIO can support various types of I/O. During chip implementation, Faraday's MPIO can be blended with any other Faraday I/O for easy integration.

- UMC 0.25µm, 0.18µm, 0.15µm, 0.13µm and 90nm CMOS technologies
- Require only one metal layer to program MPIO to support
- LVTTL/CMOS
- SSTL-2

  -Class I

  -Class II : Support DDR266, 333 and 400
- PCI-66
- PCI-X
- Input buffer with programmable pull-up, pull-down, keeper and Schmitt-trigger
- Different output driving strengths Output slew rate control
- Built-in level-shifter
- ESD robustness and latch-up immunity proven by silicon

4.3 ChipX: CX5000 Structured

The CX5000 Structured ASIC family is based on 0.18µ technology using 3 layers of programmable metal, and supports performance levels up to 200 MHz. The 0.18µ CX5000 is a Structured ASIC that utilizes the combination of advanced metal programmable gate array technology and an optimized EDA system to implement high performance ASIC designs while reducing application tooling costs and design turnaround time. ASIC designers using the CX5000 are able to meet or exceed their design schedules and budgets without compromising technical objectives.

Key Features

- 30K to 1.1M usable ASIC gates
- Up to 2.6M bits of fast block memory
- 2ns access time single-port/dual-port SRAM and ROM
- Configurable I/O: PCI, PCI-X, SSTL, HSTL, USB FS, RSDS, LVTTL, LVCMOS, LVPECL and LVDS up to 622Mbps as well as Power & Ground
- Up to 1,152 total pads
- 200MHz general core logic operation, 500MHz in constrained clock domains
- 1.5V or 1.8V or mixed supply voltage operation
- 4 Low-jitter APLLs macro with internal loop filter
- 3-4 week lead time for tested prototypes

ChipX: CX2000

The CX2000 Structured ASIC family is based on 0.6µ technology using 2 layers of programmable metal, and supports performance up to 50 MHz. This family is favored for legacy applications with industrial and military customers for its proven reliability, 5V operation and radiation resistance. This mature family has a long established record of reliability and a long future for assured supplies continuity.

Key Features

- 23k to 100k usable ASIC gates

- Up to 96k bits total SRAM configurable as single-port RAM, dual-port RAM or ROM
- Configurable I/O: PCI(3V/5V)
- 5V drive or 3.3V with 5V tolerance
- 50 MHz general core logic operation, 100 MHz in constrained clock domains operation
- Analog PLL
- 5-6 weeks lead time for tested prototypes

4.4 NEC: ISSP90

The ISSP90 employs a leading-edge 90 nm CMOS process and features an operating frequency of 500 MHz, double that of the ISSP1. In addition, the number of user gates is four times higher (6.5 million gates), and a two-fold SRAM capacity of 5.7 Mb is provided.

Through the use of a 90 nm process, ISSP90 enables high-performance devices that could not be realized heretofore. A wide range of applications ranging from next-generation high-speed and high-capacity (broadband) communications or network equipment and servers to high-performance measuring instruments and industrial/consumer equipment is supported. ISSP90 is already widely acclaimed and used as a solution that realizes high-performance devices at inimal cost.

Using the 90 nm process technology, the ISSP90 achieves high density. Besides, two customization layers are embedded. The separation of the common layers that contain power-supply, clock-circuit, and testing lines from the user logic layers (customization layers) achieves a structure that is highly effective for signal integrity (SI). Accordingly, high quality is guaranteed.

4.5 AMIS: Xpress Array

Targeted at medium-density, high-speed, 1.5V ASIC applications and high-density FPGA-to-ASIC conversions, the XPressArray®-II 0.15µm structured ASIC is an innovative next-generation technology platform that reduces time-to-market for system-on-chip (SoC) applications while delivering significant NRE and unit cost savings.

XPressArray-II offers a true drop-in replacement for Xilinx Virtex-II, Virtex-II Pro, Altera APEX-II, and Stratix FPGAs, making it the industry's lowest cost FPGA conversion solution. The result is a simplified route to cost reductions for OEMs looking to combine the flexibility of FPGA prototyping with a path to low cost ASICs for final production.

Operating with system clock speeds up to 210MHz for 18x18 soft multipliers and local clocks up to 500MHz, XPressArray-II 0.15µm devices deliver high performance, low power ASIC solutions with densities to 4.8M ASIC gates.

Configurable memory ranges from 258kbits to 4.8Mbits, which increases up to 6.1Mbits of memory with the addition of distributed configurable memory, assuming 50 percent of the logic sites are used for memory.

Flexible I/O technology includes support for a comprehensive array of common standards and compatibility with 1.5V, 1.8V, 2.5V, and 3.3V I/O schemes. I/Os support digital controlled impedance (DCI) on-chip termination. Dual data rate (DDR) support for high-speed memory interface is built-in. High fault coverage is provided through integrated scan-test, memory BIST and JTAG support.

For FPGA conversions, rapid access to XPressArray technology can be achieved via AMI Semiconductor's NETRANS® FPGA-to-ASIC design flow. Alternatively, the availability of XPressArray synthesis libraries for leading commercial synthesizers allows conversion of FPGA designs to ASICs by simply re-targeting fro an FPGA library to an XPressArray library.

Company	Product	Tech.	Layers	Embedded
Altera	*HardCopy*	90nm	2M	No
Farada Tech.	*MPCA*	90nm	3M	Yes
AMIS	*Xpressarray*	180nm	3M	No
ChipX	*CX5000*	180nm	3M	Yes
NEC	*ISPP*	90nm	2M	Yes
LightSpeed	*LS Reconf. IP*	90nm	3M	Yes
ViASIC	*Std. Metal*	130nm	1Via	Yes
eASIC	*Nextreme*	90nm	1Via	Yes

Table 2: Structured ASIC Products

Chapter: 5 Flow Automization and Banchmarming

1. Flow Automation

Flow Automization makes the flow fast and very readable to the user so that even novice also can use the tool for initial results. Various optimization options available to the user like scan chain insertion, testability, tools and run time. So that user can have flexibility to realize design with available options.So for initial results knowledge of tool handling is not required

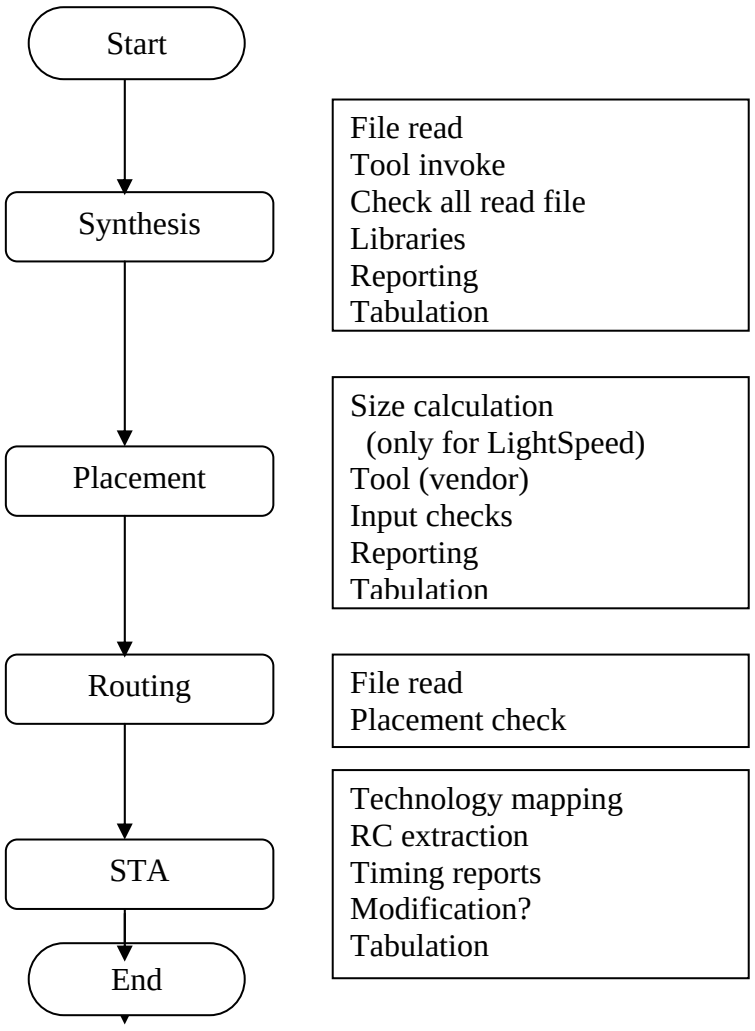


Fig 1: Automization

Requrement and Expected Results

- Automatic Report generation irrespective of design
- User should be reported if any file missing because It is possible that if any file missed than initially tool will run but lateron it gives error and all in vain.
- Minimum interfearance with thew user.
- Make maximum possible flow options available to the user.
- User should be notified for any type of error rather than Tool crashing.
- If required report should be in form of table or excel sheet.
- Temporary files and log files should be removed after execution completed.
- Database format should be uniform for all the designs.
- User able to change the generated scripts and run it.
- Options like Script generation and execution, generation only, generation and modification, generation, modification and execution should be available.
- It should check the tool updates.
- Other groups can use it without any trouble.

Benefit to the Group

- Easy design implementation.
- No need to check background processes.
- Short run time.
- Early result prediction and exhaustive flow setup.
- Less chance of any accidental errors like missing file, missing parameters wrong commands.
- Benchmarking takes 90% less time compare to the earlier flow.

Limitations

- No dirtect tool interfearance with the tools unless explicitly specify.
- If any error occurs, it is hard to debug.
- Sometimes need to modify the script for better results.

2. Benchmarking

It is a procedure to find out the architecture suitability and flow suitability fot the different designs and available Structured ASIC cores like ViASIC, eASIC and LIghtSpeed.

This procedure deals with the implementation on eASIC, ViASIC, LightSpeed and its architecture comparision and Synthesis tool effect on the respective implementation. This report compares their architecture goodness using experimental techniques. Comparison is made with respect to each other. A design suite is defined for the experiments, upon which different flows are executed. Along with the results, various assumptions and observations are highlighted in the report.

2.1 Banchmark Siute

The benchmark suite consists of designs collected in-house as well as from different open source communities. The benchmark suite is intended to cover all application domains and design character. Also, we have ensured that the design suite covers designs with wide variety of gate counts. We have designs rich in arithmetic operators as well as ones with heavy control logic in them. All the designs have been customized to successfully pass the flow. This customization is required to overcome Synthesis issues or to workaround unsupported features. Table below describes the design suite, along with approximate ASIC gate count for the design. As an on-going activity, the benchmark suite is updated as and when suitable designs are found.

Few common modifications are listed below: -

1. All asynchronous reset signals should follow strictly active-low behaviour
2. All asynchronous set signals should be converted to asynchronous reset signals
3. Remove all logic associated with asynchronous reset signal
4. All tri-state buffers should be converted to multiplexors
5. Remove all unsupported constructs in Verilog / VHDL

Table 1 shows the different designs taken for the benchmarking along with the design size in terms of the equivavalent KGates.

Designs	Approx. GateCount (Kgates) =CellArea/Area of 2input NAND gate
AC97_CNTRL	*18.36*
AES_CORE_INV	*34.81*
C3V2	*149.84*
DCT	*129.07*
FPU	*19*
IRDA	*22.39*
JPEG	*120.91*
NoC_70K	*76.05*
OC_8051	*28.31*
OCIDEC2_VER	*2.95*
PANCHAM	*20.11*
PIC	*1.97*
PROG_V	*0.49*
RGB2YCRCB	*4.88*
SPI	*3.49*
SXP	*27.97*
TV_80	*6.82*
USB2	*18.77*
VGA	*7.47*

Table 1: Bencmark Suite

Libraries

The libraries used in this flow are as following

1. eASIC Libraries

   It consists of various possible combination of eCell configuration.

2. LightSpeed Libraries

   It consists of basic gates of different driving capability and various boolean functions realize using these Gates.

   NOR, Xor, XNOR, NAND, AND, OR, AOI, OAI, MUX, ADDER, 3-input function etc

3. ViASIC Libraries

   It also consist of basic gates made up from NAND gate and Mux.

   NAND, AND, OR, NOR, XOR, ANDXOR (A&B^C), XORAND, MUX, FULL ADDER

2.2 Benchmark Flow

Here design suite is taken for the Benchmarking which has already discussed. The Benchmarking flow has shown in *figure 2*.First same RTL code and constraints given to the all Structutred ASIC Flow which has automized.

After complition of the Frontend and Backend flow reports are generated as per the core. Here if design satisfied the constraints for any of the core than the constraints are squeezed until it does not meet the same. Here area is not the constraints so the aim of bencmarking is to find out maximum frequency it can support for the given design

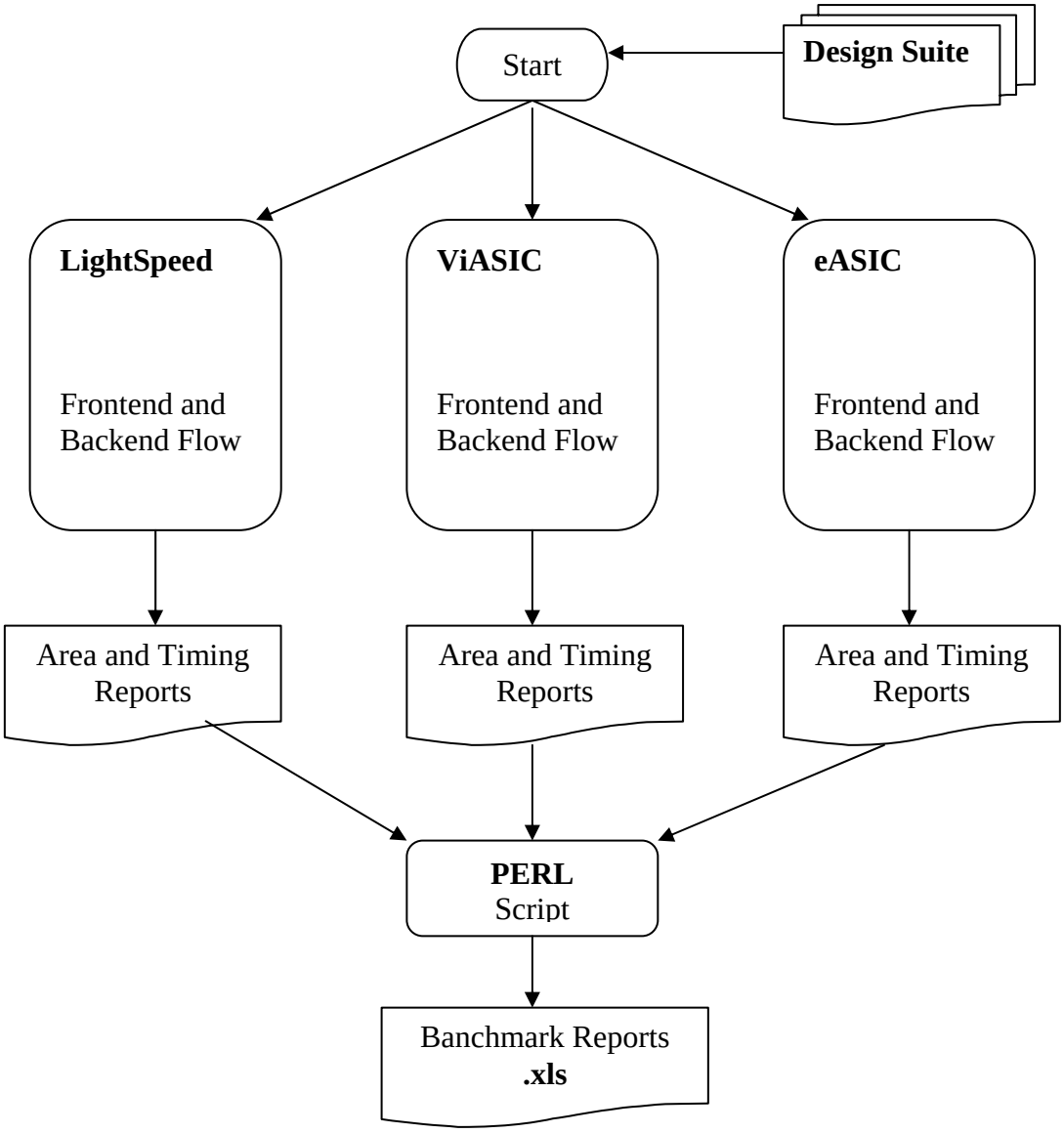


Fig 2: Benchmarking flow

3. Architecture Comparision

Flow (*refer Fig 2*) intends to constraint the design for area. The flow compares usage of architecture resources in eASIC, ViASIC and LightSpeed architectures.

Firstly, the designs are run through Design Compiler, once using the respective Structured ASIC core libraries, setting very relaxed timing constraint and stringent cell area constraints. Total number of real estate resources occupied by the core is calculated from the reports. The verilog netlist obtained is then targeted for the respective Structured ASIC cores (LightSpeed,

eASIC and ViASIC) available. The final report highlights number of basic building blocks in Structured ASIC cores required for the implementation.

4. Reports and Conclusions

Table 2 shows the relation between area and frequency for eASIC,ViASIC and LightSpeed Cores.The table shows the maximum frequency achieve for the given designs.The results are normalized with respect to the same of LightSpeed as LighSpeed can achieve maximum frequency among all of the cores.

Based on the results we can draw following conclusions and trade-offs.

Figure 3 the timing results for the all designs. Here the minimum possible clock period is nomaliza with respect to the same of LightSpeed as it has smallest clock period.from the figure we can conclude that the minimum clock period in case of eASIC is 2.5 to 4.2 times that of the LightSpeed and in case of ViSIC it is varying between 1.5 to 3 times that of the LightSpeed.On an average the clock time period ratio between LightSpeed, ViASIC and eASIC is 1:2.26: 3.3.

Design	LightSpeed		eASIC				ViASIC	
	Cell Area	Time Period (ns)	3-LUT	FF	Time Period (ns)	normalize	Time Period (ns)	normalize
AC97_CNTRL	117351	1.09	3499	2259	2.73	2.50	1.56	1.43
AES_CORE_INV	4629	0.82	225	41	2.15	2.62	1.54	1.88
C3V2	514919	1.41	6898	7976	3.91	2.77	2.98	2.11
DCT	23053	0.93	1090	304	2.66	2.86	1.89	2.03
FPU	6961	0.7	175	117	2.11	3.01	1.74	2.49
IRDA	155919	1.17	9226	739	3.59	3.07	2.66	2.27
JPEG	34571	1.09	2074	362	3.39	3.11	2.25	2.06
NoC_70K	87887	1.33	6868	815	4.14	3.11	3.37	2.53
OC_8051	284936	2.03	19963	2096	6.48	3.19	4.98	2.45
OCIDEC2_VER	52816	1.8	2815	246	5.78	3.21	3.87	2.15
PANCHAM	34510	1.87	1608	159	6.09	3.26	5.5	2.94
PIC	1614492	4.22	82675	4501	13.91	3.30	8.62	2.04
PROG_V	1177545	1.41	63317	8808	4.84	3.43	2.98	2.11
RGB2YCRCB	15020	1.41	899	172	4.92	3.49	2.98	2.11
SPI	1527053	3.67	73541	5067	12.97	3.53	8.94	2.44
SXP	120255	0.7	5714	1758	2.66	3.80	1.58	2.26
TV_80	277028	6.72	11928	1861	27.03	4.02	18.98	2.82
USB2	58150	2.34	3625	347	9.53	4.07	4.85	2.07
VGA	163419	5.47	10397	1022	23.28	4.26	15.36	2.81

Table 2: Bencmarking Results

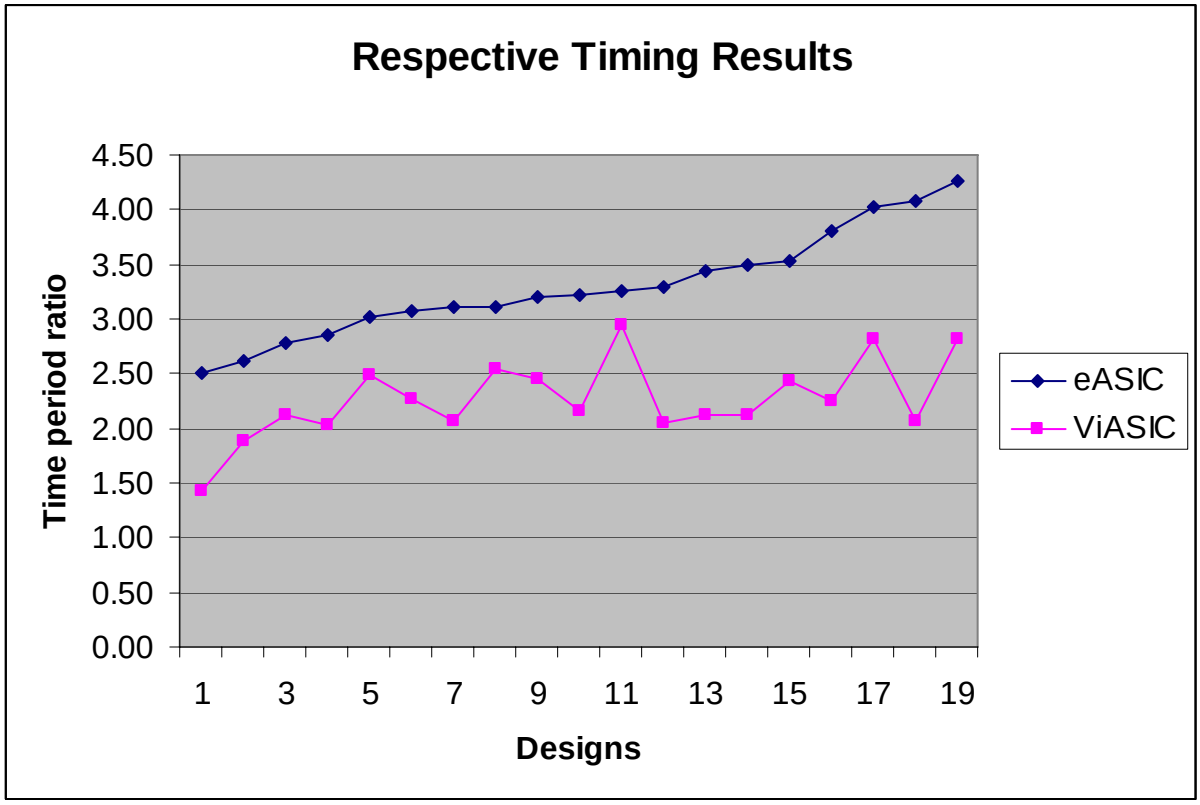


Fig 3: Timing Results

eASIC has produsedthe worst timing results because the architecture consist of LUTs and it is slower compare to the basic gates which are used in LightSpeed and ViASIC.the advantage of eASIC is that it only one layer configurable and programmability up to some extend possible by changing the bit stream of LUTs.LightSpeed is the fastest among all because it can achieve higher architecture utilization as compare to other cores as LightSpeed is 4M4V configurable.In case of LightSpeed the cost being paid in terms of Mask cost.

As per the results ViASIC is proved always better than the eASIC in terms of performance and cost.but to achieve good performance in ViASIC takes more design time than the eASIC.So time to market point of view eASIC is best among all.

Chapter: 6 Design Implementation

Design Implementaion

In bechmarking flow the idea is to find out the archirtecture suitability and goodness for various designs ranging from 10K gates to 10M gates, arithmetic designs to sequential designs.In the bencmarking flow timing constraints was not stringent but in this implementation flow timing constraints are given as per the specifications and the the duty of the engineer is to meet the specified criteria in terms of area or timing.

Generally for structured ASIC area is not the stringent requirement as core is going to occupy fix area but timing constraints are very critical in this flow as it is not easy to satisfy.Thoug there is tradeoff between area and timing but it is not always and you cannot go beyond some point as far as maximum possible frequency is concern.

Here one design is taken which is having following characterstics and specification.

Design A

1. Constraints

- HCLK = 7.5ns
- HCLK_n = 7.5ns
- EXT_CLK_clk = 15ns
- CUSTOM_CLK = 15ns
- CUSTOM_CLK_n = 15ns

2. Nature

Design is with 85K gates.

5 clock domain

Hard Macro RAM

Around 45000 nets

3. Flow

As shown in *figure 1* here same RTL code and the constraints are given to the

LightSpeed, eASIC and ViASIC Flows which have discussed in earlier chapters.After the PnR flow .def file can be extractracted. The .def file contains all theinformation related to cell placements, its connectivity and the parasitics associate with it. Post layout STA will give you the timing results and reports that can be generated so that the we can make sure the critical paths of the design can be found and verified to be within the required specifications.If paths are not within the specifications, the static timing analyzer shows the entire path so that the designer can try to fix the problem.

As shown in *figure 1* front end and backend flow we hace discussed in earlier chapter. [*refer figure 3,6,8 Chapter 5*]. In Analysis process the faulty path or problem found out and based on that some modification done in the script.In the next section all the potential problems encountered and possible remedies for the same has discussed.

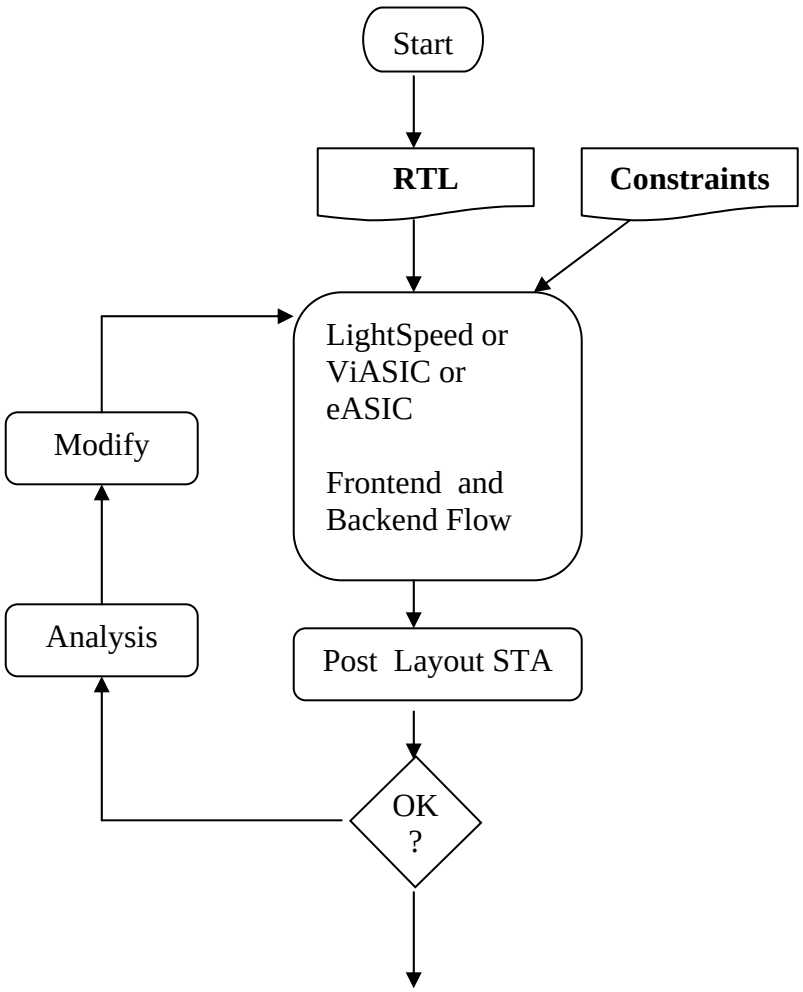


Fig 1: Implementation Flow

4. Design Issues

The main challenge lies here is to meet the timing requirements and for that designer has to follow several approches. The difficulities encounter during this design flow is discusses below.

4.1 LightSpeed

Routing Violations

In LightSpeed the routing is being done by the third party tool.here nanorouter from cadence has used.because of poor or congested placement by DesignBuilder tool routing is not possible or getting to mauch routing violations.in these cases following stratagies has used. As Placement is the starting phase of Physical synthesis it has significant impact on routing so Placement should be such a way that later Design can be easily routed

Strategy 1:

If the no of violations are very few then violations can be removed by just moving the particular set of cells. Movement of those cells should not be too much.

Strategy 2:

If no. of violations are too much and clustered i.e. concentrated on particular region then Blockage of that region for some percentage should be applied without changing Placement.
Here the shape of region defined under Blockage and percentage of Blockage is very critical.
Check Overflow and Cell density for the Violated Region which helps to

- Select suitable Blockage region
- Select suitable Blockage percentage

After applying Suitable Blockages run incremental Placement and then re route the design.

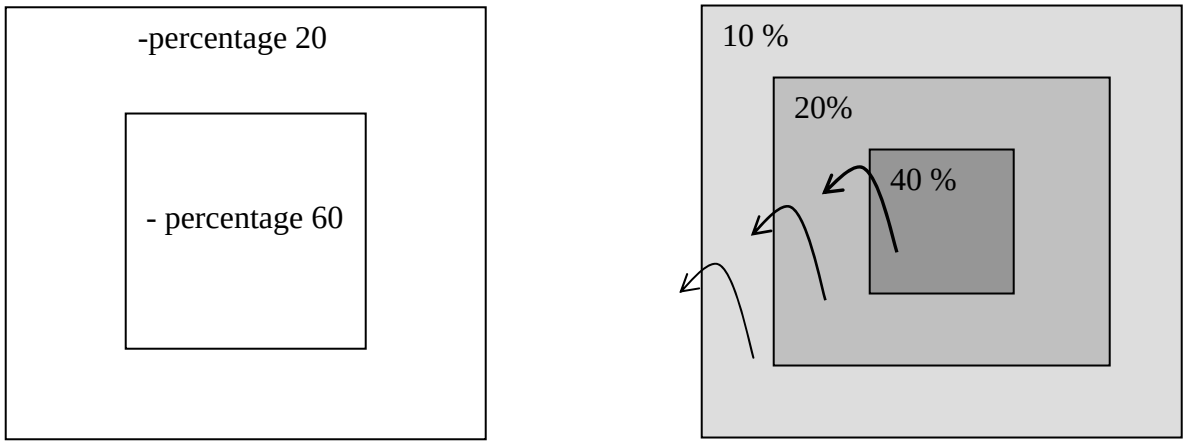


Fig 2: Hierarchical Blockage

Library bug

Found out problem in the scan flop

In normal mode

D = Q but in test mode it shows Q = D’

So synthesis tool consoder the cell functionality as Q = D but Formalty tool took Q = D’ for the formal verification. Formal verification shown the mismatch between RTL and netlist and bug get detected.

Memories

As LightSpeed architecture doesn’t support the memories so the memory Macros in the RTL code has to be modified and made new wrapper for the LightSpeed core.here memories are placed outside of the core as a Hard micro in the chip so that LightSpeed design flow will be memoryless.

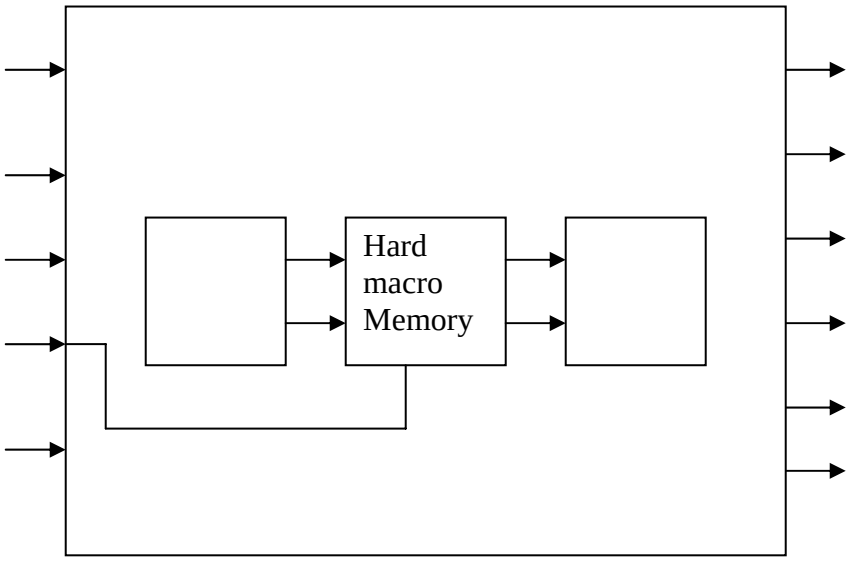


Fig 3: Original Design with RAM Macro

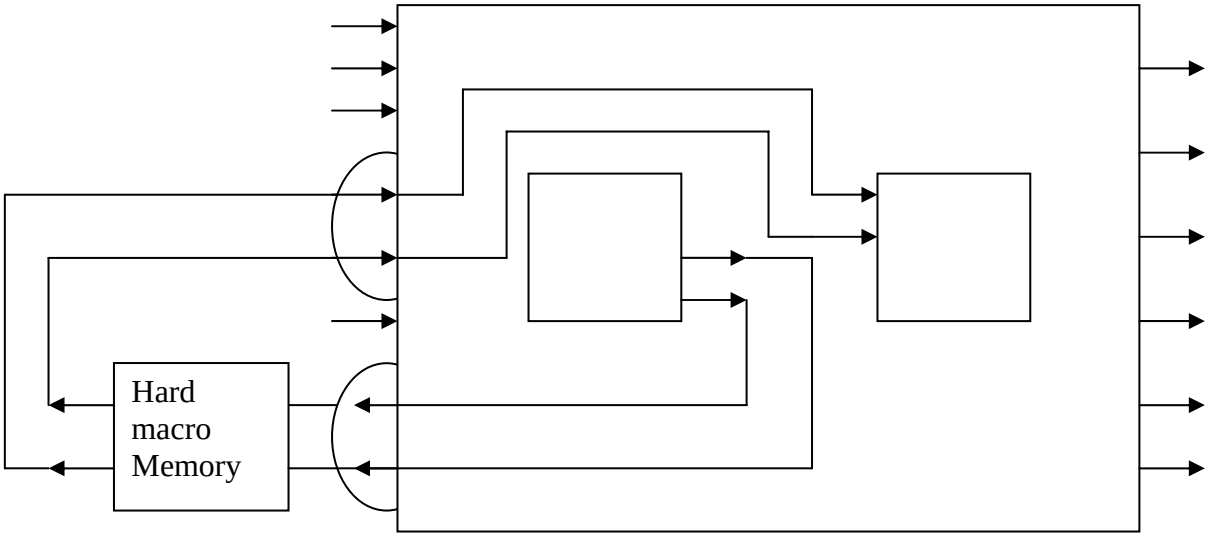


Fig 4: Modified wrapper for LS

4.2 eASIC

Asynchronous RAM

As much timing violations are found in the memory paths because of Asyncronous reset so the Asynchronous RAM has converted to the Synchronous using a flop flop. Now path will be broken into two parts so path lengh will be reduced.

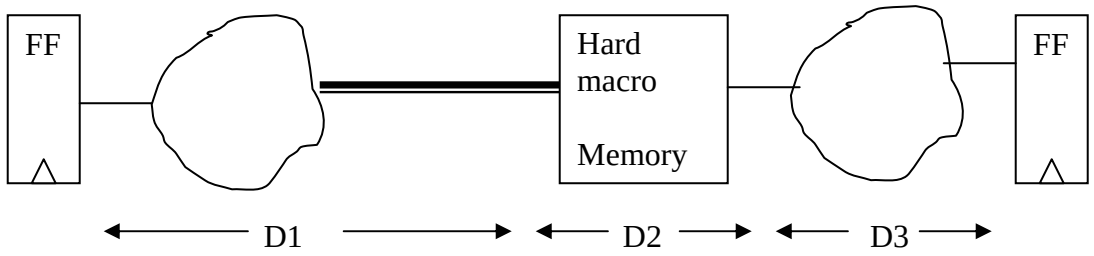


Fig 5: Asynchronous read RAM

Path length = D1 + D2 + D3

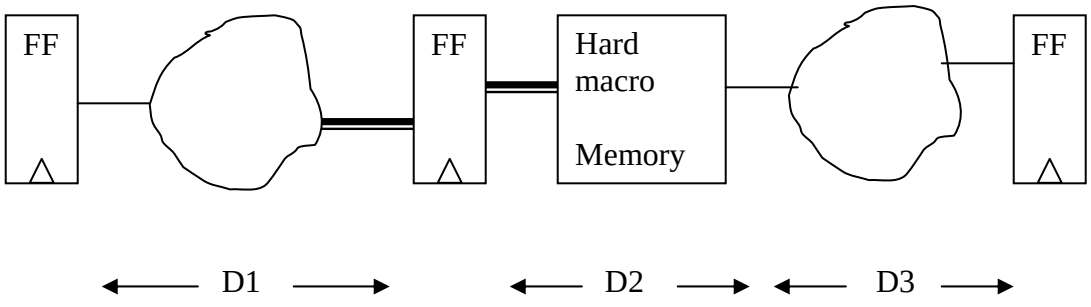


Fig 6: Async. RAM converted to Sync. Read RAM

Path1 length = D1

Path2 length = D2 + D3

Formality Checks

Formality is used for the formal verification between RTL code and generated netlist , netlist to netlist or RTL to RTL.After placement eTools are using different notations for flops so formality tool is not able to match it.

So to get formal verification checked it is required to have manual interference with the tool.So the script has written which found out the unmatched point and modify register name as specified in the script so that the formality can find the proper match point.
Here it needs to again and again until you get the match or found real unmatched point.

Exp.
Following name is generated after synthesis but it is not matched with the netlist name.

Name in netlist:
/WORK/EASIC_MACRO/iDESIGN_TOP_u_Ahb2AhbAsync32_uAsyncMaster32_HADDRM_reg [0]/dff_ecell_u1/dff_1_dffx/C1/OUT_reg

Name in RTL
r:/WORK/EASIC_MACRO/iDESIGN_TOP/u_Ahb2AhbAsync32/uAsyncMaster32/HADDRM_reg[0] i
So first potential problem found out and according to that name has changed and mentioned in the formality script.

Set_user_match
r:/WORK/EASIC_MACRO/iDESIGN_TOP/u_Ahb2AhbAsync32/uAsyncMaster32/HADDRM_reg[0] i

/WORK/EASIC_MACRO/iDESIGN_TOP_u_Ahb2AhbAsync32_uAsyncMaster32_HADDRM_reg [0]/dff_ecell_u1/dff_1_dffx/C1/OUT_reg

4.3 ViASIC

In ViASIC the driving strength doubled by connecting two cells in parallel so two different nets Make one nets at output as shown in *figure 7*.

At the time of STA prime time infer the differn net delay on the nets so it gives warning but actually nothing wrong in that.so this issue is reported and still under process and reported to the ViASIC technical team working with us.

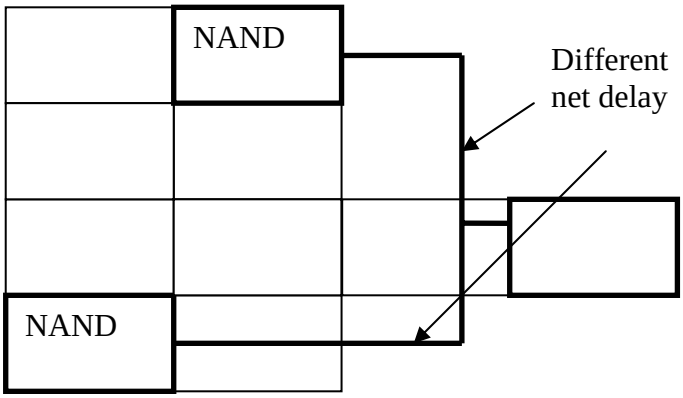


Fig 7: Multidriven nets in ViASIC

Here clock is not prefabricated so at the time of clock tree synthesis same problem occurs with the clock net also.

The proposed solution is to create combine wrapper for both of the cell and represent both as one combine cell to the Primetime.

4.4 General Issues

In the design the longest paths are from primary inputs to the register and register to the primary outputs [refer figure]

Approach 1:

We can overconstraint the design and can achieve desired timings. The problem with this approach is that unnecessarily it overconstraints the reg to reg paths and synthesis tool will try to optimize the whole paths.This approache leads to higher design area.

Exp .If we want 10ns clock period, we can give clock constrints of 8ns to the synthesis tool. And Synthesis tool will optimize whole path

Approach 2:

By inserting the dummy clock (pis and pos) we can group the input and output path separately.Now tool will treat this path under different clocks.now designer can give different constraints to the pis and pos clock paths.[*refer figure 9*]

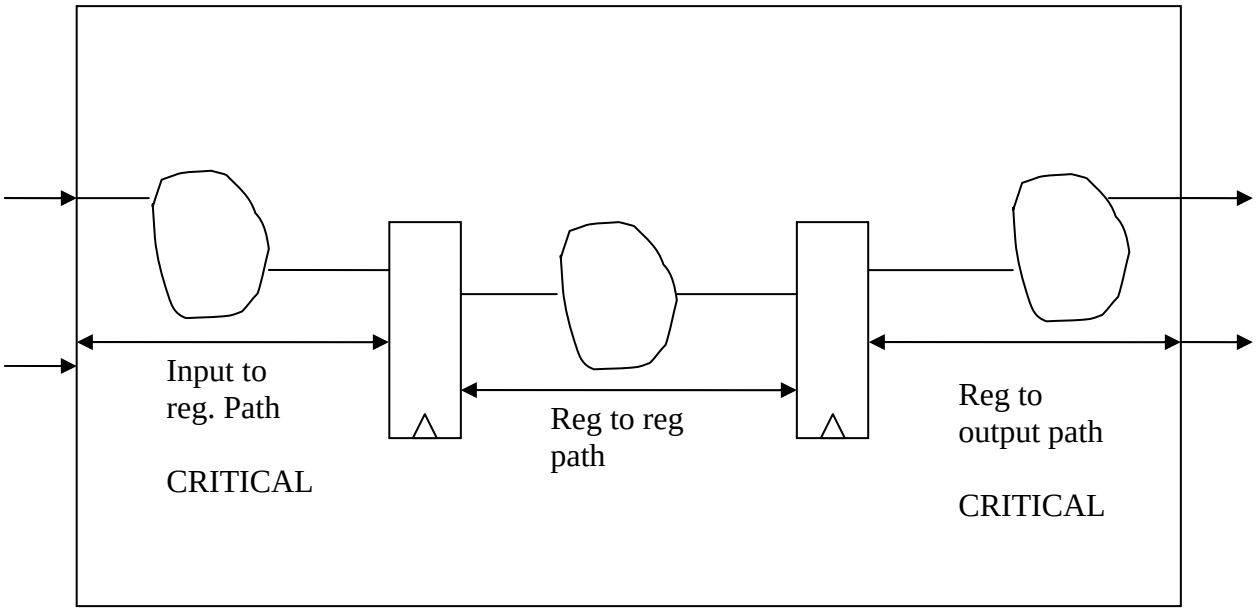


Fig 8: Timing Paths

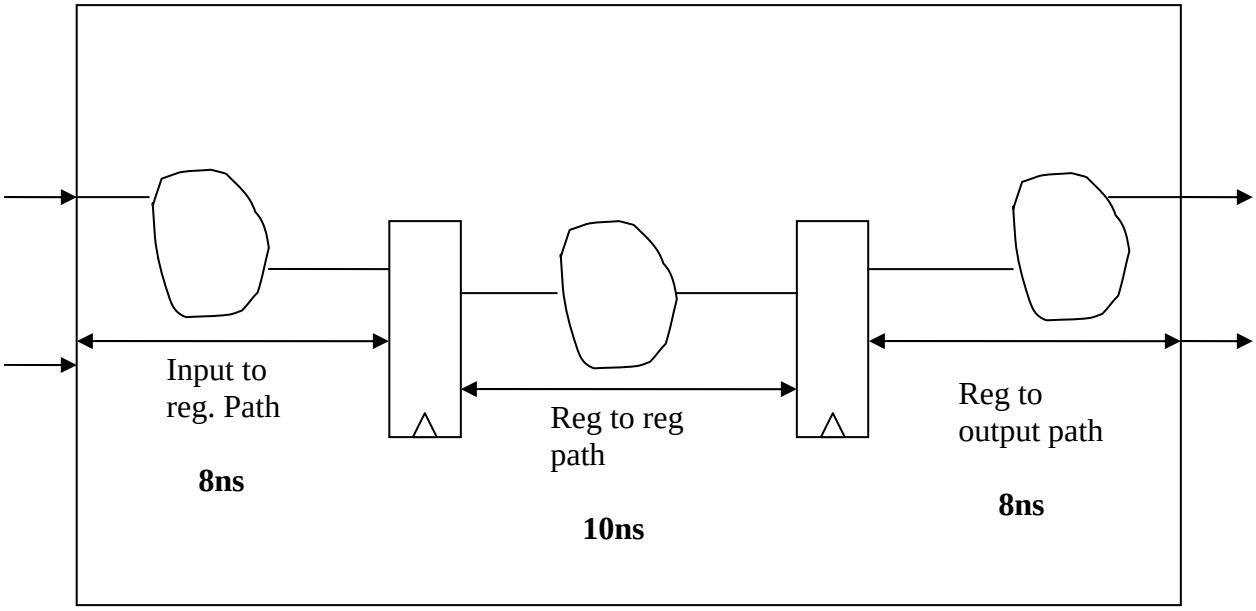


Fig 9: Overconstrainted I/O paths

4.5 Results and Comparision

ViASIC

Path Group	Worst Slack (ns)	Constraints (ns)
CUSTOM_CLK	**-0.51**	**15**
CUSTOM_CLK_n	**5.51**	**15**
HCLK_clk	**-0.23**	**7.5**
HCLK_clk_n	**4.73**	**7.5**
EXT_CLK_clk	**10.87**	**15**

Table 1: ViASIC Results

- Total Logic Cells = 23.04K
- Tile = 256 Tile size = 10*9
- Utilized Logic Cells = 8780
- Tile Size = 9x10
- Percentage Utilization = 38.11 %

After so many iterations the minimum negative slack observed is -0.51ns in the reg to reg path.so this design implementation on ViASIC gives maximum performance of 14.4 ns clock period (CUSTOM_CLK).

eASIC

Path Group	Worst Slack (ns)	Constraints (ns)
CUSTOM_CLK	**-2.67**	**15**
CUSTOM_CLK_n	**5.20**	**15**
HCLK_clk	**-1.23**	**7.5**
HCLK_clk_n	**3.92**	**7.5**
EXT_CLK_clk	**9.42**	**15**

Table 2: eASIC results

- eCore = 8
- Total eCells = 8K
- Utilized eCells = 4.3K
- Percentage Utilization = 53.7 %

LightSpeed

Path Group	Worst Slack (ns)	Constraints (ns)
CUSTOM_CLK	**0.02**	**15**
CUSTOM_CLK_n	**5.62**	**15**
HCLK_clk	**0.12**	**7.5**
HCLK_clk_n	**4.22**	**7.5**
EXT_CLK_clk	**11.21**	**15**

Table 3: LightSpeed result

- Total Logic Tile = 35 * 30 = 1050
- Utilized Logic Tile = 767
- Utilizatio = 72.9 %

As shown in table LightSpeed implementation met the constrain. The reason being 4M4V configurability in this case so the place can place blocks nearby and the net delay can reduce significantly.As technology shrinking the net delay become more significant to the gate delay.The advantage with the LightSpeed is that it has less net delay compare to ViASIC.

Core	Spesifications MHz	Post Route STA	Conf. Layers	Utilization
LightSpeed	**66.7(15ns)**	**66.7 (15ns)**	**4**	**72.9%**
eASIC	**66.7(15ns)**	**55.6(18ns)**	**1**	**53.8%**
VIASIC	**66.7(15ns)**	**62.5(16ns)**	**1**	**37.6%**

Table 4: Overall result

Chapter: 7 Conclusion

It is not the purpose of this report to suggest that structured ASICs will displace any of the existing implementation technologies. However, structured ASICs are closer to FPGAs in terms of the low costs and fast turnaround associated with creating a design. At the same time, they are much closer to standard cell implementations in terms of capacity, performance, low power consumption, and low per-unit costs for medium-low to medium-high production runs.

The benchmarking results show that there is clear cut trade off between the mask cost and the performance.In LightSpeed achived performace is better than the others because it is 2M2V configurable.As eASIC is LUT base its design flow is simplest but produced worst timing results so the nature of eASIC is near to the FPGA.LightSpeed is near to the ASIC as its performance is good and 4 configurable layers. For Structured ASIC design flow designer has to work with the different third party tools and ASIC tools unlike FPGA and ASIC technology so that design flow for structured ASIC is complex than the ASIC and FPGA design flow.Designer has to judge the suitability of the architecture before design implementation on particular SA.At the time of implementation issues of Hard Macros, Memories, buffers must be taken into account.

Currently Structured ASIC actually supports 400Kgates (LightSpeed) implementation which is very good and this figure further expected to increase.Cost wise mid volume designs (Hundreds of Kilo Gates) are now becoming exclusively for Structured ASIC.

Chapter : 8 Future Scope

- Compare the performance of Structured ASIC, Standard ASIC and FPGA with given timing constraints and find out maximum possible operation frequency.So we can have technology comparision and feel of the Structured ASIC.
- Design flow and performance can be further improved using different synthesis tools lile RTL compiler (Cadence) and Magma blast_create.
- Automized buffer insertion flow for eASIC.

References

1. T. Okamoto, T. Kimoto, N. Maeda, “Design Methodology and Tools for NEC Electronics - Structured ASIC ISSP", [p. 90] Proceeding of the 2004 international symposium on Physical design
2. B. Zahiri, “Structured ASICs: Opportunities and Challenges,” Proceedings of the 21st International Conference on Computer Design (ICCD’03)
3. K. Wu, Y. Tsai, “Structured ASIC, Evolution or Revolution?” Faraday Technology Corporation, Proceedings of the 2004 International Symposium on Physical Design
4. J. Brown, R. Packer, J. Prasad, K. Kofford, T. Dye, and B. Kirk, “Hybrid Approach to Structured ASICS for Minimizing the Impact of Reticle Costs and Interconnect Delay” AMI Semiconductor, lEEE 2004 Custom Integrated Circuits Conference.
5. N. Sherwani, ”Algorithms for VLSI Physical Design Automation”,Kluwer Academic Publishers.
6. Janick Bergeron, “Writing testbenches and functional verification of HDL model”, Kluwer Academic Publishers.
7. www.st.com
8. www.easic.com
9. www.xilinx.com
10. www.altera.com
11. www.lightspeed.com
12. www.fpgajournal.com
13. www.electronicsweekly.com
14. User Guide - TCL, PERL.
15. Tool user manuals - Formality, Design Compiler, eTools, ViPath, DesignBuilder, Nanorouer.